md_poly: A Performance-Portable Polyhedral Compiler Based on Multi-Dimensional Homomorphisms

Ari Rasch
a.rasch@wwu.de
University of Muenster (Germany)

Sergei Gorlatch
gorlatch@wwu.de
University of Muenster (Germany)

Abstract

Polyhedral compilers automatically parallelize sequential programs (e.g., written in the C programming language) for multi- and many-core architectures, such as CPU and GPU. However, parallel code generated by state-of-the-art polyhedral compilers (e.g. PPCG) often lacks performance portability, because the existing compilers are usually optimized toward only a single particular parallel architecture (e.g., GPU). Moreover, even on their target architecture, polyhedral compilers sometimes tend to fail reaching high performance, because they often miss important optimizations, e.g., efficiently exploiting fast memory resources.

We present our work-in-progress results for `md_poly` – a novel polyhedral compiler that generates portable high-performance code from sequential C programs with perfect loop nests and rectangular iteration spaces. In contrast to the existing polyhedral compilers, `md_poly`'s code generation approach relies on Multi-Dimensional Homomorphisms (MDHs): we transform the internal program representation of polyhedral compilers (a.k.a. *polyhedral model*) automatically to an equivalent MDH representation which is suitable for generating portable high-performance program code for CPUs and GPUs. Our preliminary experimental comparison against PPCG – for benchmarks *Gaussian Convolution* and *Matrix Multiplication* – shows encouraging results: speedups up to 7× on Intel CPU and 3× on NVIDIA GPU using real-world input sizes from deep learning.

1 Overview

Figure 2 demonstrate the overview of `md_poly`'s internal design. Starting from a sequential C program – currently limited to C programs with perfect loop nests and rectangular iteration spaces – we first extract in step 1 in the figure the polyhedral model – this is same step in all C-based polyhedral compilers – using the *Polyhedral Extraction Tool* (`pet`) [9]. Afterwards, we transform in step 2 the extracted polyhedral model into an equivalent MDH representation [6] – this is a novel transformation step which interconnects the polyhedral approach with the recent MDH formalism. The MDH representation is suitable for generating portable high-performance code [8]: we use the MDHs' code generator (MDH-CG) [8] in step 3 to transform the MDH representation into an automatically optimizable (auto-tunable) OpenCL code; the generated code is then auto-tuned in step 4 for different architectures and input sizes using the *Auto-Tuning Framework (ATF)* [7]. We execute the generated and auto-tuned OpenCL code in step 5 using the dOCAL framework [5].

2 Experimental Evaluation

Figure 1 shows the speedup of `md_poly`'s generated code – for benchmarks *Gaussian Convolution* (left) and *Matrix Multiplication* (right) – over PPCG and hand-optimized vendor libraries (VL). As VLs, we use Intel MKL-DNN [1] and NVIDIA cuDNN [3] for Gaussian Convolution; for Matrix Multiplication, we use Intel MKL [2] and NVIDIA cuBLAS [4]. We experiment on both Intel Xeon CPU and NVIDIA V100 GPU. As input sizes, we use i) real-world sizes (abbreviated with RW in the figure) from deep learning, and ii) sizes that are preferable for PPCG (abbreviated with PP), e.g., large powers of two. We auto-tune both the programs generated by `md_poly` and the optimization parameters of PPCG for 48h – the wall time of our system – using the *Auto-Tuning Framework (ATF)* [7].

	Gaussian Convolution				Matrix Multiplication			
	CPU		GPU		CPU		GPU	
	RW	PP	RW	PP	RW	PP	RW	PP
PPCG	7.78	4.75	3.70	1.13	2.03	4.58	1.01	1.03
VL	1.30	14.31	3.31	19.11	2.24	0.73	1.67	0.76

Figure 1. Speedup (higher is better) of the `md_poly`-generated OpenCL code over: i) PPCG, and ii) hand-optimized vendor libraries (VL).

We observe competitive and often better performance of `md_poly` than both PPCG and vendor libraries. As compared

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

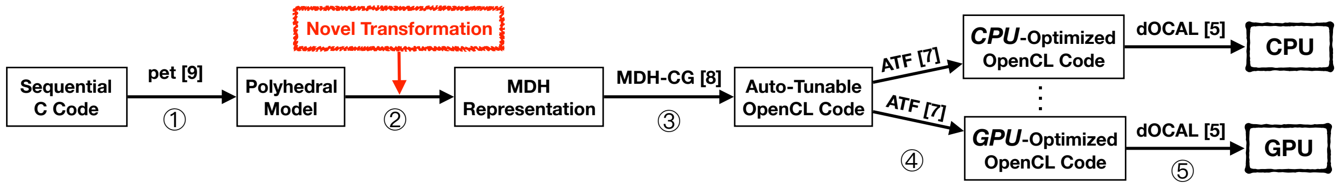

Figure 2. Overview of `md_poly`'s internal design.

to PPCG, `md_poly`'s better performance is because our generated OpenCL code has more tuning parameters than PPCG, e.g., parameters for enabling/disabling using OpenCL's fast local and private memory [8]; thereby, we enable a more fine-grained optimization of our generated code. In comparison to vendor libraries, we rely on auto-tuning, while the libraries use hand-crafted heuristics.

References

[1] Intel. 2018. Math Kernel Library for Deep Learning Networks. https://software.intel.com/en-us/articles/intel-mkl-dnn-part-1-library-overview-and-installation

[2] Intel. 2019. Math Kernel Library. https://software.intel.com/en-us/mkl

[3] NVIDIA. 2018. CUDA Deep Neural Network library. https://developer.nvidia.com/cudnn

[4] NVIDIA. 2019. cuBLAS library. https://developer.nvidia.com/cublas

[5] Ari Rasch, Julian Bigge, Martin Wrodarczyk, Richard Schulze, and Sergei Gorlatch. 2019. dOCAL: high-level distributed programming with OpenCL and CUDA. *The Journal of Supercomputing* (30 Mar 2019). https://doi.org/10.1007/s11227-019-02829-2

[6] Ari Rasch and Sergei Gorlatch. 2018. Multi-dimensional Homomorphisms and Their Implementation in OpenCL. *International Journal of Parallel Programming* 46, 1 (01 Feb 2018), 101–119. https://doi.org/10.1007/s10766-017-0508-z

[7] Ari Rasch and Sergei Gorlatch. 2019. ATF: A generic directive-based auto-tuning framework. *Concurrency and Computation: Practice and Experience* 31, 5 (2019), e4423. https://doi.org/10.1002/cpe.4423 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4423 e4423 cpe.4423.

[8] A. Rasch, R. Schulze, and S. Gorlatch. 2019. Generating Portable High-Performance Code via Multi-Dimensional Homomorphisms. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 354–369. https://doi.org/10.1109/PACT.2019.00035

[9] Sven Verdoolaege and Tobias Grosser. 2012. Polyhedral Extraction Tool. In *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*. Paris, France. http://impact.gforge.inria.fr/impact2012/workshop_IMPACT/verdoolaege.pdf