Universität Münster

C4ML 2026

7th Workshop on Compilers for Machine Learning

Systematic Code Generation for ML Computations based on Multi-Dimensional Homomorphisms

Ari Rasch, Richard Schulze
University of Münster, Germany

Goal of this Talk

Observation:

State-of-the-art **ML compilers/libraries**
struggle with **fully automatically** achieving **high performance**
for an **extensible set of target architectures**

Examples:

NVIDIA CUTLASS

tvm

PyTorch Inductor

OpenXLA

oneAPI oneMKL/oneDNN

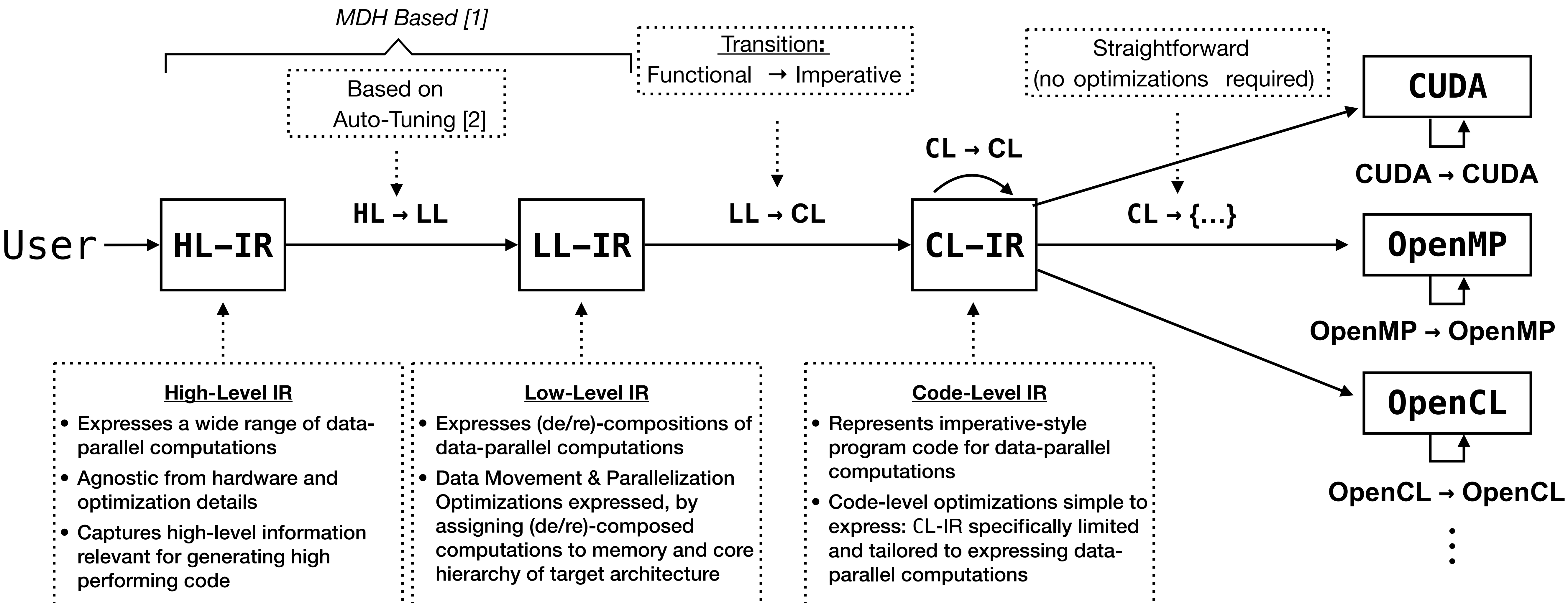
IREE

NVIDIA cuBLAS/cuDNN

We present a systematic **ML code generation process** that **fully automatically** achieves **high performance** across an **extensible set of architectures**

Overview

End-to-end *ML Code-Generation* approach:



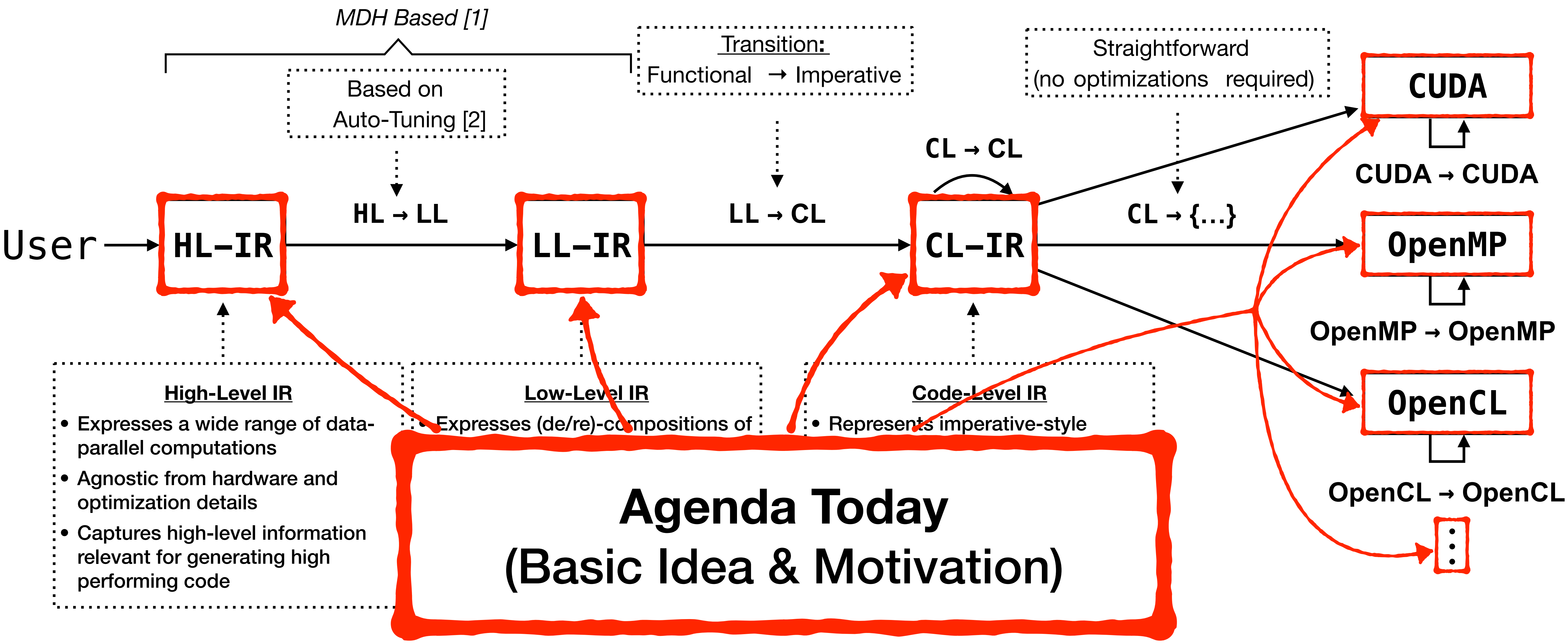
Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

Overview

End-to-end *ML Code-Generation* approach:



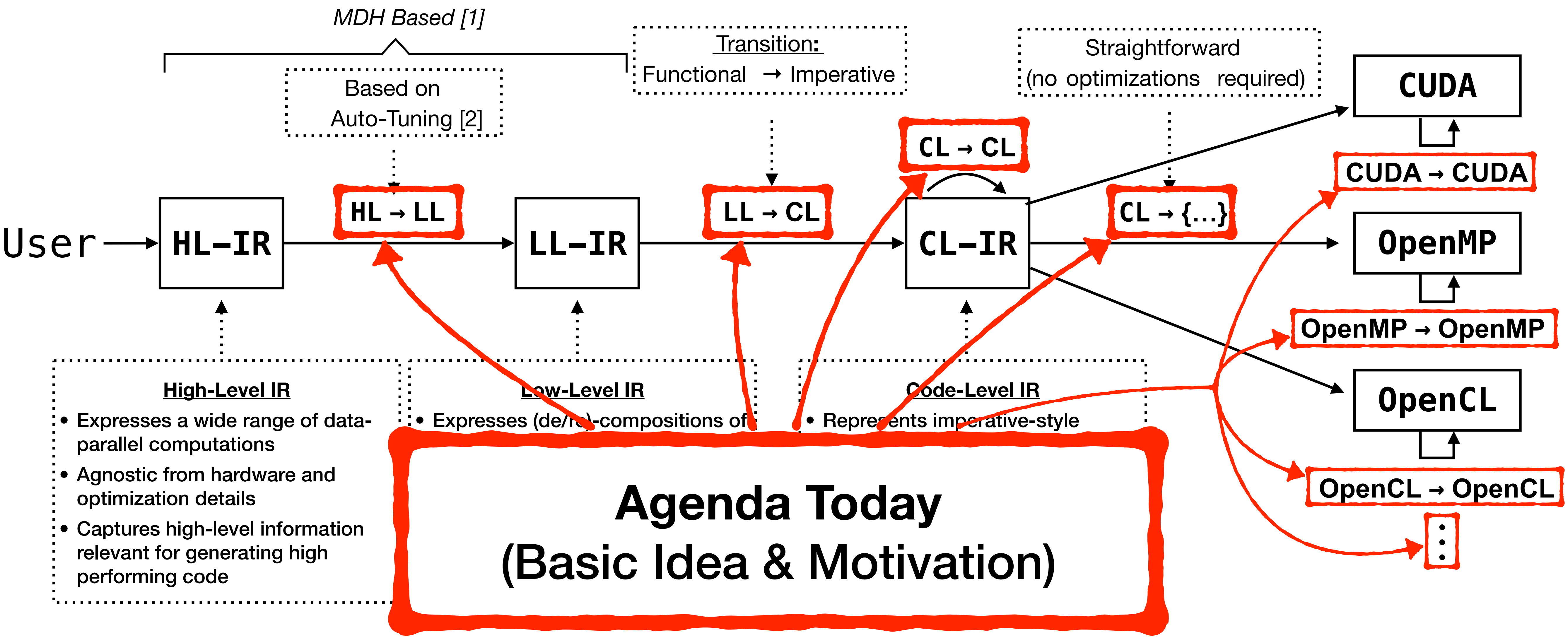
Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

