Westfälische Wilhelms-Universität Münster

PLDI Orlando 2023

Array Programming via Multi-Dimensional Homomorphisms

Ari Rasch, Richard Schulze, Sergei Gorlatch

University of Münster, Germany

Existing Work

Array programming is at the heart of popular, existing approaches:

Class	*Popular Examples*	*Performance*	*Portability*	*Productivity*
Scheduling	`TVM, Halide, Fireiron`	✔	often require re-design/extension for new architectures	incorporate user into optimization process
Polyhedral	`TC, PPCG, Pluto`	struggle with reductions (e.g., dot in MatMul)	transformations chosen toward particular architectures and data characteristics	✔
Functional	`Lift`	✔	transformations designed toward particular architectures and data characteristics	often incorporate user into optimization process
Domain-Specific	`cuBLAS, oneMKL`	✔	hand-optimized toward particular architecture and data characteristics	(✔)
Higher-Level	*`Futhark, Dex, ATL, Yang et al.[POPL'21], …`*	*We consider these approaches as greatly combinable with our approach 👍 (as frontends)*		

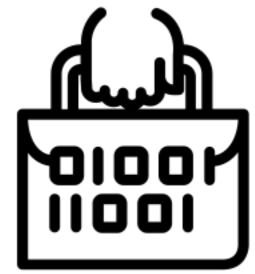
The existing approaches struggle with achieving <u>simultaneously</u>
Performance & Portability & Productivity

Existing Work

Why is **simultaneously** achieving **Performance** & **Portability** & **Productivity** challenging:

Performance

- hardware-specific code generation & optimization: exploiting core & memory hierarchy
- data-specific code generation & optimization: respecting data layouts and sizes
- …



Portability

- flexible in hardware/data-specific optimizations (*performance portability*)
- easily extensible toward new target architectures (*functional portability*)
- …

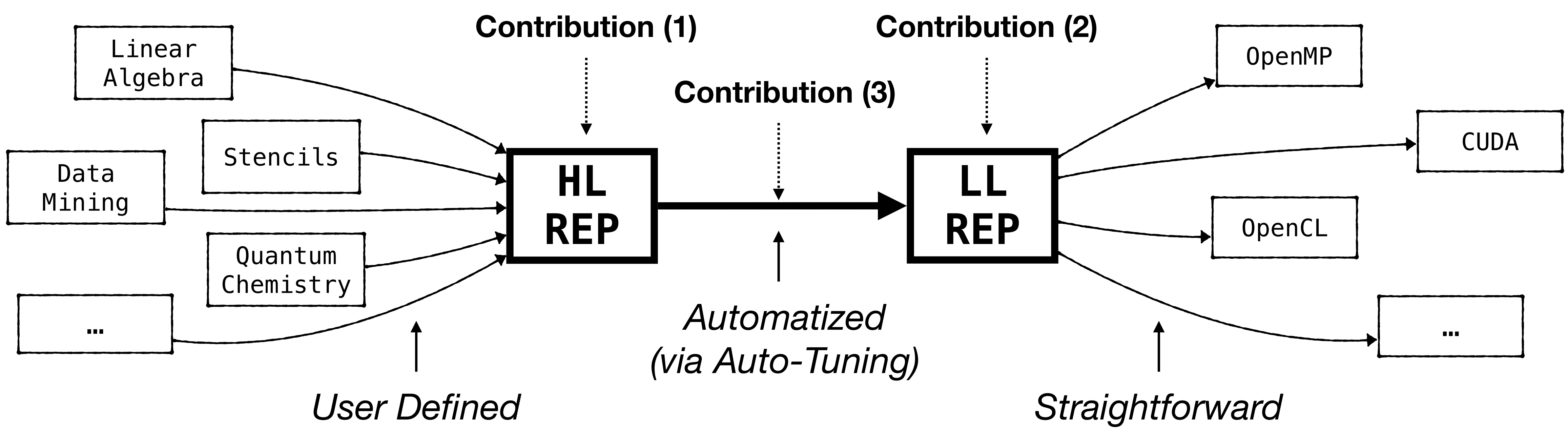
Productivity

- freeing programmer from hardware and optimization details
- supporting wide range of target computations (*expressivity*)
- …

Achieving <u>simultaneously</u>
Performance & Portability & Productivity
poses challenging, often even contradicting, requirements

Goal of this Work

Achieving ***Performance*** & ***Portability*** & **Productivity** in **one approach**, based on three major contributions:



1. ***High-Level Functional Representation*** that is formally defined, and that *expresses data-parallel computations* on a high-level of abstraction while still capturing all information relevant for generating high performing code

2. ***Low-Level Functional Representation*** that enables formally *expressing and reasoning about optimizations* for the memory and core hierarchies of state-of-the-art architectures, and that can be straightforwardly *transformed to executable program code* (e.g., in OpenMP, CUDA, OpenCL, …)

3. ***Lowering Process*** that *fully automatically* lowers our *high-level representation* to an hardware/data-optimized instance of our *low-level representation,* in a formally sound and automatically optimizable (auto-tunable) manner

Agenda

1. **<u>Contribution 1:</u>** High-Level Representation

2. **<u>Contribution 2:</u>** Low-Level Representation

3. **<u>Contribution 3:</u>** Lowering: *High-Level Representation* → *Low-Level Representation*

4. Experimental Results (Performance & Portability & Productivity)

5. Conclusion

6. Future Work

High-Level Representation

Goals:

1. **Uniform**:
   should be able to express any kind of data-parallel computation, without relying on computation-specific building blocks, extensions, etc.
2. **Minimalistic**:
   should rely on less building blocks to keep language small and simple
3. **Structured**:
   avoiding compositions and nestings of building blocks as much as possible, thereby further contributing to usability and simplicity of our language

$$\texttt{MatVec}^{<T\in\texttt{TYPE}\,|\,I,K\in\mathbb{N}>} := \texttt{out\_view<T>(}\; w:(i,k)\mapsto(i)\;\texttt{)} \circ \texttt{md\_hom<I,K>(}\;*,\,(+\!\!\!+,+)\;\texttt{)} \circ \texttt{inp\_view<T,T>(}\; M:(i,k)\mapsto(i,k)\,,\, v:(i,k)\mapsto(k)\;\texttt{)}$$

***High-Level Representation of* MatVec**

High-Level Representation

Quick Reminder

<u>Data-parallel computations:</u> how are they characterized?

