Universität Münster

Compilers for Machine Learning

5th C4ML workshop, at CGO 2024

Code Generation & Optimization for Deep-Learning Computations via Multi-Dimensional Homomorphisms

Ari Rasch, Richard Schulze, …

University of Muenster, Germany

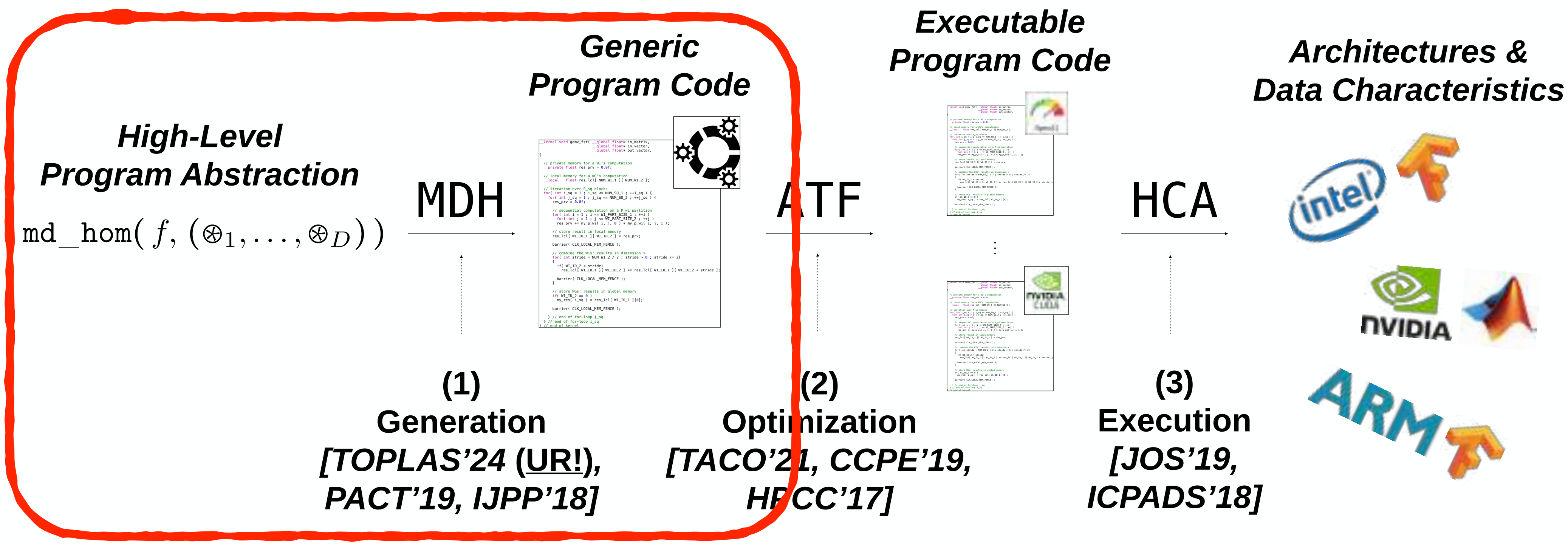
Who are we?

https://mdh-lang.org

https://atf-tuner.org

https://hca-project.org

We are the developers of the **MDH+ATF+HCA** approaches:



