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

Ari Rasch (supervisor: Sergei Gorlatch), University of Muenster, Germany

I. Introduction

Stencils are important in popular fields, ranging from machine learning to physical simulations. Code generation and optimization for stencils is the focus of many important projects; however, the increasing heterogeneity in state-of-the-art parallel systems requires rethinking the traditional approaches which are usually designed and optimized toward particular architectures only.

Our poster presents work-in-progress results for `md_stencil` – a novel approach, implemented as a library, toward fully automatic code generation and optimization for stencil computations targeting different kinds of modern architectures: multi-core CPU, GPU, etc. Our approach relies on the algebraic formalism of Multi-Dimensional Homomorphisms (MDH) and their code generation mechanism in OpenCL [PACT'19]. We demonstrate how important stencil computations are expressed as MDHs, which enables `md_stencil` to internally exploit the existing code generation and optimization mechanism for MDHs, while providing to the user a standard, high-level stencil user interface. Our preliminary experiments are encouraging: we show for real-world stencil computations that `md_stencil` achieves better performance on both CPU and GPU as compared to state-of-practice hand- and machine-optimized code: Intel MKL-DNN for CPU and NVIDIA cuDNN for GPU, as well as the popular approaches Lift [CGO'18], TVM [OSDI'18], and Artemis [IPDPS'19] on their own stencil examples and data sets.

II. Approach

The main challenge addressed by our poster is expressing important stencil computations in the MDH formalism, which enables generating and optimizing code for them based on the MDHs' existing mechanisms. A general, straightforward MDH stencil representation is presented in our previous work [PACT'19]; in contrast, this poster introduces optimized MDH representations for important classes of stencils. For this, we first briefly recapitulate the MDHs' formalism using the popular stencil computation *Gaussian Convolution 2D* (`conv2d`): `md_hom( ∗, (++, ++, +,+) ) o view(in, weights) ( p,q , r,s )( in[ p+r, q+s ], weights[r,s] )`. Here, we first use pattern `view` of MDHs' formalism to prepare `conv2d`'s domain-specific input – an `(P+4)x(Q+4)` input image `in` and a `5x5` matrix `weights` – as array of size `PxQxRxS`: it contains at position `p,q,r,s` the neighbor r,s of element `p,q` in image `in`, as well as the neighbor's corresponding weight. Afterwards, the `md_hom` pattern specifies that elements in the array are summed up in dimension `R` and `S` (denoted by "+") and concatenated in the two remaining dimensions `P` and `Q` (denoted by "++").

III. Popular Stencils as MDHs

We present new MDH representations for stencils, including stencils belonging to four popular classes with very different characteristics (dimensionality, transition function, etc.):

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

  `md_hom( ∗, (++, ++, +,+) ) o view( in, weights )( p,q , r,s )( in[ q+s, p+r ], weights[r,s] )`

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

  `md_hom( j_f, (++,++,++) ) o 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 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 view( A )( i_1,…,i_n )( A[i_1,…,i_n] )`, where `f` is the transition function.

IV. Experimental Results

Our preliminary experimental results on Intel Xeon CPU and NVIDIA V100 GPU show competitive and often better performance than competitors – speedups of `md_stencil` over:

- Lift: 1.9x-4.9x on CPU and 1.02x-2.34x on GPU for `conv2d` and `j3d7pt` on Lift's own data sets;
- TVM: 2.75x on GPU for `MCC` on their own real-world data set from deep learning;
- Artemis: 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.

We achieve better performance than Lift and TVM, because MDHs' code generation mechanism relies on larger optimization spaces, which allows more-fine grained code optimizations; for a fair comparison, we use for `md_stencil` the same auto-tuning time that Lift uses for stencils – 5h. In contrast to Artemis, we target also CPUs – by relying on OpenCL, rather than CUDA. Compared to the hand-optimized libraries by Intel and NVIDIA, our performance is better, because we rely on auto-tuning while the libraries use hand-crafted heuristics for optimization – thereby, the libraries avoid the overhead for auto-tuning which sometimes might not be amortized.