1. applying the *same function* f to each point in a multi-dimensional grid of data
2. combining the obtained results in the grid's different dimensions using *combine operators*

<u>`Matrix-Vector Multiplication:`</u>

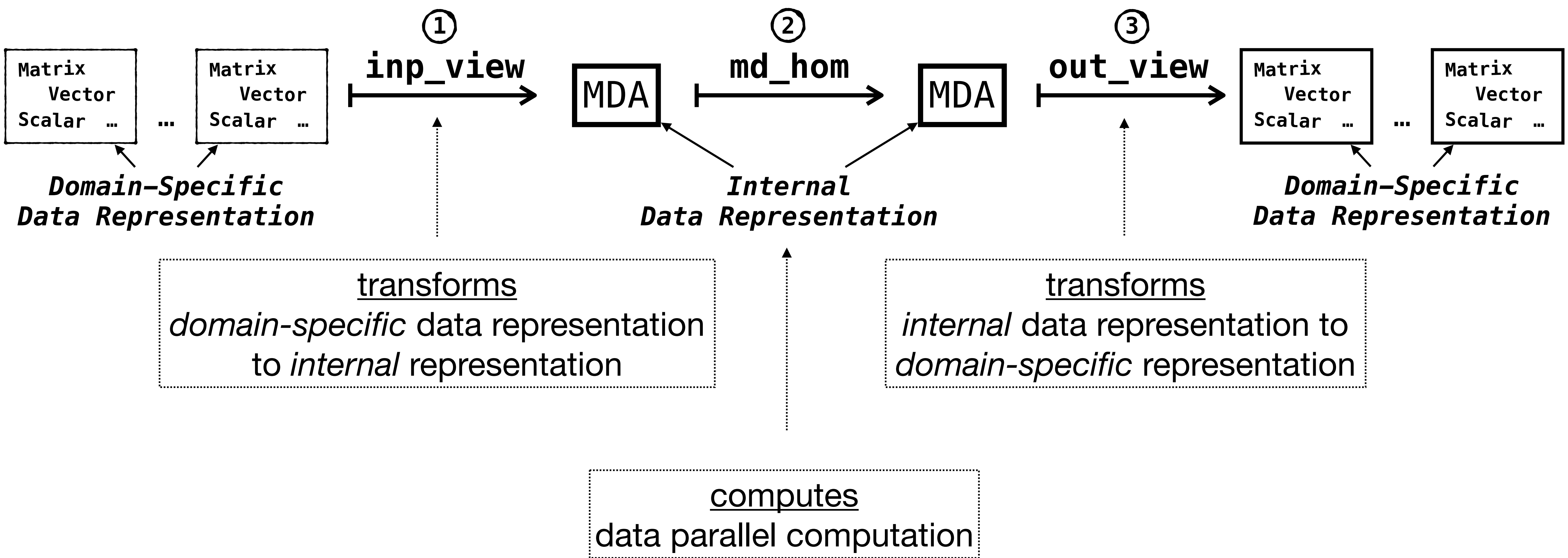
$$\begin{pmatrix} M_{1,1} & \dots & M_{1,K} \\ \vdots & \ddots & \vdots \\ M_{I,1} & \dots & M_{I,K} \end{pmatrix}, \begin{pmatrix} v_1 \\ \vdots \\ v_K \end{pmatrix} \overset{\texttt{MatVec}}{\mapsto} \begin{array}{c} \xrightarrow{\hspace{8em}\circledast_2} \\ \left.\begin{array}{ccc} f(M_{1,1}, v_1) & \dots & f(M_{1,K}, v_K) \\ \vdots & \ddots & \vdots \\ f(M_{I,1}, v_1) & \dots & f(M_{I,K}, v_K) \end{array}\right\downarrow \circledast_1 \end{array} = \begin{pmatrix} M_{1,1} * v_1 + \ldots + M_{1,K} * v_K \\ \vdots \\ M_{I,1} * v_1 + \ldots + M_{I,K} * v_K \end{pmatrix} = \begin{pmatrix} w_1 \\ \vdots \\ w_I \end{pmatrix}$$

<u>`Jacobi 1D:`</u>

$$\begin{pmatrix} v_1 \\ \vdots \\ v_N \end{pmatrix} \overset{\texttt{Jacobi1D}}{\mapsto} \left.\begin{array}{c} f(v_1, v_2, v_3) \\ f(v_2, v_3, v_4) \\ \vdots \end{array}\right\downarrow \circledast_1 = \begin{pmatrix} c * (v_1 + v_2 + v_3) \\ c * (v_2 + v_3 + v_4) \\ \vdots \end{pmatrix} = \begin{pmatrix} w_1 \\ \vdots \\ w_{N-2} \end{pmatrix}$$

High-Level Representation

<u>Overview:</u>



Our high-level representation expresses data-parallel computations
— *agnostic from hardware and optimization details* —
using exactly three higher-order functions only

High-Level Representation

Example: `MatVec` expressed in *MDH's High-Level Representation*[1]

```
MatVec<T∈TYPE|I,K∈ℕ> :=  out_view<T>( w:(i,k)↦(i) ) ∘
                             md_hom<I,K>( *, (⧺,+) ) ∘
                                 inp_view<T,T>( M:(i,k)↦(i,k) , v:(i,k)↦(k) )
```

***High-Level Representation of* `MatVec`**

What is happening here:

- `inp_view` captures the accesses to input data
- `md_hom` expresses the data-parallel computation
- `out_view` captures the accesses to output data

[1] *We can generate such MDH expressions also automatically from straightforward (annotated) C code [IMPACT'19]*

High-Level Representation

md_hom	f	$\circledast_1$	$\circledast_2$	$\circledast_3$	$\circledast_4$
Dot	*	+	/	/	/
MatVec	*	++	+	/	/
MatMul	*	++	++	+	/
MatMul^T	*	++	++	+	/
bMatMul	*	++	++	++	+

Views	inp_view		out_view
	A	B	C
Dot	(k) $\mapsto$ (k)	(k) $\mapsto$ (k)	(k) $\mapsto$ ()
MatVec	(i,k) $\mapsto$ (i,k)	(i,k) $\mapsto$ (k)	(i,k) $\mapsto$ (i)
MatMul	(i,j,k) $\mapsto$ (i,k)	(i,j,k) $\mapsto$ (k,j)	(i,j,k) $\mapsto$ (i,j)
MatMul^T	(i,j,k) $\mapsto$ (k,i)	(i,j,k) $\mapsto$ (j,k)	(i,j,k) $\mapsto$ (j,i)
bMatMul	(b,i,j,k) $\mapsto$ (b,i,k)	(b,i,j,k) $\mapsto$ (b,k,j)	(b,i,j,k) $\mapsto$ (b,i,j)

