WWU MÜNSTER

STUDENT RESEARCH COMPETITION

# `md_stencil`: High-Performance Stencil Computations on CPU and GPU via Multi-Dimensional Homomorphisms

Ari Rasch and Sergei Gorlatch

WIP Results

# Goals

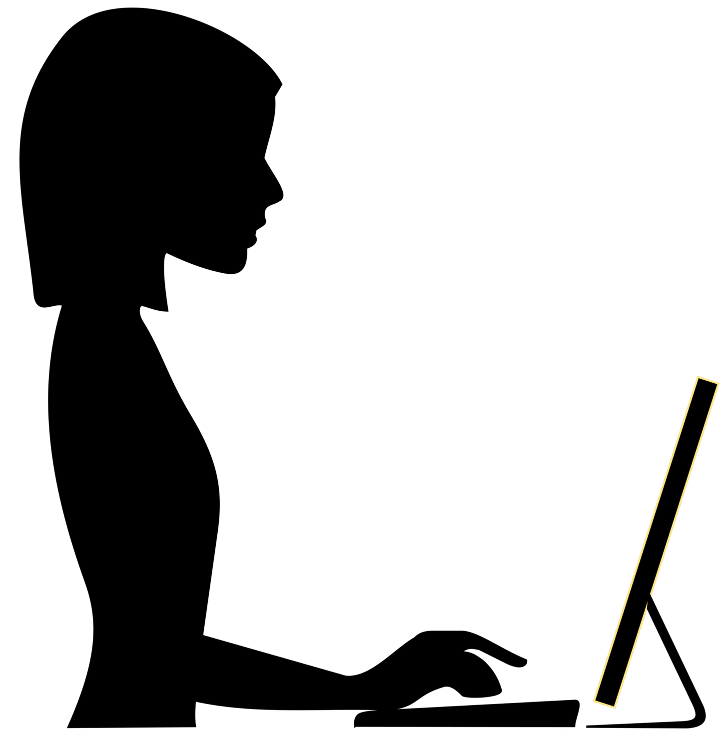
We aim to achieve for stencil computations <u>in one approach</u> three major goals:

***Performance***

competitive to best available solutions

***Portability***

functional and performance — over architectures and input/output characteristics



***Productivity***

easy to use & extensible

# Approach


1. **Transforming DSL programs to MDH representation.**

2. Generating auto-tunable OpenCL code from MDH representation.

3. Auto-tuning OpenCL code for target device and input/output char.

4. Executing auto-tuned OpenCL code.

*[1] Rasch, Schulze, Gorlatch, "Generating Portable High-Performance Code via Multi-Dimensional Homomorphisms.", PACT'19*

*[2] Rasch, Haidl, Gorlatch, "ATF: A Generic Auto-Tuning Framework.", HPCC'17*

*[3] Rasch, Gorlatch, "ATF: A Generic, Directive-Based Auto-Tuning Framework.", CCPE'19*

*[4] Rasch, Wrodarczyk, Schulze, Gorlatch, " OCAL: An Abstraction for Host-Code Programming with OpenCL and CUDA.", ICPADS'18*

*[5] Rasch, Bigge, Wrodarczyk, Schulze, Gorlatch. "dOCAL: High-Level Distributed Programming with OpenCL and CUDA.", JOS'19*

# Transformation: DSL → MDH

The MDH Representation relies on three higher-order functions (patterns):

1. `in_view` → uniformly prepares stencil-specific *input data*
2. `md_hom` → specifies stencil *computation*
3. `out_view` → uniformly prepares stencil-specific *output data*

<u>Example:</u> Conv 2D → *`conv2d = out_view( … ) o md_hom( … ) o in_view( … )`*

```
1. in_view( im, w )( p,q , r,s )( in[ p+r , q+s ], w[r,s] )
```

input image

weight matrix

indices for input image

indices for weight matrix

data accesses

```
2. md_hom( *, (++,++,+,+) )
```

multiplie elements in `in` and `w`

sum in dimensions `r` & `s`

concatenate in dimensions `p` & `q`

```
3. out_view( out )( p,q )( out[ p,q ] )
```

output image

indices for output image

data accesses

# Preliminary Results

**Hardware**
- CPU: Intel Xeon E5
- GPU: NVIDIA V100

***Lift [6]:*** *1.9x-4.9x on CPU and 1.02x-2.34x on GPU for `conv2d` and `j3d7pt` on Lift's own data sets*

Best Paper CGO'18

***TVM [7]:*** *2.75x on GPU for `MCC` on their own real-world data set from deep learning*

Speedups of `md_stencil` over well-performing **machine-** and **hand-optimized approaches** on CPU and GPU

***Artemis [8]:*** *0.98x-1.07x on GPU for `conv2d` and `j3d7pt`*

***Intel MKL-DNN / NVIDIA cuDNN:*** *1.3x on CPU and 3.31x on GPU for `MCC` on TVM's real-world data set*


[6] Hagedorn, et al., "High Performance Stencil Code Generation with Lift.", CGO'18, **(Best Paper Award)**

