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

WIP Results

a.rasch@wwu.de

Ari Rasch, Richard Schulze, and Sergei Gorlatch

WWU MÜNSTER

Goals

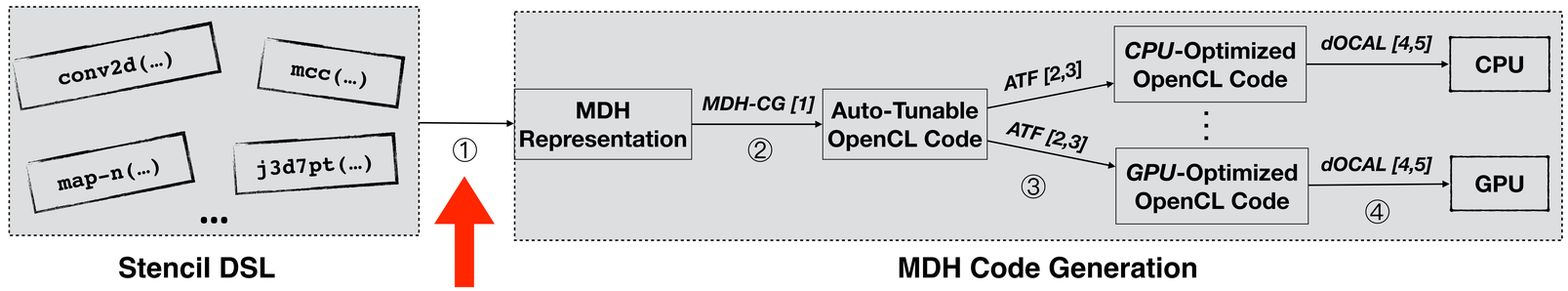
We aim to achieve for stencil computations in one approach three major goals:

Portability — functional and performance — over architectures and input/output characteristics

Performance — competitive to best available solutions

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, Schulze, Steuwer, Gorlatch, "Efficient Auto-Tuning of Parallel Programs with Interdependent Tuning Parameters via Auto-Tuning Framework ATF", TACO'20 (accepted)

[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*

Example: Conv 2D (`conv2d`)

```
1. in_view( im, w )( p,q , r,s )( in[ p+r , q+s ], w[r,s] )
2. md_hom( *, (++,++,+,+) )
3. out_view( out )( p,q )( out[ p,q ] )
```

- `im`: input image
- `w`: weight matrix
- `p,q`: indices for input image
- `r,s`: indices for weight matrix
- `in[ p+r , q+s ]`, `w[r,s]`: data accesses
- `*`: multiplie elements in `in` and `w`
- `++,++,+,+`: concatenate in `p` & `q` dimension; sum in `r` & `s` dimension
- `out`: output image
- `p,q`: indices for output image
- `out[ p,q ]`: data accesses

⇨ **`conv2d = out_view( … ) o md_hom( … ) o in_view( … )`**

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*

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**

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