Expressing Hierarchical Code Optimizations via MDH-Based Schedules

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

Richard Schulze
University of Muenster, Germany
r.schulze@wwu.de

Sergei Gorlatch
University of Muenster, Germany
gorlatch@wwu.de

I. Introduction & Related Work

Program code in state-of-the-art low-level approaches, like CUDA and OpenCL, requires complex, hierarchical optimizations to efficiently target the deep and complex memory and core hierarchies of modern architectures, such as GPU and multi-core CPU. Modern compilers [1]–[6] automatically generate well-performing low-level code from sequential programs; the optimization processes of these compilers are often manually guided by a performance expert who explicitly expresses low-level code optimizations (like tiling and parallelization) in form of programs in a so-called *scheduling language*. While such compilers with an expert-guided optimization process have high performance potentials, their scheduling languages usually consist of a set of fine-grained, low-level commands which have to be composed by the performance expert in complex ways for expressing hierarchical optimizations, making the optimization process complex, cumbersome, and error-prone for the expert.

We present our work-in-progress results toward a novel compiler whose scheduling language is based on the approach of *Multi-Dimensional Homomorphisms (MDH)* [7]. We argue that our MDH-based scheduling language enables a structured, hierarchical code optimization process, by offering scheduling commands that systematically de- and re-compose computations to/from the memory and core hierarchies of state-of-the-art architectures (GPUs, multi-core CPUs, etc). Thereby, the performance expert expresses hierarchical code optimizations in a concise and structured way, contributing to a simplified code optimization process for the expert. To further simplify the optimization process for the expert, we have integrated the notion of auto-tuning into our language design (e.g., for automatically identifying optimized tile size values), based on the *Auto-Tuning Framework (ATF)* [8]. Moreover, by relying on the MDH formalism and its algebraic foundation, we can mathematically guarantee the expert the correctness of optimizations expressed in our language.

Our first experiments on NVIDIA GPU and Intel CPU show that our scheduling language is capable of expressing optimization decisions of the popular deep learning compiler TVM [2] (which suffers from weaknesses listed above) as hierarchical code optimizations. Our experiments also confirm that via auto-tuning, we are able to achieve better performance than TVM on both architectures.

II. Overview

Figure 1 shows the overview of our approach. The original work on MDHs takes as input a sequential C program; the program is then transformed via different tools [7]–[11] to executable program code, e.g., for GPU or CPU.

In this work, we extend the existing MDH workflow in Figure 1 by the parts highlighted in red in the figure: we enable expert users to incorporate expert knowledge about optimizations into the workflow, by introducing *MDH-based schedules*. Our schedules conveniently express MDH optimizations, e.g., exploiting fast memory resources and parallelization, in a structured, hierarchical way (discussed in the next section). The user decisions are then incorporated in step ③ into the generated code, rather than generating the code in step ③ as generic in these decisions and requesting them later in step ④ from the auto-tuner (as done in the original MDH work). By incorporating the user into the optimization process, we enable both: 1) possibly better optimization, as the auto-tuning system might not always make the same high-quality optimization decisions as a human expert; 2) faster auto-tuning process, as some (or even all) optimization decisions are possibly made by the user and thus do not require costly auto-tuning.

III. MDH-Based Schedules

We illustrate our MDH-based scheduling language by showing how it is used for expressing the particular optimization decisions of the TVM compiler; the decisions are made by TVM's recent *Ansor* [12] optimization engine. As our case study, we use *Matrix Multiplication (`MatMul`)* on a real-world input size taken from the *ResNet-50* [13] neural network when computing the popular *ImageNet* [14] data set – a case study for which TVM is specifically designed and optimized. To express `MatMul` in our approach, we provide our compiler: 1) a straightforward implementation of `MatMul` in the C programming language; 2) a program in our scheduling language describing to our compiler the optimizations to be performed.

Listing 1 shows a program in our scheduling language – it expresses TVM's optimization decisions for NVIDIA A100 GPU when computing `MatMul` on input matrices of sizes 1×2048 and 2048×1000 taken from ResNet-50's most time-intensive `MatMul` computation (inference phase).