1) Linear Algebra Routines

md_hom	f	$\circledast_1$	$\circledast_2$	$\circledast_3$	$\circledast_4$	$\circledast_5$	$\circledast_6$	$\circledast_7$	$\circledast_8$	$\circledast_9$	$\circledast_{10}$
Conv2D	*	++	++	+	+	/	/	/	/	/	/
MCC	*	++	++	++	++	+	+	+	/	/	/
MCC_Capsule	*	++	++	++	++	+	+	+	++	++	+

Views	inp_view		out_view
	I	F	O
Conv2D	(p,q,r,s) $\mapsto$ (p+r,q+s)	(p,q,r,s) $\mapsto$ (r,s)	(p,q,r,s) $\mapsto$ (p,q)
MCC	(n,p,...) $\mapsto$ (n,p+r,q+s,c)	(n,p,...) $\mapsto$ (k,r,s,c)	(n,p,...) $\mapsto$ (n,p,q,k)
MCC_Capsule	(n,p,...) $\mapsto$ (n,p+r,q+s,c,cm,ck)	(n,p,...) $\mapsto$ (k,r,s,c,ck,cn)	(n,p,...) $\mapsto$ (n,p,q,k,cm,cn)

2) Convolution Stencils

md_hom	f	$\circledast_1$	$\circledast_2$
MBBS	id	$++_{\texttt{prefix-sum}}(+)$	+

Views	inp_view	out_view
	A	Out
MBBS	(i,j) $\mapsto$ (i,j)	(i) $\mapsto$ (i)

8) Maximum Bottom Box Sum

md_hom	f	$\circledast_1$	$\circledast_2$
Jacobi1D	J_{1D}	++	/
Jacobi2D	J_{2D}	++	++

Views	inp_view	out_view
	I	O
Jacobi1D	(i) $\mapsto$ (i+0) , (i) $\mapsto$ (i+1) , (i) $\mapsto$ (i+2)	(i) $\mapsto$ (i)
Jacobi2D	(i,j) $\mapsto$ (i,j+1) , (i,j) $\mapsto$ (i+1,j) , ...	(i,j) $\mapsto$ (i,j)

3) Jacobi Stencils

md_hom	f	$\circledast_1$	$\circledast_2$
PRL	wght	++	max_{PRL}

Views	inp_view		out_view
	N	E	M
PRL	(i,j) $\mapsto$ (i)	(i,j) $\mapsto$ (j)	(i,j) $\mapsto$ (i)

4) Probabilistic Record Linkage

md_hom	f	$\circledast_1$
map(f)	f	++
reduce($\oplus$)	id	$\oplus$
reduce($\oplus$,$\otimes$)	(x) $\mapsto$ (x,x)	($\oplus$,$\otimes$)

Views	inp_view	out_view	
	I	O_1	O_2
map(f)	(i) $\mapsto$ (i)	(i) $\mapsto$ (i)	/
reduce($\oplus$)	(i) $\mapsto$ (i)	(i) $\mapsto$ ()	/
reduce($\oplus$,$\otimes$)	(i) $\mapsto$ (i)	(i) $\mapsto$ ()	(i) $\mapsto$ ()

6) Map/Reduce Patterns

md_hom	f	$\circledast_1$
scan($\oplus$)	id	$++_{\texttt{prefix-sum}}(\oplus)$

Views	inp_view	out_view
	I	O
scan($\oplus$)	(i) $\mapsto$ (i)	(i) $\mapsto$ (i)

7) Scan Pattern

md_hom	f	$\circledast_1$	$\circledast_2$
Histo	f_{Histo}	++	+

Views	inp_view		out_view
	Bins	Elems	Out
Histo	(b,e) $\mapsto$ (b)	(b,e) $\mapsto$ (e)	(b,e) $\mapsto$ (b)

5) Histogram

Our high-level representation is capable of expressing the various kinds of data-parallel computations which often differ in major characteristics

Summary: High-Level Representation

- **Uniform**: for all the different kinds of data-parallel computations, by exploiting the common algebraic properties of data-parallel computations (rather than relying on domain-specific building blocks)
- **Minimalistic**: by relying on exactly three higher-order functions only
- **Structured**:
  - `inp_view` prepares the *input data*
  - `md_hom` computes *data-parallel computation*
  - `out_view` prepares the *output data*
- Full **formal** foundation

Definition 5 (Buffer). Let $T \in \mathtt{TYPE}$ be an arbitrary scalar type, $D \in \mathbb{N}_0$ a natural number[9], and $N := (N_1, \ldots, N_D) \in \mathbb{N}^D$ a sequence of natural numbers.

A *Buffer (BUF)* $\mathfrak{b}$ that has *dimensionality* D, *size* N, and *scalar type* T is a function with the following signature:

$$\mathfrak{b} : [0, N_1)_{\mathbb{N}_0} \times \ldots \times [0, N_D)_{\mathbb{N}_0} \to T \cup \{\perp\}$$

Here, $\perp$ denotes the *undefined* value. We refer to $[0, N_1)_{\mathbb{N}_0} \times \ldots \times [0, N_D)_{\mathbb{N}_0} \to T \cup \{\perp\}$ as the *type* of BUF $\mathfrak{b}$, which we also denote as $T^{N_1 \times \ldots \times N_D}$, and we refer to set $\mathtt{BUF\text{-}IDX\text{-}SETs} := \{ [0, N)_{\mathbb{N}_0} \mid N \in \mathbb{N} \}$ as *BUF index sets*. Analogously to Notation 1, we write $\mathfrak{b}[\, i_1, \ldots, i_D \,]$ instead of $\mathfrak{b}(i_1, \ldots, i_D)$ to avoid a too heavy usage of parentheses.

Definition 5 (Buffer). Let $T \in \mathtt{TYPE}$ be an arbitrary scalar type, $D \in \mathbb{N}_0$ a natural number[9], and $N := (N_1, \ldots, N_D) \in \mathbb{N}^D$ a sequence of natural numbers.

A *Buffer (BUF)* $\mathfrak{b}$ that has *dimensionality* D, *size* N, and *scalar type* T is a function with the following signature:

$$\mathfrak{b} : [0, N_1)_{\mathbb{N}_0} \times \ldots \times [0, N_D)_{\mathbb{N}_0} \to T \cup \{\perp\}$$

Definition 1 (Multi-Dimensional Array). Let $\mathtt{MDA\text{-}IDX\text{-}SETs} := \{ I \subset \mathbb{N}_0 \mid |I| < \infty \}$ be the set of all finite subsets of natural numbers, to which we also refer as set of *MDA index sets*. Let further $T \in \mathtt{TYPE}$ be an arbitrary scalar type, $D \in \mathbb{N}$ a natural number, $I := (I_1, \ldots, I_D) \in \mathtt{MDA\text{-}IDX\text{-}SETs}^D$ a sequence of D-many MDA index sets, and $N := (N_1, \ldots, N_D) := (|I_1|, \ldots, |I_D|)$ the sequence of index sets' sizes.