Focus Today

Richard Schulze

Ari Rasch

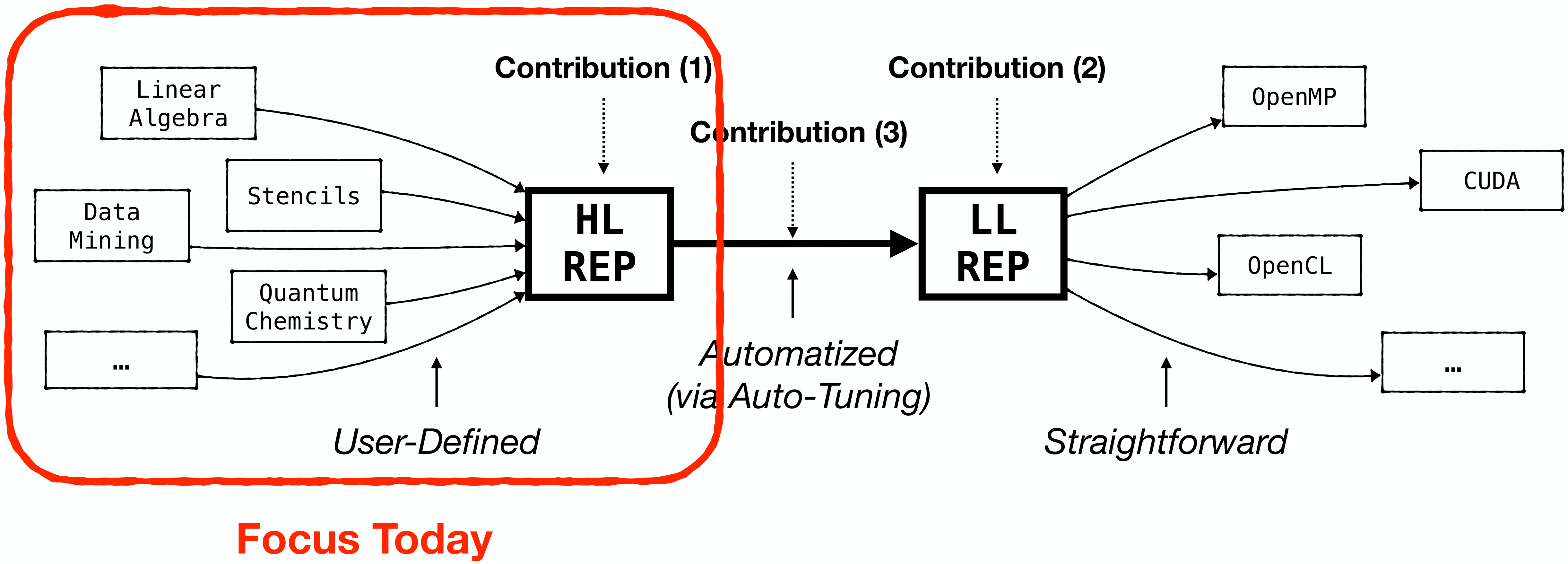
A holistic approach to code *generation* (MDH) & *optimization* (ATF) & *execution* (HCA):

(1) MDH (*Multi-Dimensional Homomorphisms*): How to generate automatically optimizable (auto-tunable) code?

(2) ATF (*Auto-Tuning Framework*): How to optimize (auto-tune) code?

(3) HCA (*Host Code Abstraction*): How to execute code on (distr.) multi-dev. systems?

The MDH Approach



The MDH approach [1] (formally) introduces:

(1) High-Level Program Representation for conveniently expressing data-parallel computations, agnostic from hardware and optimization details

(2) Low-Level Program Representation that expresses device- and data-optimized de- and re-composition strategies of computations & straightforwardly transformable to executable program code

(3) Lowering Process that *fully automatically* lowers a high-level MDH program to a device- and data-optimized low-level MDH program (based on auto-tuning [2])

[1] "(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms" (*under review at ACM TOPLAS*)
[2] "Efficient Auto-Tuning of Parallel Programs with Interdependent Tuning Parameters via Auto-Tuning Framework (ATF)", *TACO'21*