Overview

End-to-end *ML Code-Generation* approach:



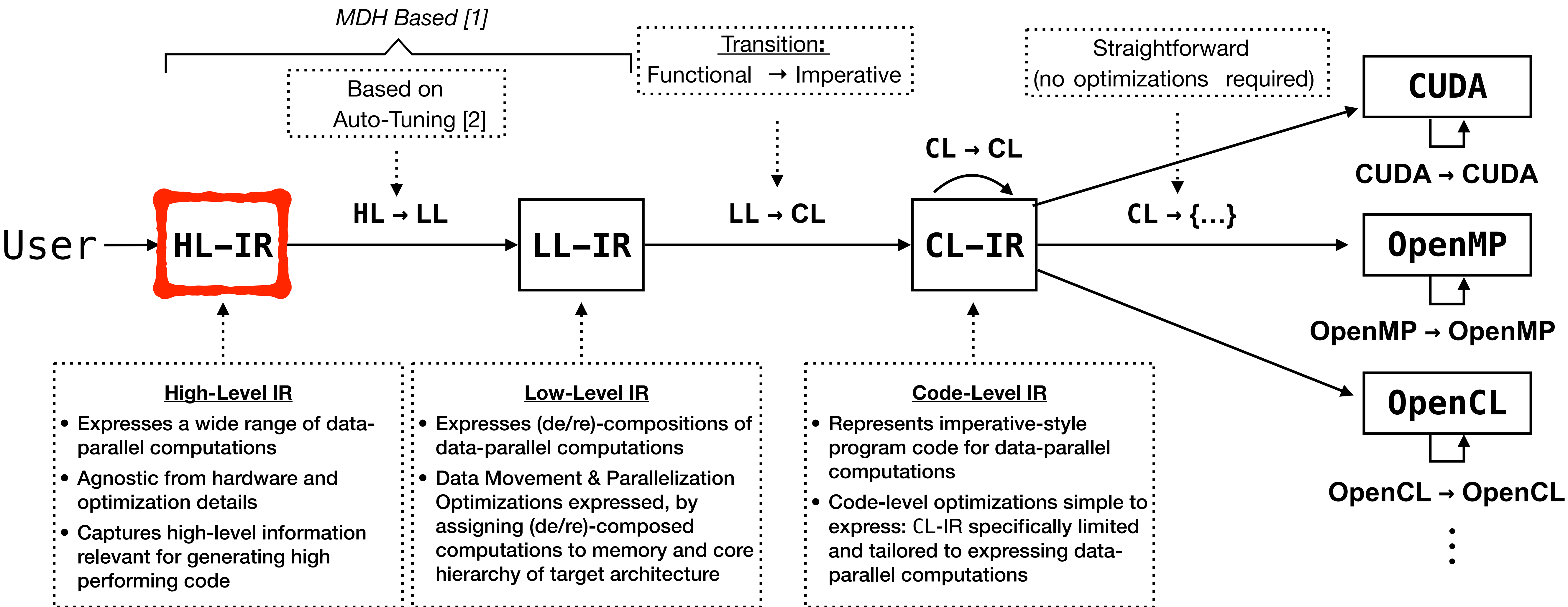
Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

Overview

<u>End-to-end *ML Code-Generation* approach:</u>



Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

HL-IR: High-Level IR

Example: `MatVec` expressed in HL-IR (using its Python DSL representation)

```
def matvec( T: BasicType, I: int, K: int ):
    @mdh()
    def mdh_matvec():
        return (
            out_view[T]( w = [lambda i,k: (i)] ),
              md_hom[I,K]( mul, ( cc, pw(scalar_plus) ) ),
                inp_view[T,T]( M = [lambda i,k: (i,k)] ,
                               v = [lambda i,k: (k)   ] ) )
```

High–Level Representation of MatVec

What is happening here:

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

The HL-IR **uniformly expresses** ML computations, **abstracting away low-level details** while **preserving high-level algebraic information**

HL-IR: ML Examples

md_hom	f	$\circledast_1,\dots,\circledast_D$
MatMul<F,F>	$*$	$\mathbin{+\!\!+},\mathbin{+\!\!+},+$
MatMul<F,T>	$*$	$\mathbin{+\!\!+},\mathbin{+\!\!+},+$
MatMul<T,F>	$*$	$\mathbin{+\!\!+},\mathbin{+\!\!+},+$
MatMul<T,T>	$*$	$\mathbin{+\!\!+},\mathbin{+\!\!+},+$
BatchMatMul<F,F>	$*$	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+},+$
$\vdots$	$\vdots$	$\vdots$
BiasAddGrad<NHWC>	id	$+,+,+,\mathbin{+\!\!+}$
BiasAddGrad<NCHW>	id	$+,\mathbin{+\!\!+},+,+$
CheckNumerics	$(x)\mapsto(x == \text{NaN})$	$\vee,\dots,\vee$
Sum<0><F>	id	$+,\mathbin{+\!\!+},\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
Sum<0><T>	id	$+,\mathbin{+\!\!+},\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
Sum<1><F>	id	$\mathbin{+\!\!+},+,\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
Sum<0,1><F>	id	$+,+,\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
$\vdots$	$\vdots$	$\vdots$
Prod<0><F>	id	$*,\mathbin{+\!\!+},\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
$\vdots$	$\vdots$	$\vdots$
All<0><F>	id	$\&\&,\mathbin{+\!\!+},\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
$\vdots$	$\vdots$	$\vdots$

Linear Algebra, Contractions, ...
(Computation Specification)

Views	inp_view		out_view
	I_1	I_2	O
MatMul<F,F>	$(i,j,k)\mapsto(i,k)$	$(i,j,k)\mapsto(k,j)$	$(i,j,k)\mapsto(i,j)$
MatMul<F,T>	$(i,j,k)\mapsto(i,k)$	$(i,j,k)\mapsto(j,k)$	$(i,j,k)\mapsto(i,j)$
MatMul<T,F>	$(i,j,k)\mapsto(k,i)$	$(i,j,k)\mapsto(k,j)$	$(i,j,k)\mapsto(i,j)$
MatMul<T,T>	$(i,j,k)\mapsto(k,i)$	$(i,j,k)\mapsto(j,k)$	$(i,j,k)\mapsto(i,j)$
BatchMatMul<F,F>	$(b_1,\dots,i,j,k)\mapsto(b_1,\dots,i,k)$	$(b_1,\dots,i,j,k)\mapsto(b_1,\dots,k,j)$	$(b_1,\dots,i,j,k)\mapsto(b_1,\dots,i,j)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
BiasAddGrad<NHWC>	$(n,h,w,c)\mapsto(n,h,w,c)$	/	$(n,h,w,c)\mapsto(n,h,w)$
BiasAddGrad<NCHW>	$(n,c,h,w)\mapsto(n,c,h,w)$	/	$(n,c,h,w)\mapsto(n,h,w)$
CheckNumerics	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto()$
Sum<0><F>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_2,\dots,i_D)$
Sum<0><T>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(0,i_2,\dots,i_D)$
Sum<1><F>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_1,i_3,\dots,i_D)$
Sum<0,1><F>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_3,\dots,i_D)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Prod<0><F>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_2,\dots,i_D)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
All<0><F>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_2,\dots,i_D)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Linear Algebra, Contractions, ... *(Data Specification)*

md_hom	f	$\circledast_1,\dots,\circledast_D$
Fill	id	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
ExpandDims<0>	id	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
ExpandDims<1>	id	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
ExpandDims<0,1>	id	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
$\vdots$	$\vdots$	$\vdots$
Transpose<σ>	id	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
Exp	exp	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
Mul	$*$	$\mathbin{+\!\!+},\dots,\mathbin{+\!\!+}$
BiasAdd<NHWC>	$+$	$\mathbin{+\!\!+},\mathbin{+\!\!+},\mathbin{+\!\!+},\mathbin{+\!\!+}$
BiasAdd<NCHW>	$+$	$\mathbin{+\!\!+},\mathbin{+\!\!+},\mathbin{+\!\!+},\mathbin{+\!\!+}$
Range	$(s,d,i)\mapsto(s+d*i)$	$\mathbin{+\!\!+}$