A *Multi-Dimensional Array (MDA)* $\mathfrak{a}$ that has *dimensionality* D, *size* N, *index sets* I, and *scalar type* T is a function with the following signature:

$$\mathfrak{a} : I_1 \times \ldots \times I_D \to T$$

We refer to $I_1 \times \ldots \times I_D \to T$ as the *type* of MDA $\mathfrak{a}$.

Definition 2 (Combine Operator). Let $\mathtt{MDA\text{-}IDX\text{-}SETs} \,\dot{\times}\, \mathtt{MDA\text{-}IDX\text{-}SETs} := \{ (P, Q) \in \mathtt{MDA\text{-}IDX\text{-}SETs} \times \mathtt{MDA\text{-}IDX\text{-}SETs} \mid P \cap Q = \varnothing \}$ denote the set of all pairs of MDA index sets that are disjoint. Let further $\Rightarrow^{\mathtt{MDA}}_{\mathtt{MDA}} : \mathtt{MDA\text{-}IDX\text{-}SETs} \to \mathtt{MDA\text{-}IDX\text{-}SETs}$ be a function on MDA index sets, $T \in \mathtt{TYPE}$ a scalar type, $D \in \mathbb{N}$ an MDA dimensionality, and $d \in [1, D]_{\mathbb{N}}$ an MDA dimension.

We refer to any binary function $\circledast$ of type (parameters in angle brackets are type parameters)

$$\circledast^{<(I_1, \ldots, I_{d-1}, I_{d+1}, \ldots, I_D) \in \mathtt{MDA\text{-}IDX\text{-}SETs}^{D-1},\ (P,Q) \in \mathtt{MDA\text{-}IDX\text{-}SETs} \,\dot{\times}\, \mathtt{MDA\text{-}IDX\text{-}SETs}>} :$$

$$T[\, I_1, \ldots, \underbrace{\Rightarrow^{\mathtt{MDA}}_{\mathtt{MDA}}(P)}_{\substack{\uparrow \\ d}}, \ldots, I_D \,] \times T[\, I_1, \ldots, \underbrace{\Rightarrow^{\mathtt{MDA}}_{\mathtt{MDA}}(Q)}_{\substack{\uparrow \\ d}}, \ldots, I_D \,] \to T[\, I_1, \ldots, \underbrace{\Rightarrow^{\mathtt{MDA}}_{\mathtt{MDA}}(P \cup Q)}_{\substack{\uparrow \\ d}}, \ldots, I_D \,]$$

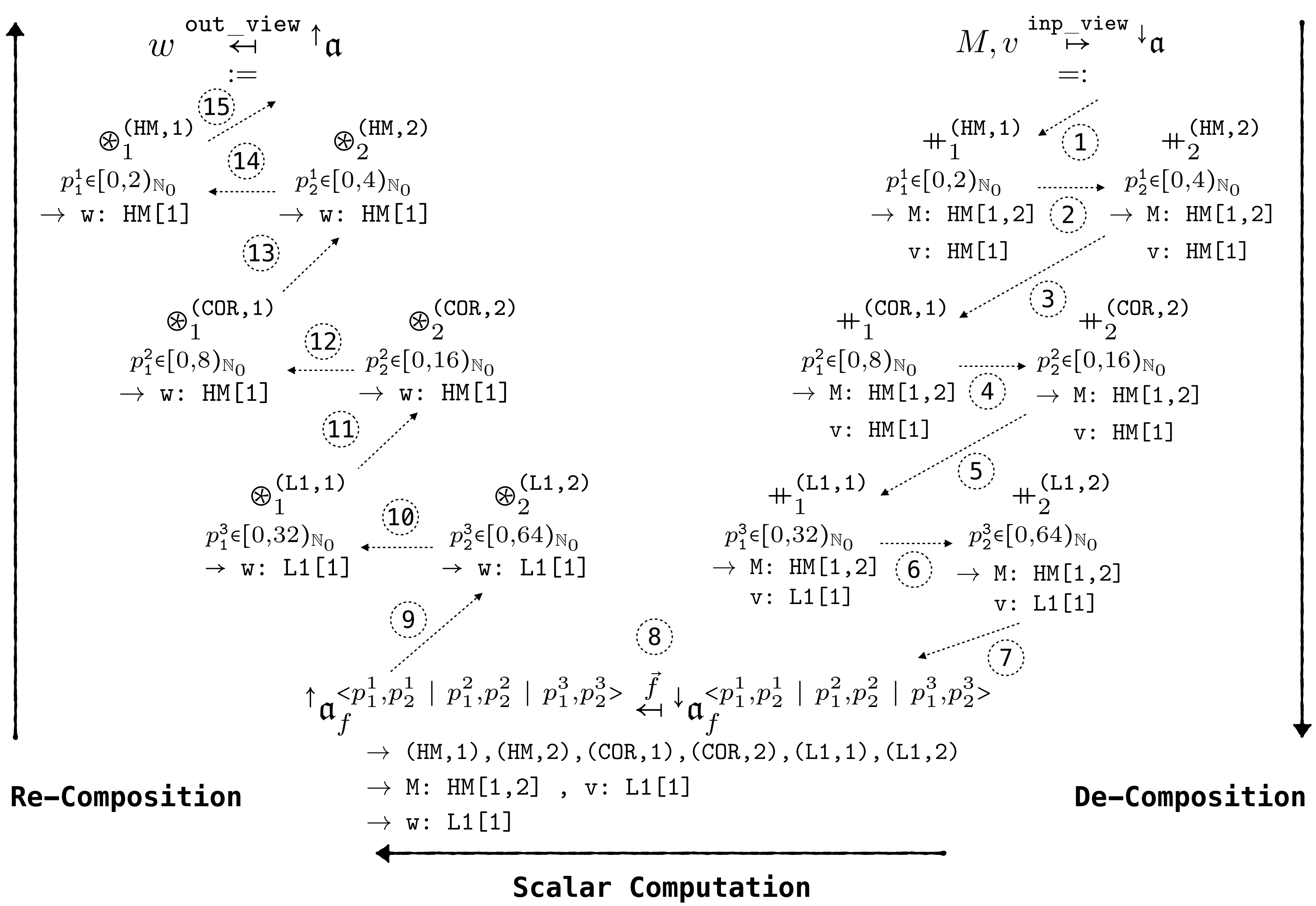
as *combine operator* that has *index set function* $\Rightarrow^{\mathtt{MDA}}_{\mathtt{MDA}}$, *scalar type* T, *dimensionality* D, and *operating dimension* d. We denote combine operator's type concisely as $\mathtt{CO}^{<\Rightarrow^{\mathtt{MDA}}_{\mathtt{MDA}} \mid T \mid D \mid d>}$.