The MDH High-Level Representation

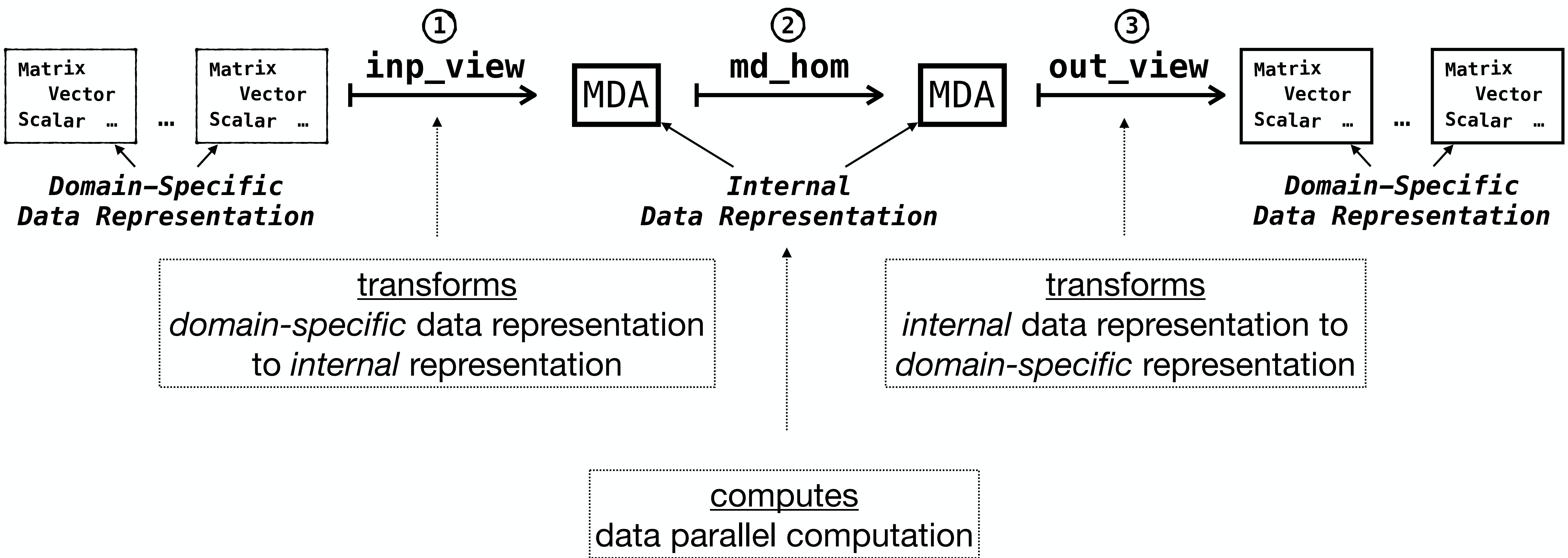
<u>Goals:</u>

1. **Uniform**:
   should be able to express any kind of data-parallel computation, but 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

While still capturing all information relevant for generating high performing program code, in a hardware- and data-agnostic manner

The MDH High-Level Representation

Overview:



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

The MDH High-Level Representation

The MDH's high-level program representation illustrated:

```
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) )
```

***MDH High-Level Representation*[1] *for* 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]*

The MDH High-Level Representation

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

CC-Based Operators
(computations specification)

Views	inp_view I_1	inp_view I_2	out_view 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	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$ $(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$ $\vdots$	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$ $(i_1,\dots,i_D)\mapsto(i_1,\dots,i_{k-1},i_{k+1},\dots,i_D)$ $\vdots$	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$ $(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$ $\vdots$
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	$(i)\mapsto()$	$(i)\mapsto()$	$(i)\mapsto(i)$

CC-Based Operators
(data-access specification)

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

CT-Based Operators
(computations specification)

Views	inp_view I_1	inp_view I_2	out_view 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>	$(i_1,\dots,i_D)\mapsto(i_1,\dots,i_D)$	/	$(i_1,\dots,i_D)\mapsto(i_2,\dots,i_D)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

CT-Based Operators
(data-access specification)