Point-Wise, Re-Shaping, ...
(Computation Specification)

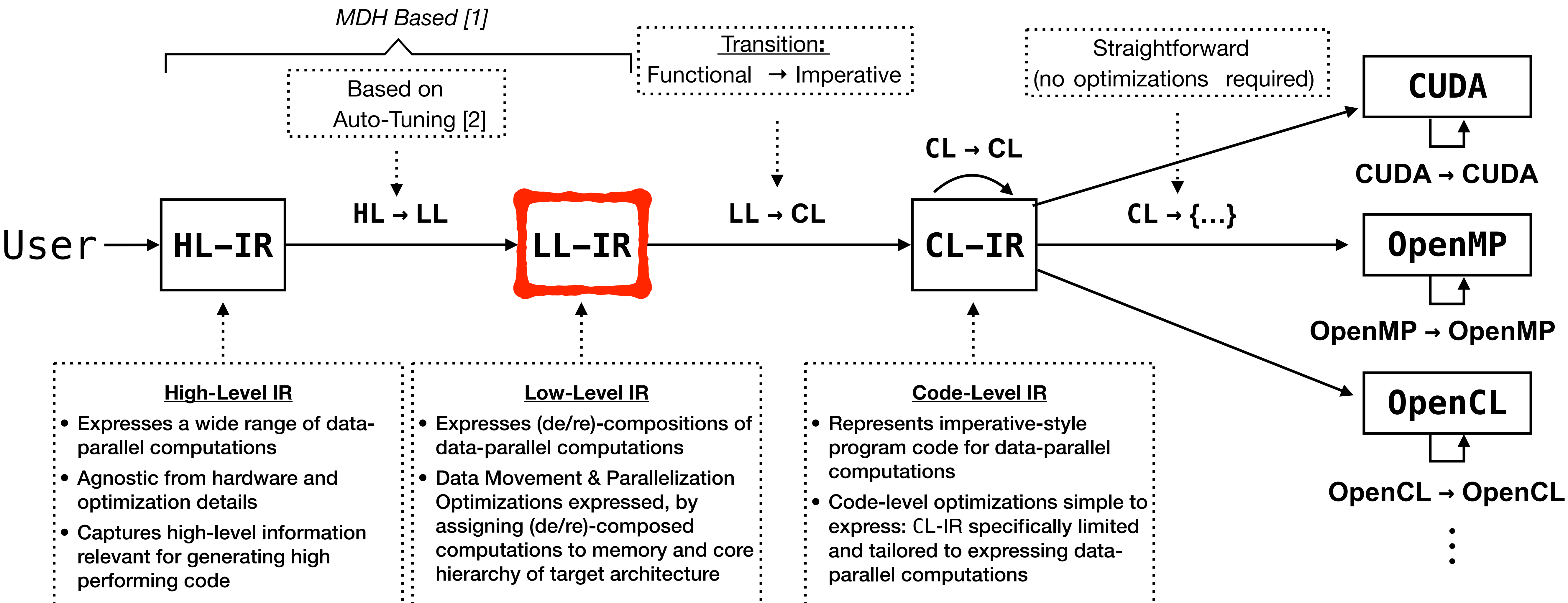
Views	inp_view		out_view
	I_1	I_2	O
Fill	$(i_1,\dots,i_D)\mapsto()$	/	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$
ExpandDims<0>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(0,i_1,i_2,\dots,i_D)$
ExpandDims<0>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_1,0,i_2,\dots,i_D)$
ExpandDims<0,1>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(0,0,i_1,\dots,i_D)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Transpose<σ>	$(i_1,\dots,i_D)\mapsto(\sigma(i_1),\dots,\sigma(i_D))$	/	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$
Exp	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$
Mul	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$
	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_{k-1},i_{k+1},\dots,i_D)$	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$
	$\vdots$	$\vdots$	$\vdots$
BiasAdd<NHWC>	$(n,h,w,c)\mapsto(n,h,w,c)$	$(n,h,w,c)\mapsto(c)$	$(n,h,w,c)\mapsto(n,h,w,c)$
BiasAdd<NCHW>	$(n,c,h,w)\mapsto(n,c,h,w)$	$(n,c,h,w)\mapsto(c)$	$(n,c,h,w)\mapsto(n,c,h,w)$
Range	$(i)\mapsto()$	$(i)\mapsto()$	$(i)\mapsto(i)$

Point-Wise, Re-Shaping, ... *(Data Specification)*

Important **ML kernels** can be expressed **uniformly** in HL-IR

Overview

<u>End-to-end *ML Code-Generation* approach:</u>



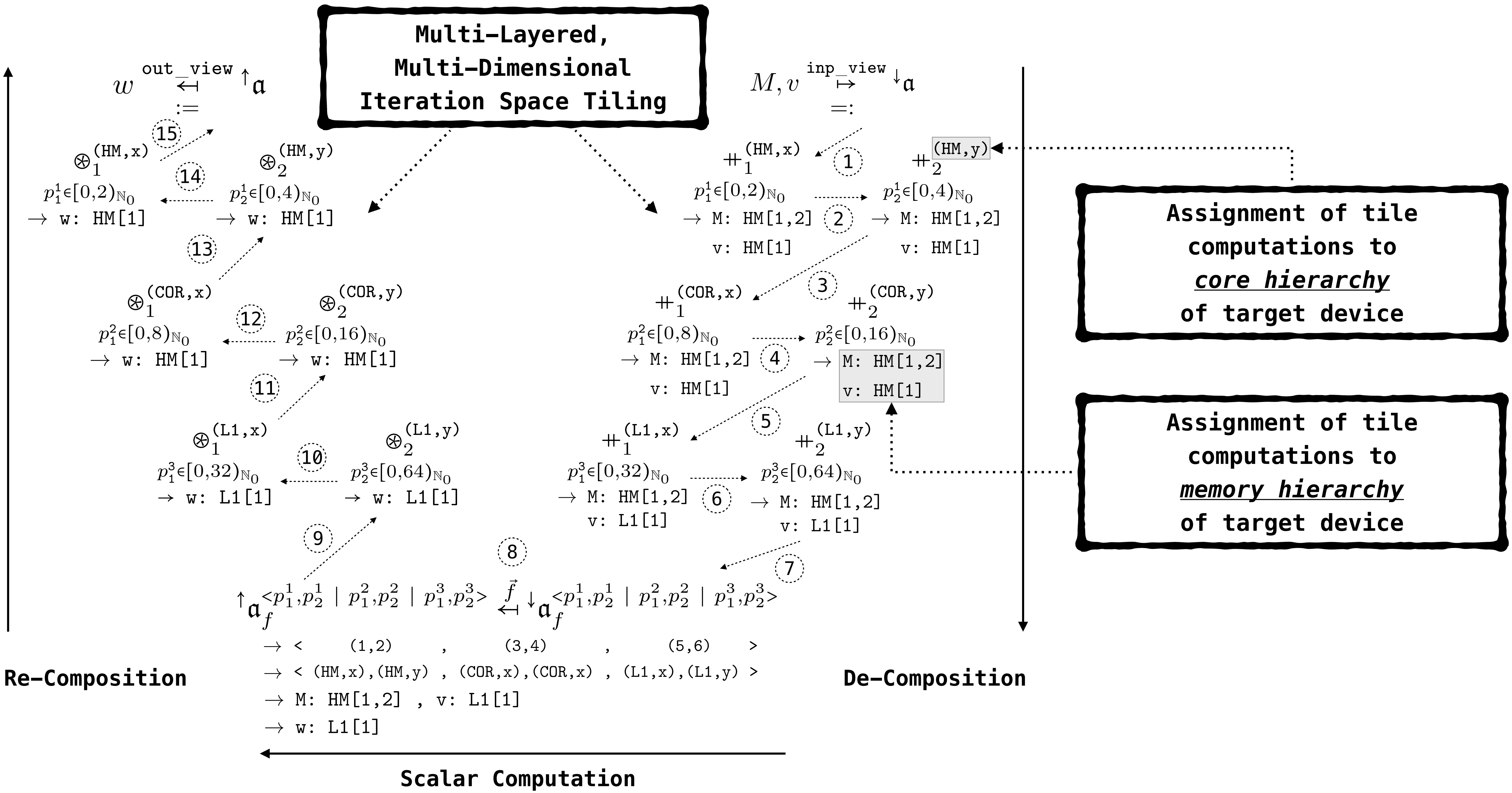
Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