...

Low-Level Representation

Goals:

1. Expressing a hardware- & data-optimized d*e-composition* and *re-composition* of data-parallel computations
2. Being straightforwardly transformable to executable program code (e.g., in OpenMP, CUDA, and OpenCL) — optimization decisions are explicitly expressed in low-level representation !

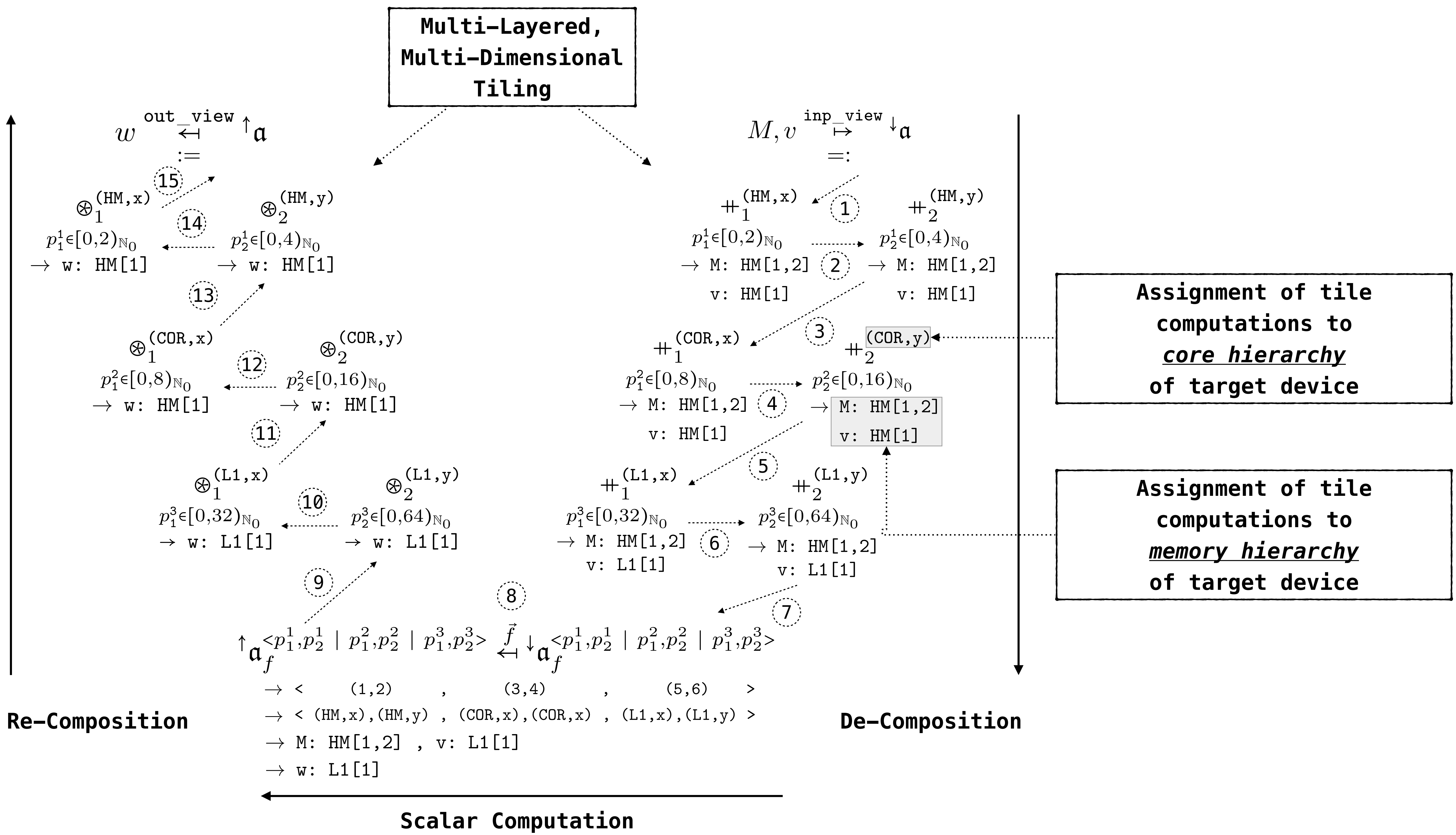


Low-Level Representation of MatVec

(particular, straightforward instance, *for an artificial architecture)*

Low-Level Representation

How do we express *(De/Re)-Compositions* in our low-level representation:



Low-Level Representation

Code generation:

```
// 0.1.2. combine operators

  // pre-implemented combine operators

  // inverse concatenation
  ∀d ∈ ℕ:
  cc_inv<<d>><I_1, ..., I_{d-1}, I_{d+1}, ..., I_D ∈ MDA-IDX-SETs, (P,Q) ∈ MDA-IDX-SETs ×̇ MDA-IDX-SETs>(
    T^INP[I_1, ..., I_{d-1}, id(P ∪̇ Q), I_{d+1}, ..., I_D] res ) -> ( T^INP[I_1, ..., I_{d-1}, id(P) , I_{d+1}, ..., I_D] lhs ,
                                                                  T^INP[I_1, ..., I_{d-1}, id(Q) , I_{d+1}, ..., I_D] rhs )
  {
    int i_1 ∈ I_1
      ⋱
        int i_{d-1} ∈ I_{d-1}
          int i_{d+1} ∈ I_{d+1}
            ⋱
              int i_D ∈ I_D
              {
                int i_d ∈ P
                  res[ i_1,...,i_d,...,i_D] =: lhs[ i_1,...,i_d,...,i_D];
                int i_d ∈ Q
                  res[ i_1,...,i_d,...,i_D] =: rhs[ i_1,...,i_d,...,i_D];
              }
  }
```

Listing 11. Pre-Implemented Combine Operators

```
// 3. re-composition phase

  // 3.1. main

  int p_ σ_{↑-ord}(1,1) ∈<↔_{↑-ass}(1,1)> #PRT( σ_{↑-ord}(1,1) )
  {
    int p_ σ_{↑-ord}(1,2) ∈<↔_{↑-ass}(1,2)> #PRT( σ_{↑-ord}(1,2) )
    {
      ⋱
        int p_ σ_{↑-ord}(L,D) ∈<↔_{↑-ass}(L,D)> #PRT( σ_{↑-ord}(L,D) )
        {
          ll_out_mda<< σ_{↑-ord}(L,D) >> :=_{co<σ_{↑-ord}(L,D)>} out_mda<<f>>;

        }
      ⋰
      ll_out_mda<< σ_{↑-ord}(1,2) >> :=_{co<σ_{↑-ord}(1,2)>} out_mda<< σ_{↑-ord}(1,3) >>;

    }
    ll_out_mda<< σ_{↑-ord}(1,1) >> :=_{co<σ_{↑-ord}(1,1)>} out_mda<< σ_{↑-ord}(1,2) >>;

  }

  // 3.2. finalization
  ll_out_mda<<⊥>> := ll_out_mda<< σ_{↑-ord}(1,1) >>
```

