Code Generation & Optimization for Deep-Learning Computations on GPUs via Multi-Dimensional Homomorphisms

Richard Schulze
University of Muenster
Germany
r.schulze@uni-muenster.de

Ari Rasch
University of Muenster
Germany
a.rasch@uni-muenster.de

Sergei Gorlatch
University of Muenster
Germany
gorlatch@uni-muenster.de

I. Introduction

Deep Learning (DL) is currently the most popular machine-learning method in the area of artificial intelligence. DL programs consist of multiple, data-parallel computations (like linear algebra routines and tensor operations), making DL programs time intensive.

We present our work-in-progress code generation and optimization approach for DL computations based on the algebraic formalism of *Multi-Dimensional Homomorphisms (MDH)* [1]. We show that popular DL computations can be expressed in the MDH formalism, thereby enabling exploiting the already existing MDH GPU code generation and optimization approach [2], [3] which has proven to achieve high performance and was so far not been focused on DL. Furthermore, we show that the MDH formalism is more expressive than the state-of-the-art DL abstractions (e.g., as provided by TensorFlow [4]): for example, MDH can express multiple DL computations (e.g., multiple element-wise computations) as a single MDH expression, which enables MDH optimizations (like tiling and parallelization) across the computations. Our experiments confirm that our MDH-based approach achieves better performance than the state of the art, including Apache TVM [5], [6] and Facebook's TC [7].

II. The MDH Formalism

We demonstrate the existing MDH formalism by expressing and discussing the example of matrix multiplication:

$$\begin{aligned}
&\texttt{MatMul}^{\texttt{<T∈Type | M,N,K∈ℕ>}} := \\
&\quad \texttt{out_view}^{\texttt{<T>}} \ (\ \texttt{C:(i,j,k)} \mapsto \texttt{(i,j)}\ ) \circ \\
&\quad \texttt{md_hom}^{\texttt{<M,N,K>}} \ (\ *,\ (+\!\!+_1, +\!\!+_2, +)\ ) \circ \\
&\quad \texttt{inp_view}^{\texttt{<T,T>}} (\ \texttt{A:(i,j,k)} \mapsto \texttt{(i,k)}\ , \\
&\qquad\qquad\qquad\quad \texttt{B:(i,j,k)} \mapsto \texttt{(k,j)}\ )
\end{aligned}$$

Here, we first fuse the domain-specific input of `MatMul` – two matrices $\texttt{A} \in \texttt{T}^{\texttt{M}\times\texttt{K}}$ and $\texttt{B} \in \texttt{T}^{\texttt{K}\times\texttt{N}}$ both of type `T` (e.g., `T=float` or `T=double`) – to a 3-dimensional array comprising pairs $\texttt{(A[i,k],B[k,j])} \in \texttt{T} \times \texttt{T}$. For this, we use pattern `inp_view` which the MDH formalism provides to uniformly prepare a domain-specific input for `md_hom`. After fusing `MatMul`s's two input matrices, we apply `MatMul`'s scalar function $*$ (multiplication) to each pair within the array, and we combine the obtained results in dimension 1 and 2 via $+\!\!+_1$ and $+\!\!+_2$ (concatenation), and in dimension 3 via $+$ (addition). Pattern `out_view` is trivial in this example, but could potentially be used to store the result matrix as transposed (by replacing "`(i,j,k)`$\mapsto$`(i,j)`" with "`(i,j,k)`$\mapsto$`(j,i)`") or in a stride fashion (replacing "`(i,j,k)`$\mapsto$`(i,j)`" with "`(i,j,k)`$\mapsto$`(i*s1,j*s2)`", for strides `s1`,`s2`$\in \mathbb{N}$), etc.

III. DL Computations expressed in the MDH Formalism

Table I shows how popular DL computations are expressed in the MDH formalism; the implementations are taken from the TensorFlow implementation of the real-world BERT neural network [8].

Fig. 1 shows for an illustrative example subgraph of BERT's computation graph that it can be expressed as a single MDH expression, thereby enabling MDH optimizations [2], [3] across the computation nodes (parallelization, tiling, etc).

IV. Experimental Results

We present our preliminary experimental results on NVIDIA V100 GPU for the example DL computations *BiasAdd Gradient (`BiasAddGrad`)* and *Batched Matrix Multiplication (`BatchMatMul`)* [4], as well as for the BERT's subgraph shown in Fig. 1. We confirm that our MDH-based approach [2], [3] achieves better performance than the newest versions of TensorFlow (TF) [4], TVM [5], [6], and TC [7] on real-world data sizes taken from the BERT [8] network:

- `BiasAddGrad`: MDH's speedup over TF is $> 1.5\times$, TVM is $> 2.9\times$, and TC is $> 1.7\times$;
- `BatchMatMul`: MDH's speedup over TVM is $> 1.1\times$, and TC is $> 1.9\times$ (TF has no GPU impl. for `BatchMatMul`);
- BERT's subgraph (Fig. 1): MDH's speedup over TF is $> 4.9\times$, TVM is $> 3.7\times$, and TC is $> 1.7\times$.