LL-IR: Low-Level IR

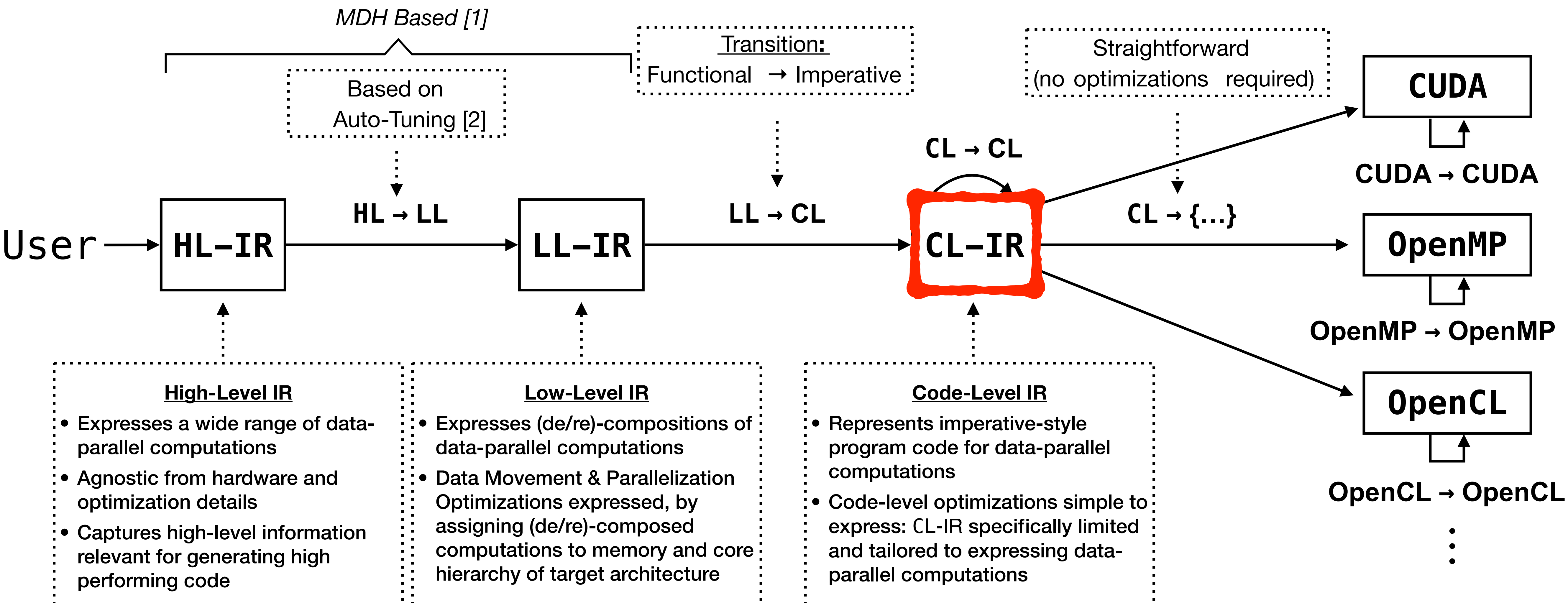
Goal: Expressing optimized *de-composition* and *re-composition* of ML computations



Expresses *parallelization* and *data-movement* optimizations

Overview

End-to-end *ML Code-Generation* approach:



Built around **three core IRs** with **automated transformations** between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

CL-IR: Code-Level IR

General Structure — (deliberately) limited to expressing ML computations:

```
⋮ // ... (further buffers)
// input buffers
IB_1__l_d[...,p_l_d,...][N__IB_1__1,N__IB_1__2,...]
                 in MEM_IB_l_d_1[ LYT_IB_l_d_1 ]

IB_2__l_d[...,p_l_d,...][N__IB_2__1,N__IB_2__2,...]
                 in MEM_IB_l_d_2[ LYT_IB_l_d_2 ]
⋮
// output buffers
OB_1__l_d[...,p_l_d,...][N__OB_1__1,N__OB_1__2,...]
                 in MEM_OB_l_d_1[ LYT_OB_l_d_1 ]

OB_2__l_d[...,p_l_d,...][N__OB_2__1,N__OB_2__2,...]
                 in MEM_OB_l_d_2[ LYT_OB_l_d_2 ]
⋮
⋮ // ... (further buffers)

⋱ // ... (outer computations)

 // iterations
 [for | par_for<ASS_l_d>]( p_l_d < P_l_d )
 {
   // de-composition
   de_comp_op<l,d>: IB_1__l_d,... ⟶ IB_1__l'_d',...
     ⋱
       // ... (inner computations)
     ⋰
   // re-composition
   re_comp_op<l,d>: OB_1__l_d,... ⟵ OB_1__l'_d',...
 } // p_l_d

⋰ // ... (outer computations)
```

De-Composition Buffers

Re-Composition Buffers

Loop Nest

De-Composition Operator

Scalar Computation

Re-Composition Operator

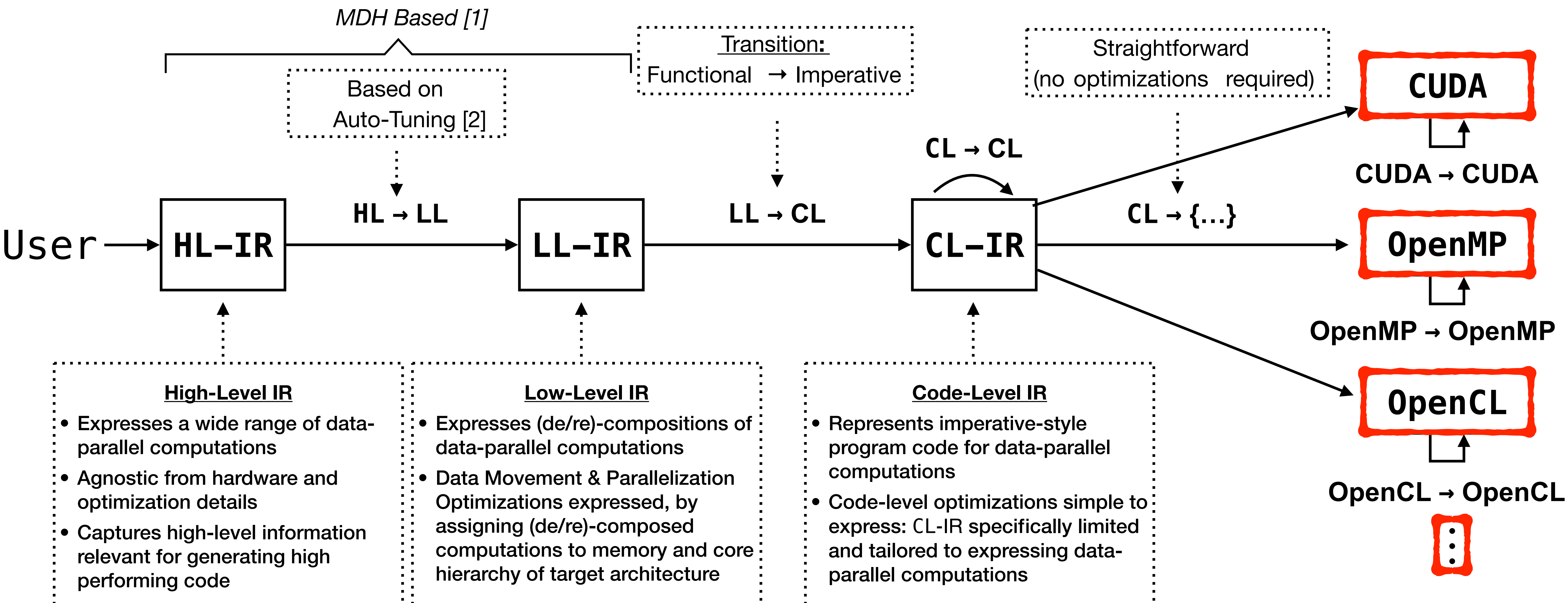
Each CL-IR always consists of the same (simple) structure:

1. De-Composition Buffers
2. Re-Composition Buffers
3. **Loop Nest (canonical):**
   - → **De-Composition OPs**
   - → **Re-Composition OPs**
4. Scalar Computation

Expresses **ML computations** as **minimalistic, structured, imperative-style** code

Overview

<u>End-to-end *ML Code-Generation* approach:</u>



Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

{...}-IR: Backend IRs

We define *minimalistic* IRs for target models, limited to required features:

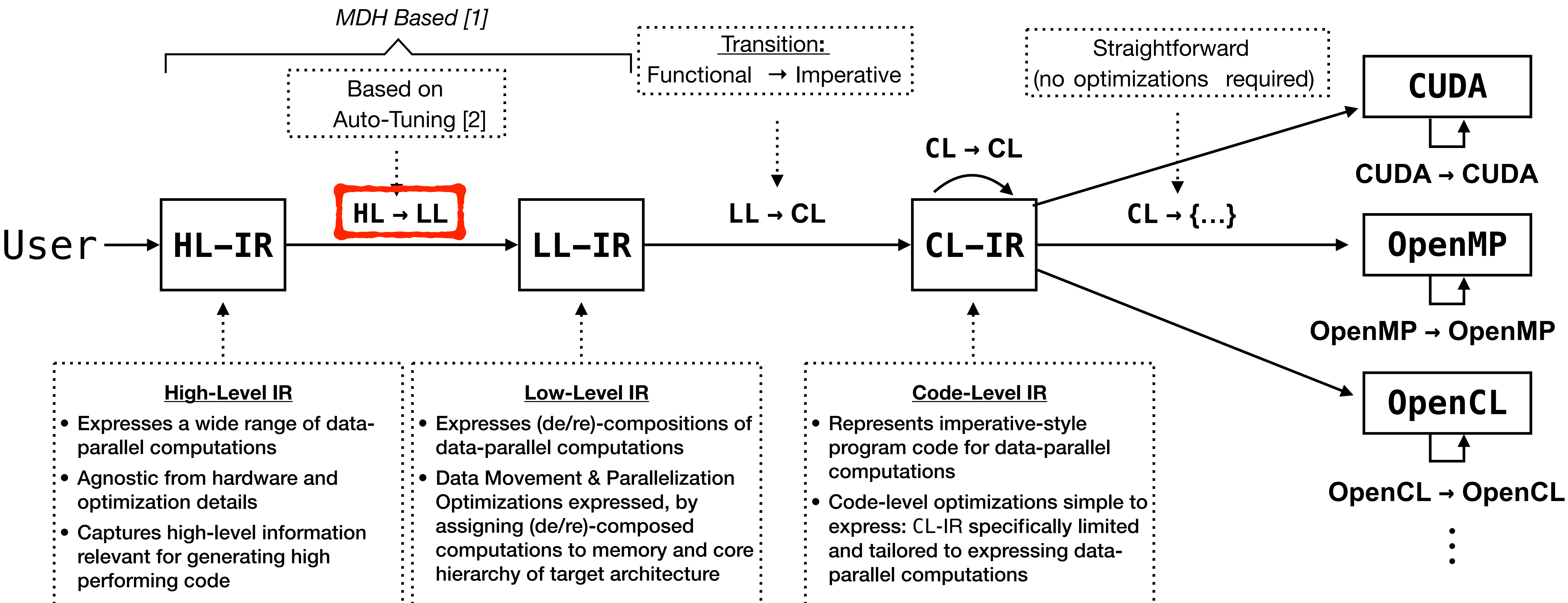
NVIDIA CUDA | OpenMP | OpenCL™ — **Currently supported**

AMD ROCm | Triton | MLIR | ... — **Future Targets**

Extend CL-IR by **target-specific features** (e.g., tensor core operations)

Overview

End-to-end *ML Code-Generation* approach:



Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

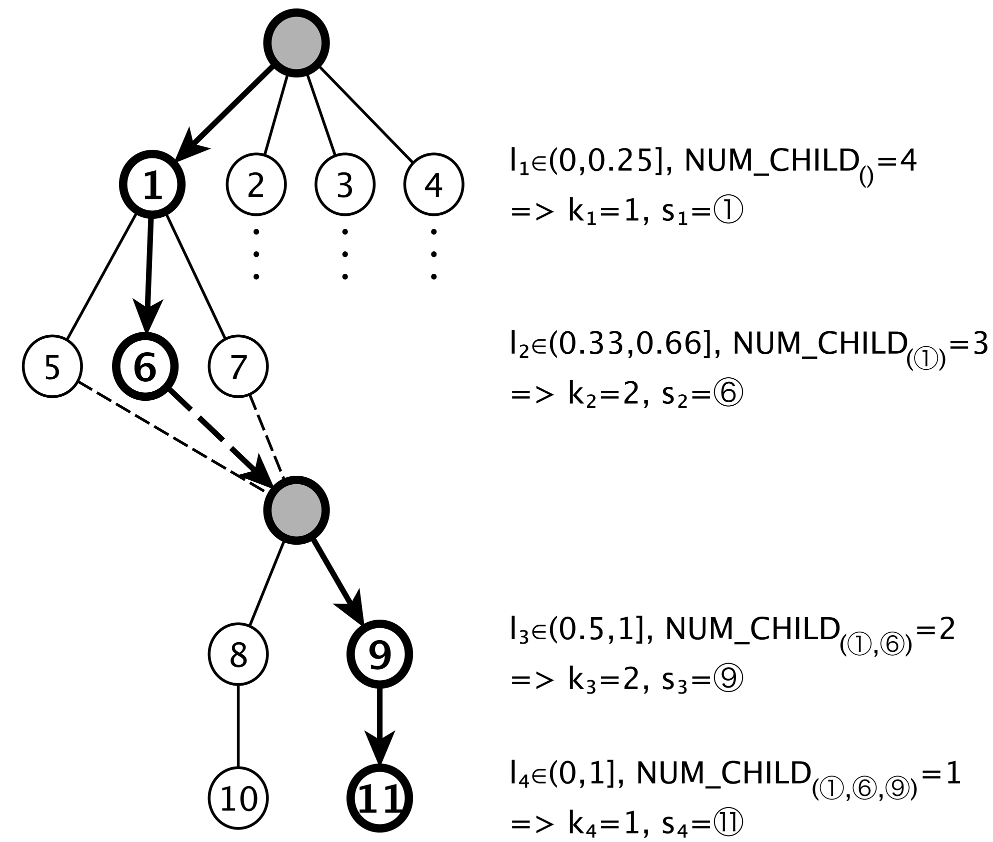
[2] Rasch, Schulze, Steuwer, Gorlatch, "*Efficient Auto-Tuning of Parallel Programs with Interdependent Tuning Parameters via Auto-Tuning Framework (ATF)*", TACO'21

<u>Transformation:</u> HL-IR → LL-IR

<u>Based on *MDH [1]* and *ATF [2]*:</u>

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

MDH-Based Tuning Parameters [1]



ATF-Based Search Space Exploration [2]

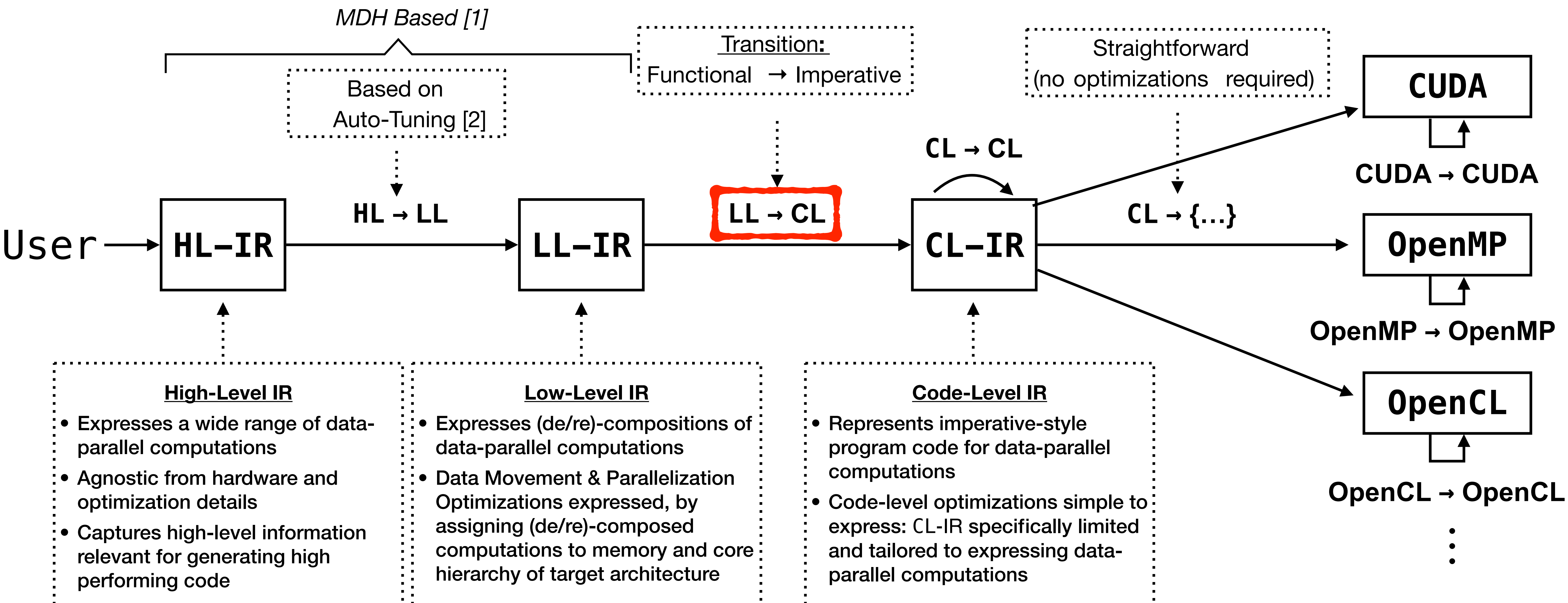
Formally defined transformation driven by **performance-critical parameters** [1] and **auto-tuning** [2]