Our high-level representation is capable of expressing important DL operators

Experimental Results for DL Operators

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	–	–

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	–	–

Deep Learning	NVIDIA Volta GPU									
	ResNet-50				VGG-16				MobileNet	
	Training		Inference		Training		Inference		Training	Inference
	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MatMul	MCC	MCC
TVM+Ansor	0.75	1.21	0.72	1.79	1.00	1.11	1.06	1.00	1.00	1.00
PPCG	1976.38	5.88	–	5.64	994.16	3.41	8.21	2.51	1411.92	7.26
PPCG+ATF	3.43	3.54	3.42	4.93	3.85	3.15	8.13	2.05	3.49	3.56
cuDNN	1.21	–	1.29	–	2.80	–	3.50	–	2.32	3.14
cuBLAS	–	1.33	–	1.14	–	1.09	–	1.04	–	–
cuBLASEx	–	1.21	–	1.07	–	1.04	–	1.03	–	–
cuBLASLt	–	1.00	–	1.07	–	1.04	–	1.02	–	–

Deep Learning	Intel Broadwell 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	1.60	1.29	1.53	1.32	1.00	1.27	1.02	2.42	1.92
Pluto	4349.20	40.41	137.21	15.96	1865.07	53.57	113.40	24.10	2255.00	53.85
Pluto+ATF	6.43	8.93	61.60	6.91	5.07	4.38	42.63	4.45	6.43	29.18
oneDNN	1.30	–	1.81	–	2.94	–	2.85	–	1.83	4.47
oneMKL	–	1.45	–	1.36	–	1.35	–	0.50	–	–
oneMKL(JIT)	–	19.78	–	9.77	–	50.58	–	10.70	–	–

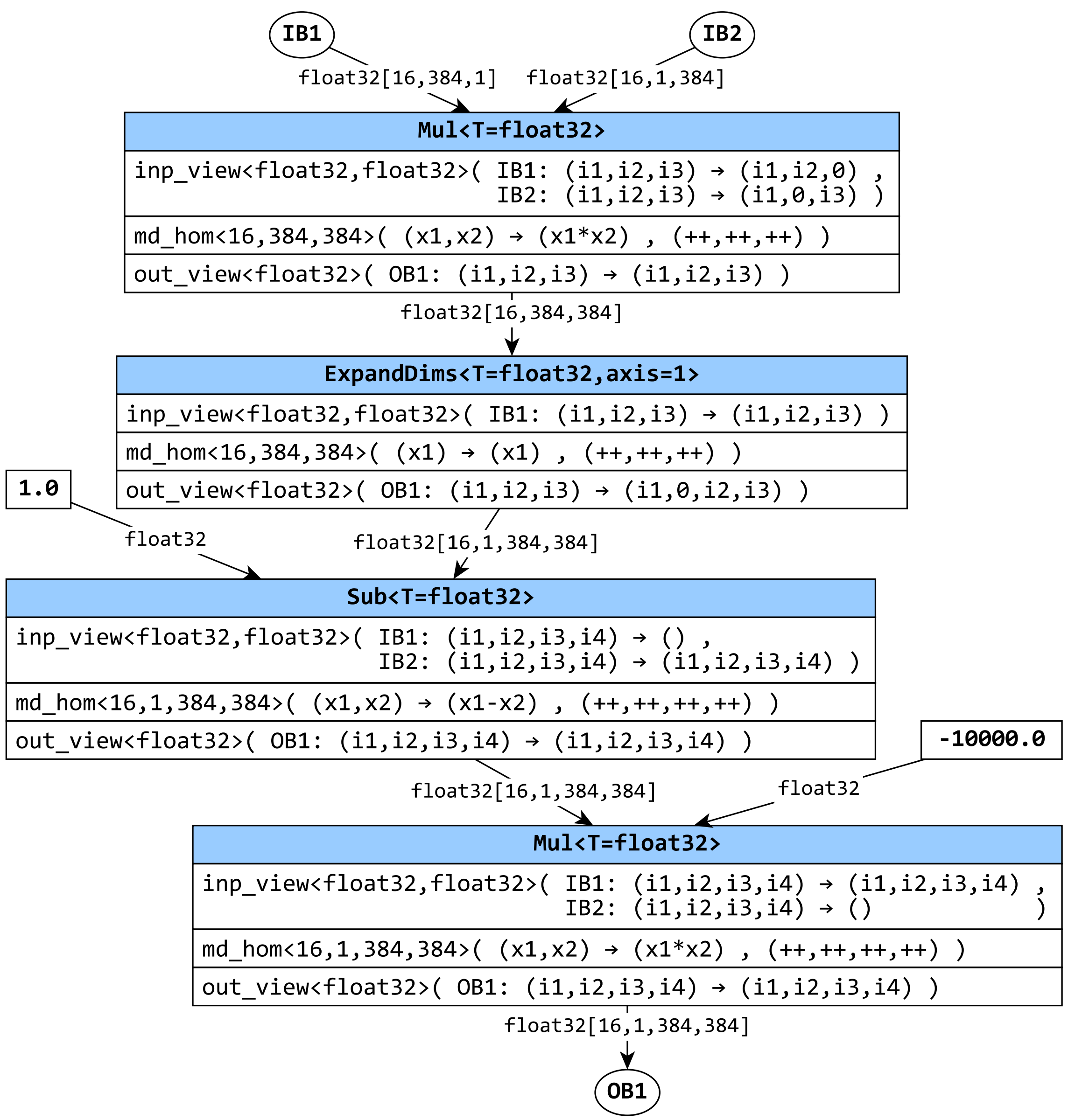
We achieve encouraging experimental results for DL Operators [1]