Listing 18. Re-Composition Phase

```
// 2. scalar phase
int p_ σ_{f-ord}(1,1) ∈<↔_{f-ass}(1,1)> #PRT( σ_{f-ord}(1,1) )
  ⋱
    int p_ σ_{f-ord}(L,D) ∈<↔_{f-ass}(L,D)> #PRT( σ_{f-ord}(L,D) )
    {
      (
        ll_out_mda<<f>><<
            p_(1,1) ,..., p_(1,D),
                            ...
            p_(L,1) ,..., p_(L,D)>><<b,a>>(
                 =1=>^{MDA}_{⊛ MDA}( I<<1>><p_(1,1),...,p_(L,1)>(0) ) ,
                 ⋮
                 =D=>^{MDA}_{⊛ MDA}( I<<D>><p_(1,D),...,p_(L,D)>(0) ) )
      )_{b∈[1,B^OB]_ℕ, a∈[1,A_b^OB]_ℕ} := f( ( ll_inp_mda<<f>><< p_(1,1) ,..., p_(1,D),
                                                    ...
                                                    p_(L,1) ,..., p_(L,D)>><<b,a>>(
                            =d=>^{MDA}_{# MDA}( I<<1>><p_(1,1),...,p_(L,1)>(0) ) ,
                            ⋮
                            =d=>^{MDA}_{# MDA}( I<<D>><p_(1,D),...,p_(L,D)>(0) ) )
                          )_{b∈[1,B^IB]_ℕ, a∈[1,A_b^IB]_ℕ} )
    }
```

Listing 17. Scalar Phase

<u>Summary:</u> Low-Level Representation

- Expresses **(de/re)-compositions** of data-parallel computations (BLAS, stencil, …) for the **memory** and **core hierarchies** of state-of-the-art parallel architectures (GPU, CPU, …)
- **Straightforwardly transformable** to imperative-style, executable **program code**
- Full **formal** foundation

Definition 12 (Low-Level MDA). Let be $L \in \mathbb{N}$ (representing an ASM's number of layers) and $D \in \mathbb{N}$ (representing an MDH's number of dimensions). Let further be $P = (\,(P_1^1, \ldots, P_D^1)\,, \ldots, (P_1^L, \ldots, P_D^L)\,) \in \mathbb{N}^{L \times D}$ an arbitrary sequence of L-many D-tuples of positive natural numbers, $T \in$ TYPE a scalar type, and $I := (\,(I_d^{<p_1^1,\ldots,p_D^1\,|\,\ldots\,|\,p_1^L,\ldots,p_D^L>} \in \texttt{MDA-IDX-SETs})_{d \in [1,D]_\mathbb{N}}\,)^{<(p_1^1,\ldots,p_D^1) \in P_1^1 \times \ldots \times P_D^1\,|\,\ldots\,|\,(p_1^L,\ldots,p_D^L) \in P_1^L \times \ldots \times P_D^L>}$ an arbitrary collection of D-many MDA index sets (Definition 1) for each particular choice of indices $(p_1^1, \ldots, p_D^1) \in P_1^1 \times \ldots \times P_D^1\ , \ldots ,\ (p_D^L, \ldots, p_D^L) \in P_1^L \times \ldots \times P_D^L$ [14] (illustrated in Figure 17).

An L-layered, D-dimensional, P-partitioned *low-level MDA* that has scalar type T and index sets I is any function $\mathfrak{a}_{ll}$ of type:

$$\mathfrak{a}_{ll}^{<\overbrace{(p_1^1,\ldots,p_D^1) \in P_1^1 \times \ldots \times P_D^1}^{\text{Partitioning: Layer 1}}\,|\,\ldots\,|\,\overbrace{(p_1^L,\ldots,p_D^L) \in P_1^L \times \ldots \times P_D^L}^{\text{Partitioning: Layer } L}>} : I_1^{<p_1^1,\ldots,p_D^1\,|\,\ldots\,|\,p_1^L,\ldots,p_D^L>} \times \ldots \times I_D^{<p_1^1,\ldots,p_D^1\,|\,\ldots\,|\,p_1^L,\ldots,p_D^L>} \to T$$

Definition 11 (Abstract System Model). An *L-Layered Abstract System Model (ASM)*, $L \in \mathbb{N}$, is any pair of two positive natural numbers

$$(\,\texttt{NUM\_MEM\_LYRs}\,,\,\texttt{NUM\_COR\_LYRs}\,) \in \mathbb{N} \times \mathbb{N}$$

for which $\texttt{NUM\_MEM\_LYRs} + \texttt{NUM\_COR\_LYRs} = L$.

…

Definition 13 (Low-Level BUF). Let be $L \in \mathbb{N}$ (representing an ASM's number of layers) and $D \in \mathbb{N}$ (representing an MDH's number of dimensions). Let further $P = (\,(P_1^1, \ldots, P_D^1)\,, \ldots, (P_1^L, \ldots, P_D^L)\,) \in \mathbb{N}^{L \times D}$ be an arbitrary sequence of L-many D-tuples of positive natural numbers, $T \in$ TYPE a scalar type, and $N := (\,(N_d^{<p_1^1,\ldots,p_D^1\,|\,\ldots\,|\,p_1^L,\ldots,p_D^L>} \in \mathbb{N})_{d \in [1,D]_\mathbb{N}}\,)^{<(p_1^1,\ldots,p_D^1) \in P_1^1 \times \ldots \times P_D^1\,|\,\ldots\,|\,(p_1^L,\ldots,p_D^L) \in P_1^L \times \ldots \times P_D^L>}$ be a BUF's size (Definition 5) for each particular choice of $p_1^1, \ldots, p_D^L$.