TABLE I
POPULAR DL COMPUTATIONS EXPRESSED IN THE MDH FORMALISM (SOME META-PARAMETERS OMITTED VIA ELLIPSIS FOR BREVITY).

Operator	out_view$^{<\ldots>}$	md_hom$^{<\ldots>}$	inp_view$^{<\ldots>}$
Mul$^{<\ldots>}$	OB1:(i,j) $\mapsto$ (i,j)	$*$, $(++_1, ++_2)$	IB1:(i,j) $\mapsto$ (i,j) , IB2:(i,j) $\mapsto$ (i,j)
Sub$^{<\ldots>}$	OB1:(i,j) $\mapsto$ (i,j)	$-$, $(++_1, ++_2)$	IB1:(i,j) $\mapsto$ (i,j) , IB2:(i,j) $\mapsto$ (i,j)
ExpandDims$^{<\text{axis},D\in\mathbb{N}\mid\ldots>}$	OB1:$(i_1,\ldots,i_D) \mapsto (\ldots,i_{\text{axis}-1},0,i_{\text{axis}},\ldots)$	id , $(++_1,\ldots ++_D)$	IB1:$(i_1,\ldots,i_D) \mapsto (i_1,\ldots,i_D)$
BiasAddGrad$^{<\text{NHWC}\mid\ldots>}$	OB1:(i,j) $\mapsto$ (j)	id , $(+, ++_2)$	IB1:(i,j) $\mapsto$ (i,j)
BatchMatMul$^{<N,N\mid\ldots>}$	OB1:(b1,b2,i,j,k) $\mapsto$ (b1,b2,i,j)	$*$, $(++_1,\ldots,++_4, +)$	IB1:(b1,b2,i,j,k) $\mapsto$ (b1,b2,i,k) , IB2:(b1,b2,i,j,k) $\mapsto$ (b1,b2,k,j)

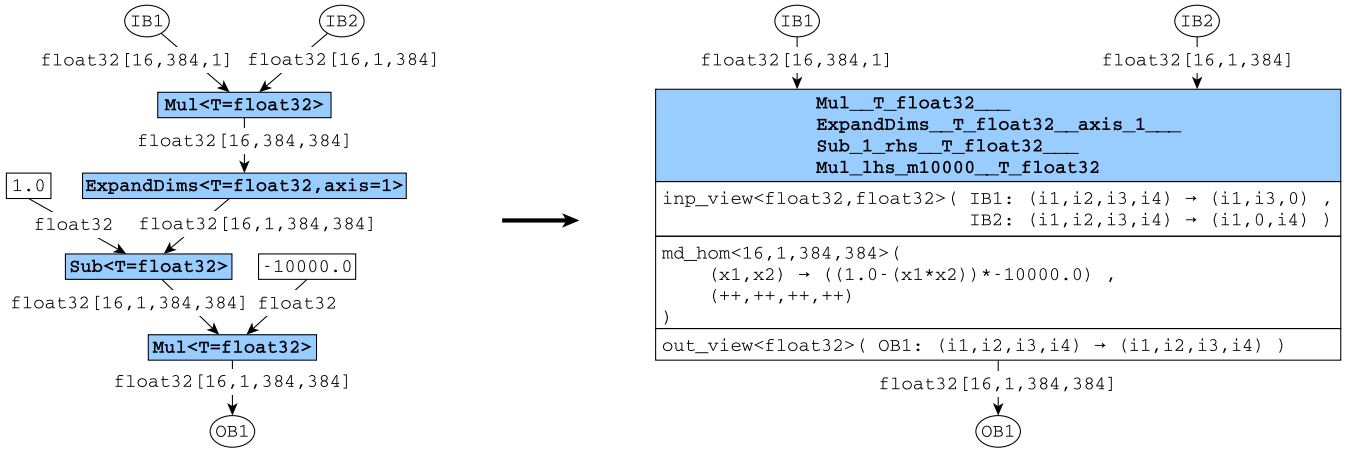

Fig. 1. Expressing an example subgraph of the BERT neural network (left) as a single MDH expression (right).

V. FUTURE WORK

We will significantly extend Table I, and we will describe in detail our methodology to generating a single MDH expression from DL subgraphs (here only briefly shown in Fig. 1 for a simple, illustrative example). Moreover, we will extend our experiments toward further architectures (multi-core CPUs, etc) and neural networks.

REFERENCES

[1] A. Rasch and S. Gorlatch, "Multi-dimensional homomorphisms and their implementation in opencl," *International Journal of Parallel Programming*, vol. 46, pp. pp. 101–119, 2018. [Online]. Available: https://doi.org/10.1007/s10766-017-0508-z

[2] 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.

[3] 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

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[5] 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

[6] 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

[7] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. Devito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "The next 700 accelerated layers: From mathematical expressions of network computation graphs to accelerated gpu kernels, automatically," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3355606

[8] Google Research. BERT GitHub Repository. [Online]. Available: https://github.com/google-research/bert