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

Overview

<u>End-to-end *ML Code-Generation* approach:</u>



Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

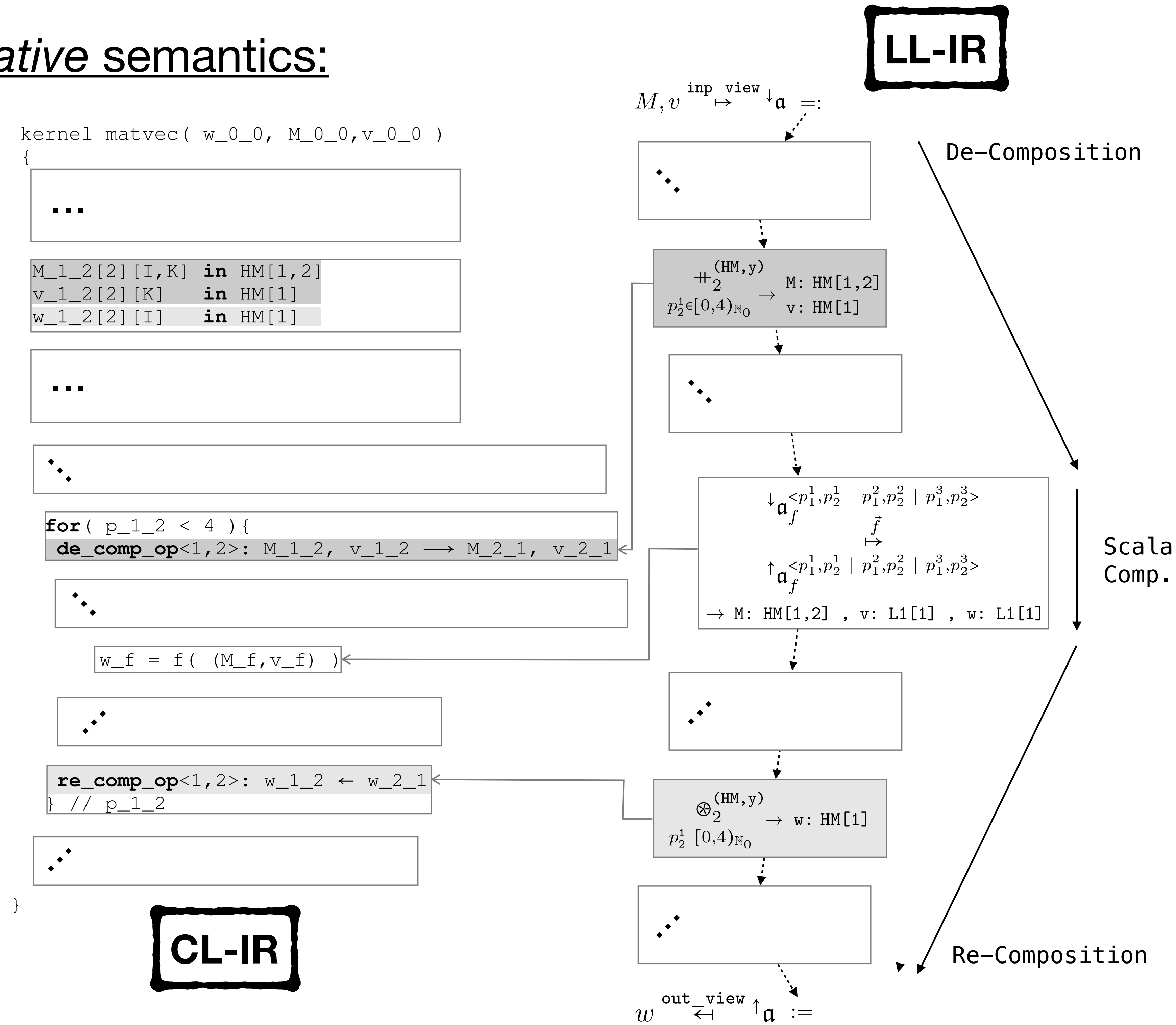
Transformation: LL-IR → CL-IR

Shift from *functional* to *imperative* semantics:

De-Compositions:
- upper part of loops
- copy tiles of input data

Scalar Comp.:
- middle part of loops
- compute scalar function

Re-Compositions:
- lower part of loops
- combine computed intermediate results

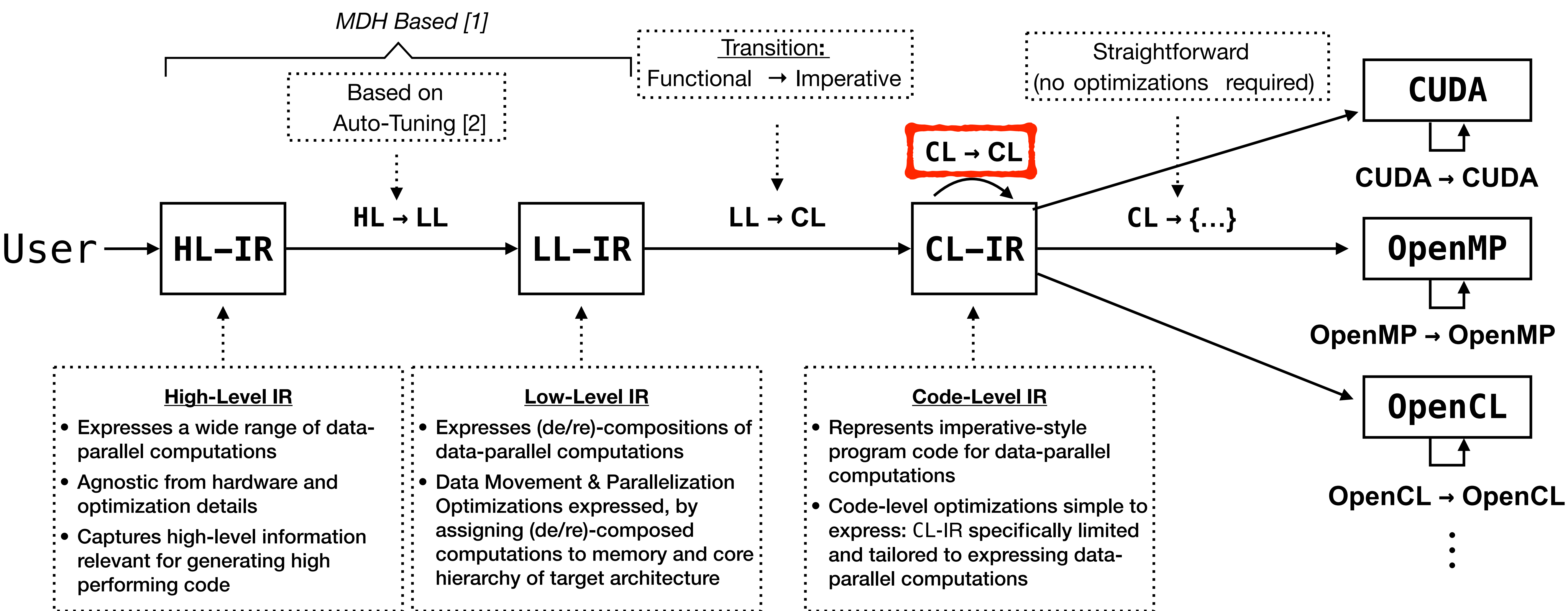


CL-IR instance generated as **sound by construction**

(numerous intermediate buffers, over-approximated buffer sizes, …)

Overview

<u>End-to-end *ML Code-Generation* approach:</u>



Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

Transformation: CL-IR → CL-IR

Code-Level Optimizations:

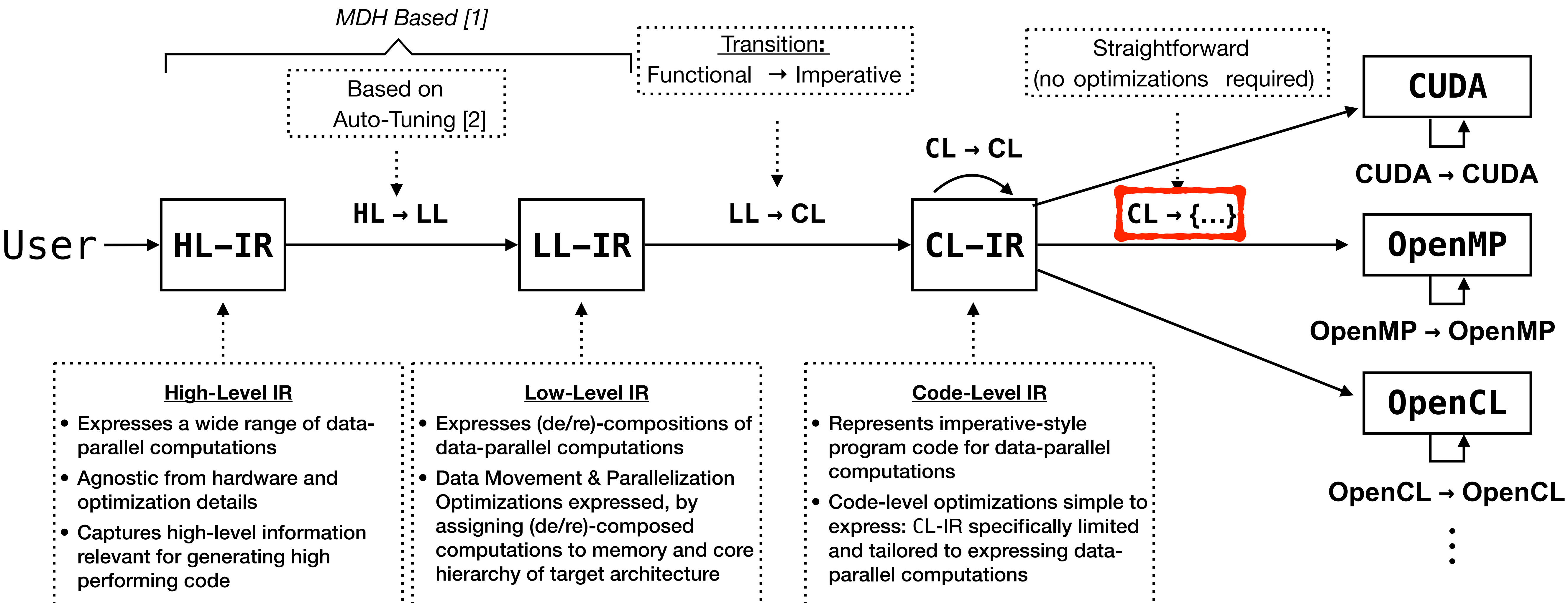
Deterministic

Eliminate Functional Overhead	*Standard Code-Level Optimizations*
`COElimination`	`Algebraic Index Simplifications`
`Partition Index Elimination`	`Loop Unrolling`
`Buffer Size Reduction ...`	`Function Inlining ...`