An L-layered, D-dimensional, P-partitioned *low-level BUF* that has scalar type T and size N is any function $\mathfrak{b}_{ll}$ of type ($\hookrightarrow\!\!\!\!\rightarrow$ denotes bijection):

$$\mathfrak{b}_{ll}^{<\overbrace{\texttt{MEM} \in [1,\texttt{NUM\_MEM\_LYRs}]_\mathbb{N}}^{\text{Memory Region}}\,|\,\overbrace{\sigma:[1,D]_\mathbb{N} \hookrightarrow\!\!\!\!\rightarrow [1,D]_\mathbb{N}}^{\text{Memory Layout}}><\overbrace{(p_1^1,\ldots,p_D^1) \in P_1^1 \times \ldots \times P_D^1}^{\text{Partitioning: Layer 1}}\,|\,\ldots\,|\,\overbrace{(p_1^L,\ldots,p_D^L) \in P_1^L \times \ldots \times P_D^L}^{\text{Partitioning: Layer } L}>} : [0, N_1^{<p_1^1,\ldots,p_D^1\,|\,\ldots\,|\,p_1^L,\ldots,p_D^L>})_{\mathbb{N}_0} \times \ldots \times [0, N_D^{<p_1^1,\ldots,p_D^1\,|\,\ldots\,|\,p_1^L,\ldots,p_D^L>})_{\mathbb{N}_0} \to T$$

Definition 14 (Low-Level Combine Operator). Let be $L \in \mathbb{N}$ (representing an ASM's number of layers) and $D \in \mathbb{N}$ (representing an MDH's number of dimensions). Let further be $\circledast \in \texttt{CO}^{<\Rightarrow^{\texttt{MDA}}_{\texttt{MDA}}\,|\,T\,|\,D\,|\,d>}$ an arbitrary D-dimensional combine operator (Definition 2).

The *low-level representation* $\circledast^{<(l_{\texttt{ASM}}, d_{\texttt{ASM}}) \in \texttt{ASM-LVL}>}$ of operator $\circledast$ is a function that for each pair

$$(l_{\texttt{ASM}}, d_{\texttt{ASM}}) \in \texttt{ASM-LVL} := \{\ (l,d)\ |\ l \in [1,L]_\mathbb{N}\,,\ d \in [1,D]_\mathbb{N}\ \}$$

has the same type and semantics as $\circledast$:

$$\circledast^{<l_{\texttt{ASM}}, d_{\texttt{ASM}}>} \in \texttt{CO}^{<\Rightarrow^{\texttt{MDA}}_{\texttt{MDA}}\,|\,T\,|\,D\,|\,d>}\quad,\quad \circledast^{<l_{\texttt{ASM}}, d_{\texttt{ASM}}>}(a,b) := \circledast(a,b)$$

i.e., $\circledast^{<l_{\texttt{ASM}}, d_{\texttt{ASM}}>}$ works exactly as combine operator $\circledast$, but its type is enriched with a meta-parameter that captures the notation of an ASM layer $l_{\texttt{ASM}} \in [1,L]_\mathbb{N}$ and dimension $d_{\texttt{ASM}} \in [1,D]_\mathbb{N}$.

<u>Lowering:</u> *High Level → Low-Level*

Our *Lowering Process* is formally proved, and it is generic in performance-critical parameters:

No.	Name	Range	Description
0	#PRT	MDH-LVL → $\mathbb{N}$	number of parts
D1	$\sigma_{\downarrow\text{-ord}}$	MDH-LVL ↪↠ MDH-LVL	de-composition order
D2	$\leftrightarrow_{\downarrow\text{-ass}}$	MDH-LVL ↪↠ ASM-LVL	ASM assignment (de-composition)
D3	$\downarrow\text{-mem}^{<ib>}$	MDH-LVL → MR	memory regions of input BUFs (ib)
D4	$\sigma^{<ib>}_{\downarrow\text{-mem}}$	MDH-LVL → $[1,\ldots,D^{\text{IB}}_{\text{ib}}]_{\mathcal{S}}$	memory layouts of input BUFs (ib)
S1	$\sigma_{f\text{-ord}}$	MDH-LVL ↪↠ MDH-LVL	scalar function order
S2	$\leftrightarrow_{f\text{-ass}}$	MDH-LVL ↪↠ ASM-LVL	ASM assignment (scalar function)
S3	$f^{\downarrow}\text{-mem}^{<ib>}$	MR	memory region of input BUF (ib)
S4	$\sigma^{<ib>}_{f^{\downarrow}\text{-mem}}$	$[1,\ldots,D^{\text{IB}}_{\text{ib}}]_{\mathcal{S}}$	memory layout of input BUF (ib)
S5	$f^{\uparrow}\text{-mem}^{<ob>}$	MR	memory region of output BUF (ob)
S6	$\sigma^{<ob>}_{f^{\uparrow}\text{-mem}}$	$[1,\ldots,D^{\text{OB}}_{\text{ob}}]_{\mathcal{S}}$	memory layout of output BUF (ob)
R1	$\sigma_{\uparrow\text{-ord}}$	MDH-LVL ↪↠ MDH-LVL	re-composition order
R2	$\leftrightarrow_{\uparrow\text{-ass}}$	MDH-LVL ↪↠ ASM-LVL	ASM assignment (re-composition)
R3	$\uparrow\text{-mem}^{<ob>}$	MDH-LVL → MR	memory regions of output BUFs (ob)
R4	$\sigma^{<ob>}_{\uparrow\text{-mem}}$	MDH-LVL → $[1,\ldots,D^{\text{OB}}_{\text{ob}}]_{\mathcal{S}}$	memory layouts of output BUFs (ob)

Table 1. Tuning parameters of our low-level expressions

tile sizes

exploiting <u>core hierarchy</u> (parallelization)

exploiting <u>memory hierarchy</u> (data movements)

…

…

…

…

*We use our <u>Auto-Tuning Framework (ATF) [1]</u> to **fully automatically** determine optimized values of parameters*

[1] Rasch, Schulze, Steuwer, Gorlatch, "Efficient Auto-Tuning of Parallel Programs with Interdependent Tuning Parameters via Auto-Tuning Framework (ATF)", TACO'21

Experimental Results

We experimentally evaluate our MDH approach in terms of ***Performance*** & ***Portability*** & ***Productivity***:

Competitors:

1. Scheduling Approach:
   - Apache TVM [2] (GPU & CPU)
2. Polyhedral Compilers:
   - PPCG [3] (GPU)
   - Pluto [4] (CPU)
3. Functional Approach:
   - Lift [5] (GPU & CPU)
4. Domain-Specific Libraries:
   - NVIDIA cuBLAS & cuDNN (GPU)
   - Intel oneMKL & oneDNN (CPU)

Case Studies:

1. Linear Algebra Routines:
   - Matrix Multiplication (MatMul)
   - Matrix-Vector Multiplication (MatVec)
2. Stencil Computations:
   - Jacobi Computation (Jacobi1D)
   - Gaussian Convolution (Conv2D)
3. Quantum Chemistry:
   - Coupled Cluster (CCSD(T))
4. Data Mining:
   - Probabilistic Record Linkage (PRL)
5. Deep Learning:
   - Multi-Channel Convolution (MCC)
   - Capsule-Style Convolution (MCC_Capsule)