[1] "(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms" (*under review at ACM TOPLAS*)

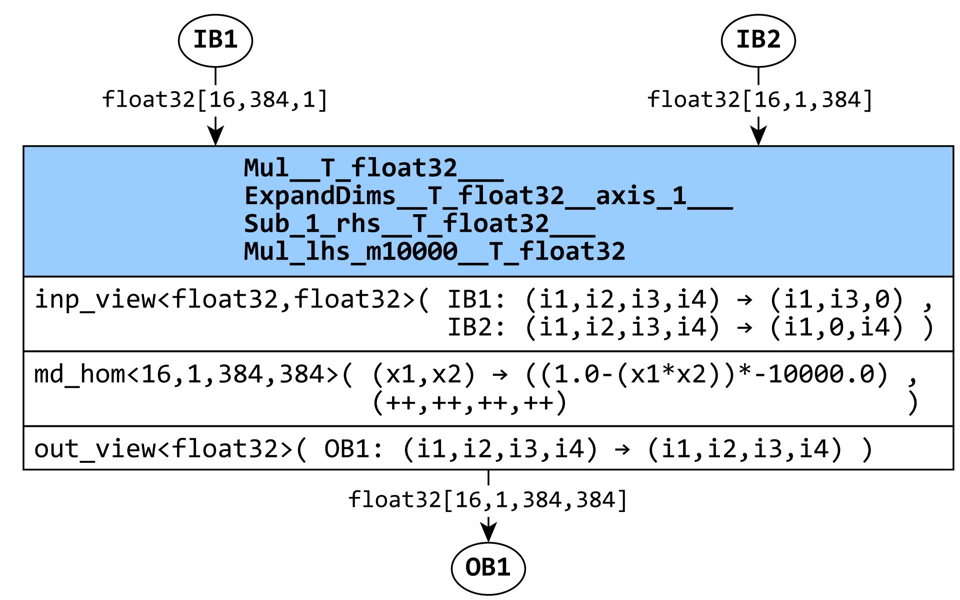
Excursion: MDH High-Level Optimization

WIP Results

An optimization is considered ***High Level*** iff it operators on ***MDH's High-Level Representation***:



b) MDH subgraph (naive)



c) MDH subgraph (fused)