Our scheduling program hierarchically de- and re-composes `MatMul` to the GPU's memory and core hierarchies, in 3 steps

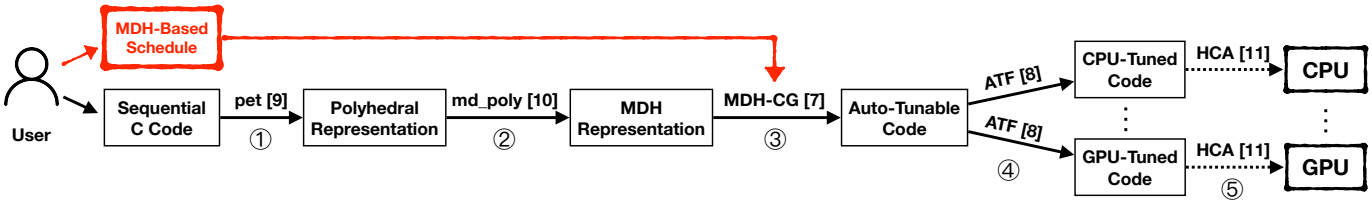

Fig. 1. Overview of our approach (contribution of this work highlighted in red)

```
1  // initialization
2  0: (de-)comp( 1,1000,2048 )
3                ( A:DM[1,2] , B:DM[1,2] ; C:DM[1,2] )
4                ( GPU.x,GPU.y,GPU.z )
5
6  // parallelization over CUDA Blocks
7  1: (de-)comp( ^,20,^ )
8                ( ^, ^ ; ^ )
9                ( BLK.y,BLK.x,^ )
10
11 // using CUDA Shared & Register Memory
12 2: (de-)comp( ^,^,512 )
13               ( A:SM[1,2] , B:SM[1,2] ; C:RM[1,2] )
14               ( ^,^,^ )
15
16 // parallelization over CUDA Threads
17 3: (de-)comp( ^,1,4 )
18               ( ^ , ^ ; ^ )
19               ( THR.y,THR.x,^ )
```

Listing 1. MDH schedule expressing TVM+Ansor optimizations

discussed in the following. Lines 1-4 in Listing 1 are optional in our approach and serve for completeness only.

a) Block Parallelization (lines 6-9): The original iteration space[1] (line 2) is split in tiles of size $1 \times 20 \times 2048$ (line 7); symbol ^ (a.k.a. *caret*) in line 7 indicates that the tile size of the previous step (line 2) is re-used in the first and third dimension, and in line 8 that no data layout changes or copy operations are performed. Line 9 indicates that the computations of tiles should be parallelized over *CUDA Blocks* (`BLK`) in the first and second dimension.

b) Shared & Register Memory Utilization (lines 11-14): Each of the $(1 \times 20 \times 2048)$-sized tiles of the previous step is split hierarchically into further tiles of size $1 \times 20 \times 512$; for each of these new tiles, line 13 indicates that the corresponding parts of input matrices `A` and `B` should be copied to CUDA's fast *Shared Memory* (`SM`), and that the result of the computed parts should be stored in even faster *Register Memory (RM)*;

c) Thread Parallelization (lines 16-19): Each tile of the previous step is again split in tiles of size $1 \times 1 \times 4$ that should be computed in parallel by *CUDA Threads* (`THR`) (line 19).

The correctness of our scheduling programs are statically verified, by checking the formal constraints defined by the MDH formalism [15] (not discussed for brevity).

IV. Auto-Tuning

Our scheduling language has integrated the notion of auto-tuning: the user can leave optimization decision to the auto-tuner [8], by using the question mark symbol. For example, if we use in line 7 of Listing 1 symbol `?` for the tile size in the second dimension (instead of the particular tile size value `20`), our auto-tuner will try to automatically identify an optimized tile size. Combining human expert knowledge with auto-tuning is important, because auto-tuning might make better performing decisions than humans for some optimizations (like choosing particular tile size values), and auto-tuning also enables using the same scheduling program for multiple devices (a.k.a *performance portability* [16]).

When replacing in Listing 1 tile size values, memory regions, and thread assignments by symbol `?`, to leave these decisions for the auto-tuner, we achieve after 3h tuning time a performance gain of $2.22\times$ over TVM's original optimization decisions in Listing 1 on NVIDIA A100 GPU (and of $2.67\times$ over NVIDIA's popular cuBLAS [17] library). Moreover, we achieve $1.79\times$ better performance than TVM on NVIDIA V100 GPU ($1.14\times$ over cuBLAS) with the same MDH scheduling program after a new auto-tuning run (performance portability); for TVM, we use its V100-optimized schedule program to make experimenting challenging for us. Similarly, when expressing TVM's OpenCL optimization decisions for Intel Broadwell E5-2683 CPU in our scheduling language, but making optimization decisions auto-tunable, we achieve a performance gain over TVM of $1.53\times$ (and of $1.36\times$ over Intel's oneMKL [18] library). For further deep learning computations, e.g., *convolutions*, we achieve similar, encouraging performance results, e.g., $1.29\times$ better performance than TVM+Ansor on Intel Broadwell E5-2683 CPU for the most time-intensive convolution computation within ResNet50's inference phase.[2]

