AUTOMATIC OPTIMIZED CODE GENERATION FOR GPU TENSOR OPERATIONS

Vedant Bhasin [*1] **Vibha Arramreddy** [*1]

ABSTRACT

Write an MLIR dialect for a small subset of core tensor operations starting with matrix-matrix multiplications to generate optimized GPU code without manually writing hand tuned kernels.

URL: https://vibhaarramreddy.github.io/

1 BACKGROUND

Rather than lowering code in a single step directly to LLVM IR, as is typical in traditional LLVM-based compilation. MLIR (Lattner et al., 2021) allows a program to be represented across multiple layers of intermediate abstractions. This is beneficial for domain-specific optimizations, with machine learning being a prominent example. MLIR uses dialects to allow optimizations at each level of abstraction.

2 THE CHALLENGE

Achieving performance comparable to hand-tuned GPU kernels is difficult due to several factors. Hand-tuned kernels often exploit low-level optimizations specific to GPU hardware, such as efficient memory access patterns, register tiling, warp synchronization, and shared memory usage. Replicating these optimizations in MLIR requires precise mapping of tensor operations to the GPU's architecture while minimizing overhead from abstractions. Furthermore, MLIR must balance general-purpose flexibility with hardware-specific tuning, particularly for operations like matrix multiplication where performance depends on data locality and minimizing memory bandwidth constraints.

3 RESOURCES

There are several interesting projects on MLIR available, some of them are summarized below. Most relevant to our preoject are the first two on the list. The MLIR documentation provides information on the different dialects available to perform optimization passes. The second is a project that was presented at one of MLIRs open meetings on High-Performance GPU Tensor Core Code Generation for Matmul which is essentially what we're trying to implement. In terms of hardware we plan to perform experiments on a NVIDA Tesla T4 GPU or on a NVIDIA GeForce RTX 2080.

1. MLIR documentation (Lattner et al., 2021)
2. High-Performance GPU Tensor Core Code Generation for Matmul Using MLIR.
3. HDNN: cross platform MLIR dialect (Martínez et al., 2022)
4. MLIR for ONNX (Jin et al., 2020)
5. Towards a high-performance AI compiler with upstream MLIR(Golin et al., 2024)

4 GOALS AND DELIVERABLES

4.1 MVP

We believe an MVP for this project would be MLIR generated CPU matmul code that has comparable performance with a reasonable optimized `C++` implementation.

4.2 Target

A good target deliverable for this project would be MLIR generated GPU matmul code that performs within 30% of the execution time of XLA libraries like CUBLAS.

4.3 Reach

A reach goal would be to target cross platform optimization i.e. optimizing for CPUs as well as GPUs.

[*]Equal contribution [1]Carnegie Mellon University, Pittsburgh, PA, USA. Correspondence to: Vedant Bhasin <vedantbhasin@cmu.edu>, Vibha Arramreddy <varramre@andrew.cmu.edu)>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2024.

5 SCHEDULE

5.1 November 13 - 17: Feasibility Study

MLIR is a relatively new compiler infrastructure with not a lot of support/ documentation available online. This timeframe will be used to test the feasibility of our project and whether or not we need to pivot to a different idea.

5.2 November 17 - 24: MVP, benchmarking, progress on target hardware

After attaining MVP, design an experimental study deciding on metrics of interest and the benchmarking process. Start implementing optimization passes for GPU.

5.3 November 24 - 27: Milestone report, slack

These three days are left as slack to ensure previous deliverables are met by this time, and we have at least MVP + preliminary results by milestone.

5.4 November 27 - December 1: Achieve Target

By December 1st we're planning on achieving our target goal of MLIR code gen for GPU.

5.5 December 1 - December 8: Experiments & Optimizations

Experiment with different dialects to maximize performance as much as possible.

5.6 December 8 - December 15: Benchmark + Report

Profile our implementation and baselines, finalize report and demo.

REFERENCES

Golin, R., Chelini, L., Siemieniuk, A., Madhu, K., Hasabnis, N., Pabst, H., Georganas, E., and Heinecke, A. Towards a high-performance ai compiler with upstream mlir, 2024. URL https://arxiv.org/abs/2404.15204.

Jin, T., Bercea, G.-T., Le, T. D., Chen, T., Su, G., Imai, H., Negishi, Y., Leu, A., O'Brien, K., Kawachiya, K., and Eichenberger, A. E. Compiling onnx neural network models using mlir, 2020. URL https://arxiv.org/abs/2008.08272.

Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., and Zinenko, O. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14. IEEE, 2021.

Martínez, P. A., Bernabé, G., and García, J. M. Hdnn: a cross-platform mlir dialect for deep neural networks. *The Journal of Supercomputing*, 78(11):13814–13830, 2022.