[2] Chen et al., "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning", OSDI'18

[3] Verdoolaege et al., "Polyhedral Parallel Code Generation for CUDA", TACO'13

[4] Bondhugula et al., "PLuTo: A Practical and Fully Automatic Polyhedral Program Optimization System", PLDI'08

[5] Steuwer et al., "Generating Performance Portable Code using Rewrite Rules", ICFP'15

Experimental Results

Highlights only

<u>Performance Evaluation:</u> (via runtime comparison)

Deep Learning	NVIDIA Ampere GPU									
	ResNet-50				VGG-16				MobileNet	
	Training		Inference		Training		Inference		Training	Inference
	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MCC
TVM+Ansor	1.00	1.26	1.05	2.22	0.93	1.42	0.88	1.14	0.94	1.00
PPCG	3456.16	8.26	-	7.89	1661.14	7.06	5.77	5.08	2254.67	7.55
PPCG+ATF	3.28	2.58	13.76	5.44	4.26	3.92	9.46	3.73	3.31	10.71
cuDNN	0.92	-	1.85	-	1.22	-	1.94	-	1.81	2.14
cuBLAS	-	1.58	-	2.67	-	0.93	-	1.04	-	-
cuBLASEx	-	1.47	-	2.56	-	0.92	-	1.02	-	-
cuBLASLt	-	1.26	-	1.22	-	0.91	-	1.01	-	-

NVIDIA

<u>MDH speedup over</u>

- TVM: 0.88x – 2.22x
- PPCG: 2.58x – 13.76x
- *(cuBLAS/cuDNN: 0.91x – 2.67x)*

Deep Learning	Intel Skylake CPU									
	ResNet-50				VGG-16				MobileNet	
	Training		Inference		Training		Inference		Training	Inference
	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MCC
TVM+Ansor	1.53	1.05	1.14	1.20	1.97	1.14	2.38	1.27	3.01	1.40
Pluto	355.81	49.57	364.43	13.93	130.80	93.21	186.25	36.30	152.14	75.37
Pluto+ATF	13.08	19.70	170.69	6.57	3.11	6.29	53.61	8.29	3.50	25.41
oneDNN	0.39	-	5.07	-	1.22	-	9.01	-	1.05	4.20
oneMKL	-	0.44	-	1.09	-	0.88	-	0.53	-	-
oneMKL(JIT)	-	6.43	-	8.33	-	27.09	-	9.78	-	-

intel

<u>MDH speedup over</u>

- TVM: 1.05 – 3.01x
- Pluto: 6.29x – 364.43x
- *(oneMKL/oneDNN: 0.39x – 9.01x)*

Case Study "Deep Learning" for which most competitors are highly optimized (<u>most challenging for us!</u>)

Significantly higher speedups for other case studies,
e.g., >170x over TVM on GPU already for straightforward dot products

Experimental Results

Highlights only

<u>Portability Evaluation:</u> (via Pennycook Metric [6])

Deep Learning	Pennycook Metric									
	ResNet-50				VGG-16				MobileNet	
	Training		Inference		Training		Inference		Training	Inference
	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MCC
MDH+ATF	0.67	0.76	0.91	1.00	0.98	0.95	0.97	0.68	0.98	1.00
TVM+Ansor	0.53	0.62	0.89	0.59	0.76	0.81	0.70	0.61	0.54	0.75

Our other competitors achieve lowest portability — of “0.00” — only,
because they are limited to particular architectures and/or application classes

[6] Pennycook et al., “Implications of a Metric for Performance Portability”, FGCS’19

Experimental Results

Highlights only

Productivity Evaluation: (via intuitive argumentation)

```
cublasSgemv( /* ... */ );
```

Listing 4. cuBLAS program expressing Matrix-Vector Multiplication (MatVec)

```
for( int i = 0 ; i < M ; ++i )
  for( int k = 0 ; k < K ; ++k )
      w[i] += M[i][k] * v[k];
```

Listing 2. PPCG/Pluto program expressing Matrix-Vector Multiplication (MatVec)

```
def MatVec(I, K):
    M = te.placeholder((I, K), name='M', dtype='float32')
    v = te.placeholder((K,), name='v', dtype='float32')

    k = te.reduce_axis((0, K), name='k')
    w = te.compute(
        (I,),
        lambda i: te.sum(M[i, k] * v[k], axis=k)
    )
    return [M, v, w]
```

Listing 1. TVM program expressing Matrix-Vector Multiplication (MatVec)

```
nFun(n => nFun(m =>
  fun(matrix: [[float]n]m => fun(xs: [float]n =>
    matrix :>> map(fun(row =>
      zip(xs, row) :>> map(*) :>> reduce(+, 0)
    )) )) ))
```

Listing 3. Lift program expressing Matrix-Vector Multiplication (MatVec)

Conclusion

- Our approach **combines** together three major goals as compared to related work: ***Performance*** & ***Portability*** & ***Productivity***

- For this, we **formally introduce program representations** on both:

  - **<u>high level</u>**, for conveniently expressing, in one uniform formalism, the various kinds of data-parallel computations, agnostic from hardware and optimization details, while still capturing all information relevant for generating high-performance program code;

  - **<u>low level</u>**, which allows uniformly reasoning, in the same formalism, about optimized (de/re)-compositions of data-parallel computations targeting different kinds of architectures (GPUs, CPUs, etc).

- We **lower** our high-level representation to our low-level representation, in a **formally sound** manner, by introducing a generic search space that is based on **performance-critical parameters** and **auto-tuning**

- Our **experiments** confirm that our MDH approach often achieves higher ***Performance*** & ***Portability*** & ***Productivity*** than popular state-of-practice approaches, including hand-optimized libraries provided by vendors

Future Work

- Expressing and optimizing simultaneously **multiple data-parallel computations**, rather than optimizing computations individually and thus independently from each other only (a.k.a. *fusing optimization*).
- Supporting computations on **sparse data formats**
- Introduce an **analytical cost model** to accelerate (or even avoid) the auto-tuning overhead (possibly guided by machine learning techniques)
- Implement our approach into **MLIR** to make our work better accessible for the community
- Targeting **assembly languages** to benefit from assembly-level optimizations
- Extending our approach toward distributed multi-device systems that may be **heterogeneous**
- …

This work provides the (formal) foundation for all our future goals!

WESTFÄLISCHE
WILHELMS-UNIVERSITÄT
MÜNSTER

Questions?

Grateful for any feedback

Ari Rasch
a.rasch@wwu.de

Richard Schulze
r.schulze@wwu.d