V. Future Work

We plan to extend our scheduling language toward multi-device systems (which may consist of multiple GPUs and/or CPUs) and also multi-node systems (a.k.a. *MPI+X*), using the same hierarchical optimization approach as in Listing 1. Moreover, we aim to extend the set of C programs accepted as input by our approach (currently, limited to perfect loop nests with static loop bounds), based on polyhedral compilation techniques [10]. Also, we will significantly extend our experiments toward further deep learning computations, as well as computations different from deep learning [20]–[22].

[1] For $I \times K$ and $K \times J$ input matrices, the iteration space is a tuple (I, J, K).

[2] We provide an artifact for reproducing experimental results [19].

References

[1] T. Ben-Nun, J. de Fine Licht, A. N. Ziogas, T. Schneider, and T. Hoefler, "Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, 2019.

[2] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/chen

[3] R. Baghdadi, J. Ray, M. B. Romdhane, E. D. Sozzo, A. Akkas, Y. Zhang, P. Suriana, S. Kamil, and S. Amarasinghe, "Tiramisu: A polyhedral compiler for expressing fast and portable code," in *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2019, pp. 193–205.

[4] B. Hagedorn, A. S. Elliott, H. Barthels, R. Bodik, and V. Grover, "Fireiron: A data-movement-aware scheduling language for gpus," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 71–82. [Online]. Available: https://doi.org/10.1145/3410463.3414632

[5] B. Hagedorn, J. Lenfers, T. Kundefinedhler, X. Qin, S. Gorlatch, and M. Steuwer, "Achieving high-performance the functional way: A functional pearl on expressing high-performance optimizations as rewrite strategies," *Proc. ACM Program. Lang.*, vol. 4, no. ICFP, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3408974

[6] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 519–530. [Online]. Available: https://doi.org/10.1145/2491956.2462176

[7] A. Rasch, R. Schulze, and S. Gorlatch, "Generating portable high-performance code via multi-dimensional homomorphisms," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019, pp. 354–369.

[8] A. Rasch, R. Schulze, M. Steuwer, and S. Gorlatch, "Efficient auto-tuning of parallel programs with interdependent tuning parameters via auto-tuning framework (atf)," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 1, jan 2021. [Online]. Available: https://doi.org/10.1145/3427093

[9] S. Verdoolaege and T. Grosser, "Polyhedral extraction tool," in *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*, Paris, France, January 2012.

[10] A. Rasch, R. Schulze, and S. Gorlatch, "`md_poly`: A performance-portable polyhedral compiler based on multi-dimensional homomorphisms," in *Proceedings of the International Workshop on Polyhedral Compilation Techniques (IMPACT'20)*, 2020, pp. 1–4.

[11] A. Rasch, J. Bigge, M. Wrodarczyk, R. Schulze, and S. Gorlatch, "docal: high-level distributed programming with opencl and cuda," *The Journal of Supercomputing*, vol. 76, no. 7, pp. 5117–5138, 2020. [Online]. Available: https://doi.org/10.1007/s11227-019-02829-2

[12] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, J. E. Gonzalez, and I. Stoica, "Ansor: Generating high-performance tensor programs for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 863–879. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/zheng

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[15] A. Rasch and S. Gorlatch, "Multi-dimensional homomorphisms and their implementation in opencl," *International Journal of Parallel Programming*, vol. 46, no. 1, pp. 101–119, 2018. [Online]. Available: https://doi.org/10.1007/s10766-017-0508-z

[16] S. J. Pennycook, J. D. Sewall, and V. W. Lee, "A metric for performance portability," 2016. [Online]. Available: https://arxiv.org/abs/1611.07409

[17] NVIDIA, "cuBLAS," https://developer.nvidia.com/cublas, 2022.

[18] Intel, "oneMKL," https://github.com/oneapi-src/oneMKL, 2022.

[19] Artifact Implementation, "https://www.gitlab.com/mdh-project/hipar22_artifact," 2022.

[20] L.-N. Pouchet. (2015) PolyBench/C: the Polyhedral Benchmark Suite. [Online]. Available: http://web.cse.ohio-state.edu/ pouchet.2/software/polybench/

[21] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, p. 27, 2012.

[22] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.