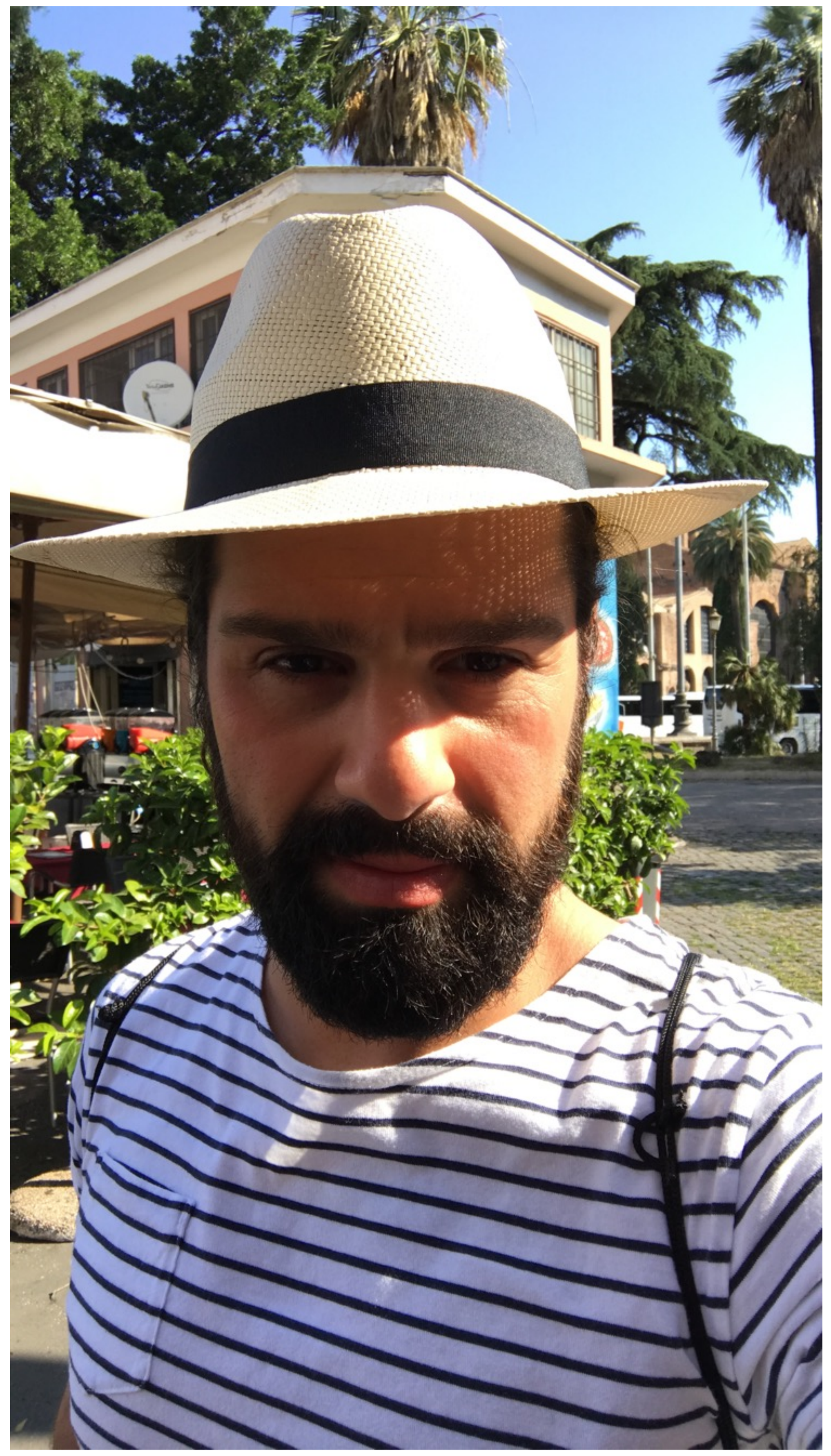
[7] Chen, et. al, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning", OSDI'18

[8] Rawat, et. al, "On Optimizing Complex Stencils on GPUs", IPDPS'19

WIP Results

# Questions?

*Grateful for any feedback*



**Ari Rasch**
a.rasch@wwu.de

*This presentation and recording belong to the authors.*
*No distribution is allowed without the authors' permission.*

# Appendix

We have two next major steps:

1. <u>Faster Auto-Tuning:</u> exploit stencil-specific, high-level information.

2. <u>Further Stencils:</u> generalized convolutions (capsule networks), etc.

# Appendix

## Further Stencils:

- *Conv 2D transposed* (`conv2d-trans`):

```
md_hom( *, (++, ++, +,+) ) o in_view( in, weights )( p,q , r,s )( in[
q+s, p+r ], weights[r,s] )
```

- *Jacobi 3D* (`j3d7pt`):

```
md_hom( j_f, (++,++,++) ) o in_view( in )( i,j,k )
( in[i,j,k],…,in[i+2,j+2,k+2] )
```

, where `j_f` is the jacobi transition function

- *Multi-Channel Convolution* (`MCC`):

```
md_hom( *, (++,++,++,++,+,+,+) ) o in_view( in, weights )
( n,k,p,q,c,r,s )( in[ n,c,p+r,q+s], weights[ k,c,r,s] )
```

- *1x1 convolution* (`map-n`):

```
md_hom( f, (++,…,++) ) o in_view( A )( i_1,…,i_n )( A[i_1,…,i_n] ),
```

where `f` is the transition function.

# Appendix

Capsule Networks

“Machine Learning Systems are Stuck in a Rut” [HotOS’19]:

$$\forall n, x, y, c_o : \boldsymbol{V}^{n,c_o}_{x,y} = \sum_{k_x} \sum_{k_y} \sum_{c_i} \boldsymbol{P}^{n,c_i}_{sx+k_x,\, sy+k_y} \cdot \boldsymbol{W}^{c_i,c_o}_{k_x,k_y}$$

```
conv2d-gen( … ) =

   in_view( P,W )( n,x,y,c0 , kx,ky,ci )( P[ n,ci , s*x+kx,
                                               s*y+ky ], W[ci,c0 , kx,ky] )

                              md_hom( •, (++,++,++,++ , +,+,+) )

  out_view( V )( n,x,y,c0 )( V[ n,c0,x,y ] )
```