Minimalistic design of CL-IR
simplifies expressing code-level optimizations

Overview

End-to-end *ML Code-Generation* approach:



Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

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

<u>Transformation:</u> CL-IR → {…}

<u>Straightforward – all major optimization decisions expressed in earlier steps:</u>

```
// …
M_1_1[2][I,K] in SM[1,2]
v_1_1[2][K]   in RM[1]
w_1_1[2][I]   in RM[1]
// …

for(p_1_2 < 4){
  par_for<3,1>(p_2_1 < 8){
    // …
  }
}
```

CL-IR

→

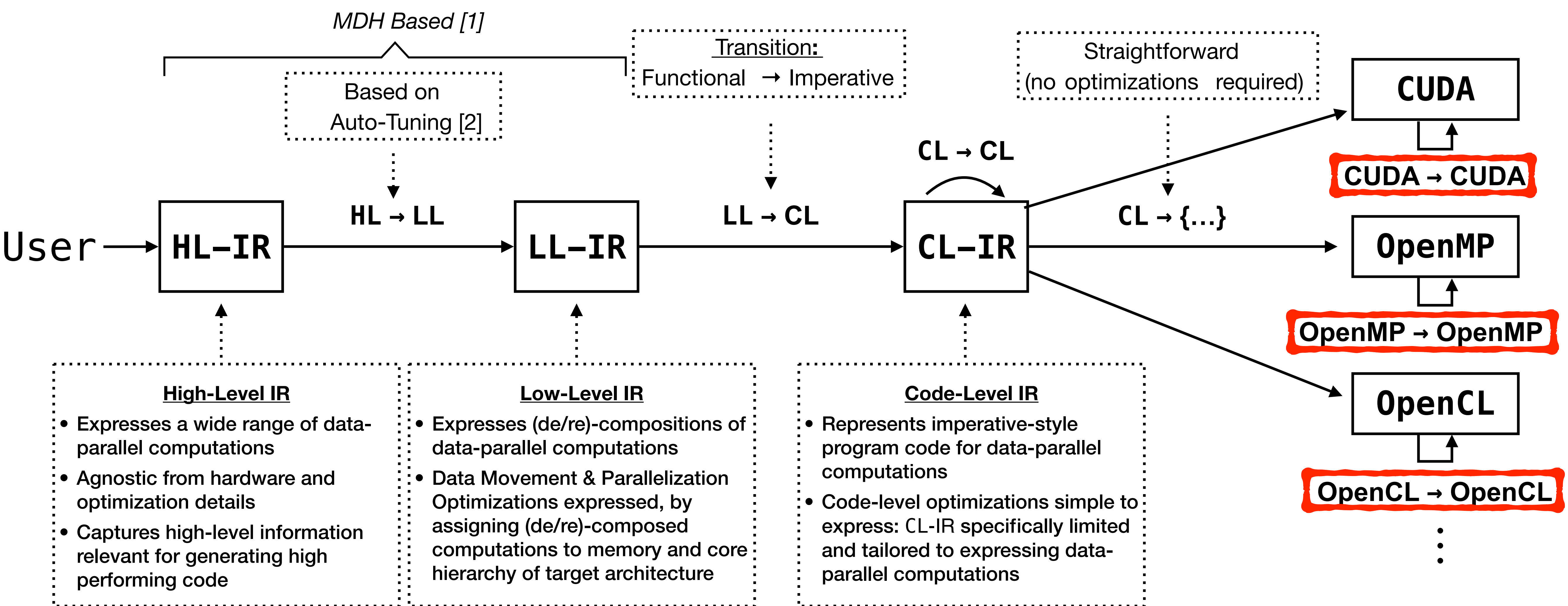
```
#define T_INP float
#define T_OUT float
#define I 1024
#define K 128
// …
__shared__ T_INP M_1_1[2][I][K];
           T_INP v_1_1[2][K];
           T_OUT w_1_1[2][I];
// …

for(int p_1_2=0; p_1_2<4; ++p_1_2){
  int p_2_1 = threadIdx.x;{
    // …
  }
}
```

CUDA

Designed to be **straightforward**, allowing
easy extension to new target ML programming models

Overview

<u>End-to-end *ML Code-Generation* approach:</u>



Built around **three core IRs** with
automated transformations between them

[1] Rasch, "*(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms*", TOPLAS'24

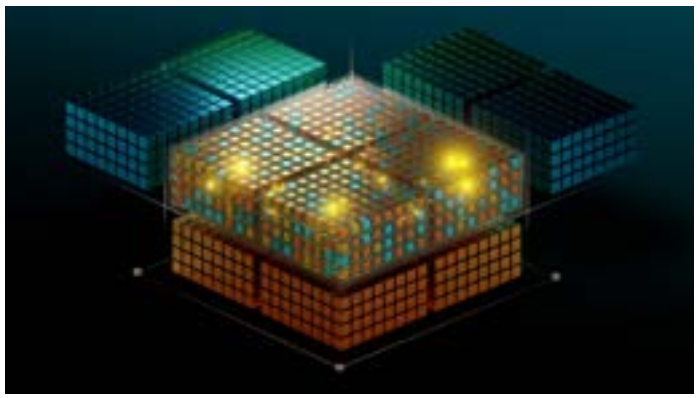
[2] Rasch, Schulze, Steuwer, Gorlatch, "*Efficient Auto-Tuning of Parallel Programs with Interdependent Tuning Parameters via Auto-Tuning Framework (ATF)*", TACO'21

Transformation: {...} → {...}

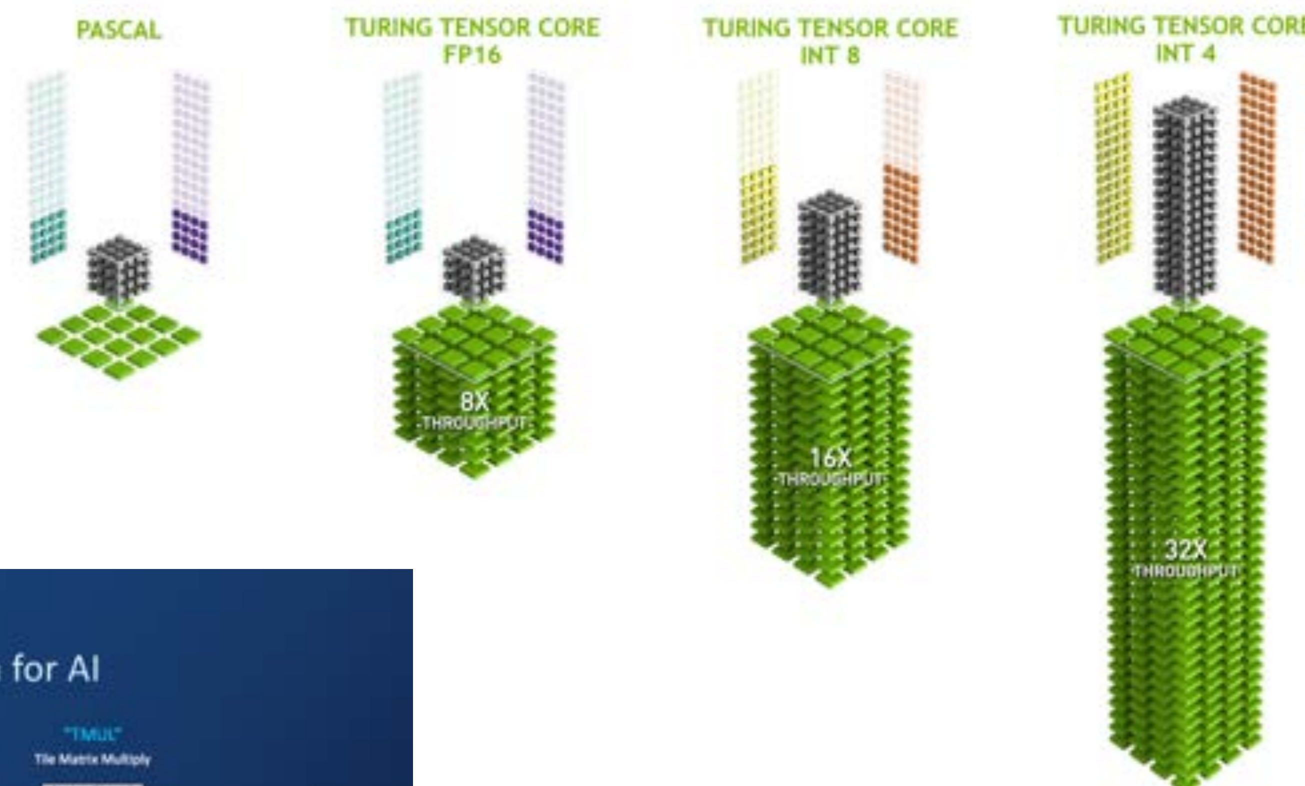
WIP

Final, target-specific optimizations:

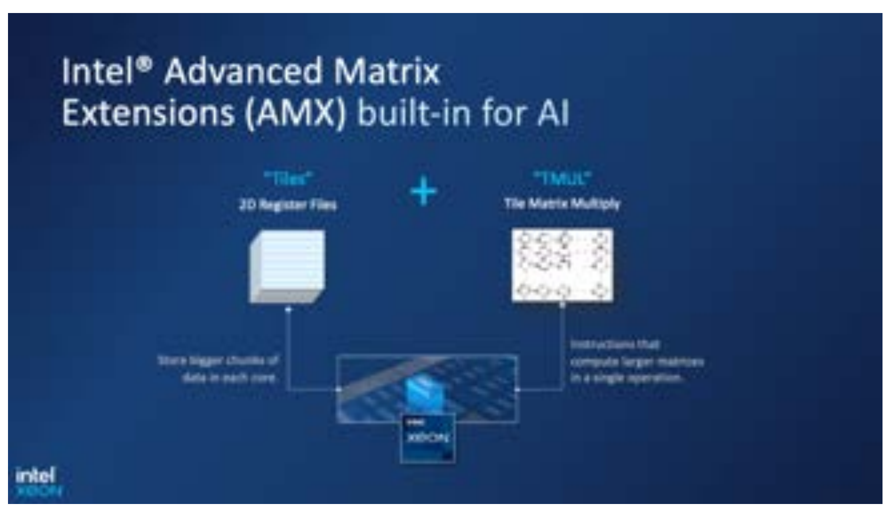
AMD
Matrix Cores



NVIDIA
Tensor Cores



Intel
AMX



Applies **target-specific optimizations** that
exceed the intended generality of the CL-IR abstraction

Experimental Results

Experimental evaluation in terms of ***Performance*** & *Portability* & *Productivity*:

Competitors:

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

ML Kernels:

1. Linear Algebra Routines:
   - Dot Product (`Dot`)
   - Matrix-Vector Multiplication (`MatVec`)
   - Matrix Multiplication (`MatMul`)
   - Matrix Multiplication Transposed (`MatMul^T`)
   - batched Matrix Multiplication (b`MatMul`)
2. Convolutions:
   - Multi-Channel Convolution (`MCC`)
   - Capsule-Style Convolution (`MCC_Capsule`)

		Data Characteristics		
Computation	**No.**	**Sizes**		**Basic Type**
`Dot`	1	2^{24}	2^{24}	fp32
	2	10^7	10^7	fp32
`MatVec`	1	4096x4096	4096	fp32
	2	8192x8192	8192	fp32
`MatMul`	1	1024x1024	1024x1024	fp32
	2	1x2048	2048x1000	fp32
`MatMul^T`	1	64x10	500x64	fp32
`bMatMul`	1	16x10x64	16x64x500	fp32
`MCC`	1	1x512x7x7	512x512x3x3	fp32
	2	1x230x230x3	64x7x7x3	fp32
`MCC_Caps`	1	16x230x230x3x4x4	64x7x7x3x4x4	fp32
	2	1x230x230x3x4x4	67x7x7x3x4x4	fp32

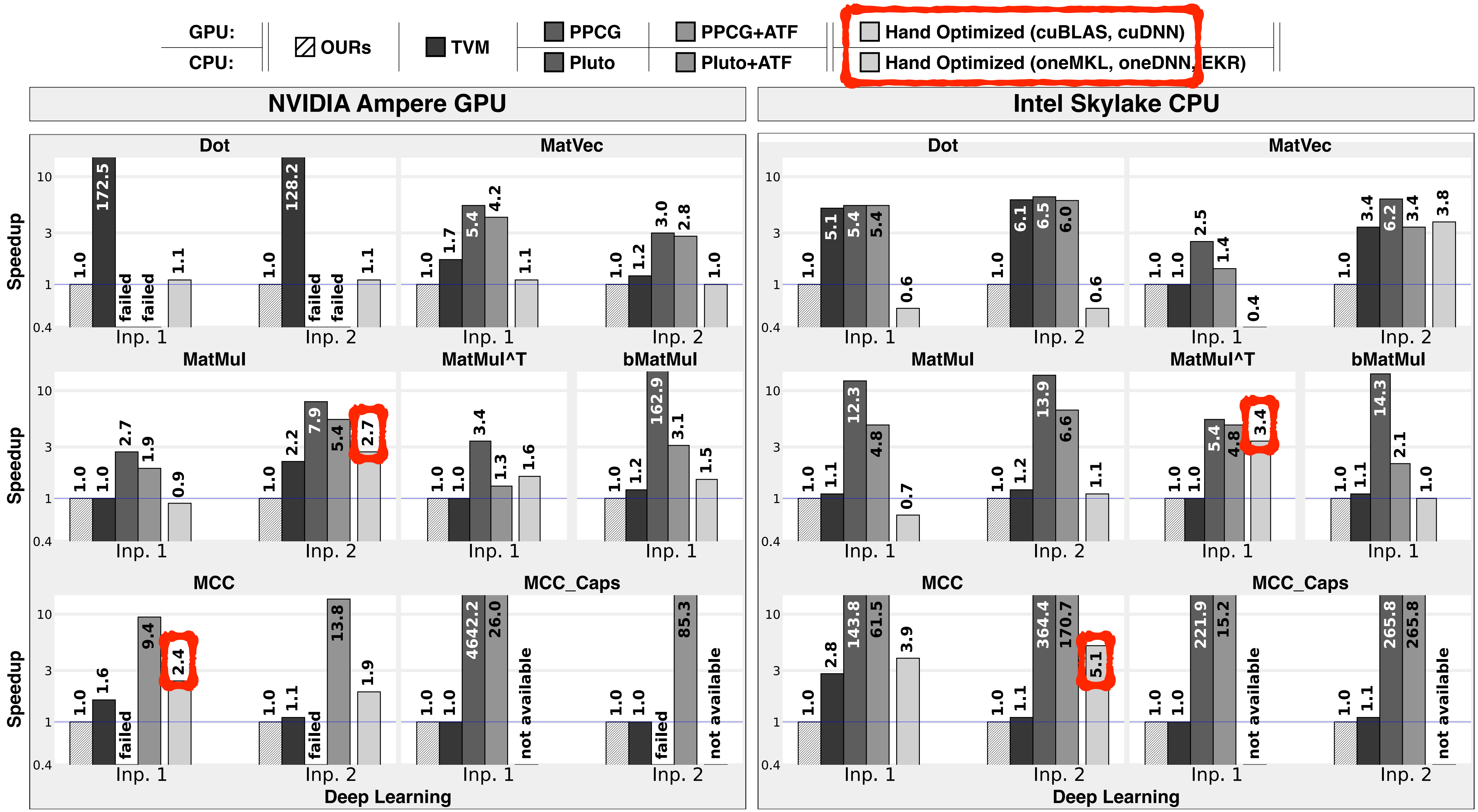
[1] Chen et al., “TVM: An Automated End-to-End Optimizing Compiler for Deep Learning”, OSDI’18

[2] Verdoolaege et al., “Polyhedral Parallel Code Generation for CUDA”, TACO’13

[3] Bondhugula et al., “PLuTo: A Practical and Fully Automatic Polyhedral Program Optimization System”, PLDI’08

Experimental Results

Performance results for ML kernels on GPU and CPU:



We achieve **high performance** compared to state-of-the-art **machine-** and **hand-optimized approaches**

Summary

We introduce a systematic ML code generation process:

- **Fully automatic**, by separating optimization concerns across abstraction levels
- Designed to be systematically **extensible for new target ML models** (Triton, etc)
- **Formal** foundation, based on algebraic MDH formalism
- **High performance** on **different architectures** (including GPUs and CPUs)

Future Work:

- Show how our design contributes to **aggressive kernel fusion** optimization
- Exploit **ML-specific hardware** extensions
- **Assembly**-level targets (PTX, …)
- **Sparse** Computations
- …

Universität Münster

Questions?

https://richardschulze.net
r.schulze@uni-muenster.de

Richard Schulze

https://arirasch.net
a.rasch@uni-muenster.de

Ari Rasch

https://mdh-lang.org

Code Generation

https://atf-tuner.org

Code Optimization