We exploit the uniform MDH representation to analyze and fuse the DL graph

Experimental Results for DL Graphs

WIP Results

	NVIDIA Ampere GPU				
	Number of Operators	Operators Occurring in Subgraph	Runtime Share	Speedup over TF	
				MDH	TC
1.	13	(Sub,1),(Mul,5),(AddV2,4),(RealDiv,1),(Sqrt,1),(Square,1)	25.96%	65.93	45.72
2.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	11.40%	30.99	30.50
3.	41	(BiasAddGrad,1),(AddN,1),(Mul,17),(TanhGrad,2),(Pow,4),(AddV2,4),(Tanh,3),(BiasAdd,9),(Sub,1)	3.79%	1.41	0.05
4.	3	(Mul,1),(Reshape,1),(AddV2,1)	3.60%	23.79	14.72
5.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	2.87%	6.79	6.73
6.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	2.70%	6.36	6.24
7.	2	(BiasAddGrad,1),(Reshape,1),(Transpose,1)	2.64%	9.86	0.57
8.	5	(BiasAddGrad,1),(Mul,2),(Reshape,1),(Cast,1),(GreaterEqual,1)	2.45%	4.55	1.60
9.	13	(Sub,1),(Mul,5),(AddV2,4),(RealDiv,1),(Sqrt,1),(Square,1)	2.40%	39.57	36.93
10.	9	(AddV2,2),(Mul,3),(Cast,1),(BiasAdd,1),(GreaterEqual,1),(Reshape,1)	1.47%	1.63	1.60
			Total Speedup over TF:	**2.29**	**0.79**

	Intel Skylake CPU			
	Number of Operators	Operators Occurring in Subgraph	Runtime Share	Speedup over TF
				MDH
1.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	17.33%	571.14
2.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	7.20%	248.81
3.	9	(AddV2,2),(Mul,3),(Cast,1),(BiasAdd,1),(GreaterEqual,1),(Reshape,1)	6.94%	110.51
4.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	6.06%	199.32
5.	8	(Mul,4),(Cast,1),(Softmax_Div,1),(GreaterEqual,1),(AddV2,1)	5.45%	12.76
6.	11	(Mul,6),(Sub,1),(Softmax_Div,1),(AddV2,1),(Cast,1),(GreaterEqual,1)	5.20%	10.81
7.	41	(BiasAddGrad,1),(AddN,1),(Mul,17),(TanhGrad,2),(Pow,4),(AddV2,4),(Tanh,3),(BiasAdd,9),(Sub,1)	3.71%	2.82
8.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	1.50%	24.77
9.	2	(Transpose,1),(Reshape,1)	0.59%	12.08
10.	15	(Sub,1),(Mul,6),(AddV2,5),(RealDiv,1),(Sqrt,1),(Square,1)	0.42%	875.92
			Total Speedup over TF:	**2.11**

We achieve encouraging experimental results also for DL Graphs with MDH

Excursion: MDH in MLIR

`MLIR` is a compiler framework that offers a solid, uniform infrastructure for compiler developers to conveniently design and implement *Domain-Specific Languages (DSLs)* (a.k.a. *dialect* in MLIR terminology)

Implemented by Jens & Lars Hunloh (University of Muenster, Germany)

```
func.func @main()
{
  %M = memref.alloc() : memref<128x64xf32>
  %v = memref.alloc() : memref<64xf32>

  %w = mdh.compute "mdh_matvec"
  {
    inp_view =
    [
      [ affine_map<( i,k ) -> ( i,k )> ],
      [ affine_map<( i,k ) -> ( k )   > ]
    ],

    md_hom =
    {
      scalar_func = @mul,
      combine_ops = [ "cc", ["pw",@add] ]
    },

    out_view =
    [
      [ affine_map<( i,k ) -> ( i )> ]
    ]
  }
  {
    inp_types = [ f32, f32 ],
    mda_size  = [ 128,64 ],
    out_types = [ f32 ]
  }( %A,%B ):( memref<128x64xf32>,memref<64xf32> )
              -> memref<128xf32>

  return
}
```

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

MDH

WIP Results

WESTFÄLISCHE WILHELMS-UNIVERSITÄT MÜNSTER

We look forward to discussions at the poster session

https://mdh-lang.org

https://atf-tuner.org

https://hca-project.org

Richard Schulze
r.schulze@uni-muenster.de

Ari Rasch
a.rasch@uni-muenster.de