ELE 548: Project Proposal

Calvin Higgins

October 29, 2025

1 Abstract

Attaining optimal program performance on modern heterogeneous hardware requires specializing compiler optimizations to individual architectures. To ease the engineering burden, researchers have proposed automatically learning optimal orderings of compiler optimization phases. In this work, we minimize (1) binary size and (2) mean execution time by ordering LLVM optimization passes with EfficientZero, a sample-efficient reinforcement learning method. We attain an X% reduction in binary size compared to `-Oz` and an X% speedup compared to `-O3` when optimizing the MiBench benchmark suite.

2 Overview

TODO

3 Motivation

TODO

4 Key Insight

TODO

5 Design

TODO

6 Evaluation

metrics, methods.

6.1 Machine Learning for Code Optimization

CompilerGym [3] is a set of environments for compiler optimization tasks such as reducing code size and instruction count. To solve these tasks, agents select and apply pre-defined optimization passes to benchmark programs (i.e. agents try to solve the **phase-ordering problem**). Although several standard benchmark suites are supported, the **SPEC CPU2017 suite** [1] is notably absent. Moreover, CompilerGym only supports directly optimizing benchmark execution time on the **Jotai suite** [5].

To address these limitations, I propose the following:

1. Add the SPEC CPU2017 benchmark suite to CompilerGym, including code-size, instruction count and execution time observation spaces.
2. Implement baseline methods (e.g. random search) for optimizing SPEC CPU2017 benchmark runtimes.

6.1.1 Performance and World Models

In the absence of confounds such as operating system context switches, execution time is determined by three factors:

1. Generated machine code.
2. Microarchitecture implementation.
3. Input data distribution.

Finding near-optimal sequences of passes (a) with few executions requires accurate **performance models** to estimate execution time, and (b) with few compilations requires accurate **world models** to approximate the impact of optimizations on either code representation or execution time. To be accurate, performance and world models must incorporate all three factors influencing execution time.

However, existing methods for ordering optimization passes, such as **CompilerDream** [4] the only model-based RL method for phase-ordering, have primarily focused on code-size reduction. Some methods have addressed execution time, including [6] and [7], although they have focused on code, and not microarchitecture and data distribution representations.

To address this limitation, I propose the following:

1. Explicitly model the microarchitecture and data distribution, and inject these representations into an existing phase-ordering approach.

Alternatively, the combination of code, microarchitecture, and data representations could be evaluated via SPEC CPU2017 score prediction. Existing approaches are simple (e.g. multilayer perceptrons [8], convolutional neural networks [2]), and only explicitly incorporate microarchitecture information.

To address this limitation, I propose the following:

1. Incorporate code and data representations into SPEC CPU2017 score prediction.

References

[1] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. "SPEC CPU2017: Next-Generation Compute Benchmark". In: *International Conference on Performance Engineering (ICPE)*. 2018. DOI: 10.1145/3185768.3185771.

[2] Mehmet Cengiz et al. "Predicting the Performance of a Computing System with Deep Networks". In: *International Conference on Performance Engineering (ICPE)*. 2023. DOI: 10.1145/3578244.3583731.

[3] Chris Cummins et al. "CompilerGym: robust, performant compiler optimization environments for AI research". In: *International Symposium on Code Generation and Optimization (CGO)*. 2022. DOI: 10.1109/CGO53902.2022.9741258.

[4] Chaoyi Deng et al. "CompilerDream: Learning a Compiler World Model for General Code Optimization". In: *Conference on Knowledge Discovery and Data Mining (KDD)*. 2025. DOI: 10.1145/3711896.3736887.

[5] Cecília Conde Kind, Michael Canesche, and Fernando Magno Quintão Pereira. "Jotai: A methodology for the generation of executable C benchmarks". In: *Journal of Computer Languages* (2025). DOI: 10.1016/j.cola.2025.101368.

[6] Chunting Liu and Riyadh Baghdadi. "Data-Efficient Performance Modeling via Pre-training". In: *International Conference on Compiler Construction (CC)*. 2025. DOI: 10.1145/3708493.3712683.

[7] Tharindu R. Patabandi and Mary Hall. "Efficiently Learning Locality Optimizations by Decomposing Transformation Domains". In: *International Conference on Compiler Construction (CC)*. 2023. DOI: 10.1145/3578360.3580272.

[8] Ashkan Tousi and Mikel Luján. "Comparative Analysis of Machine Learning Models for Performance Prediction of the SPEC Benchmarks". In: *IEEE Access* (2022). DOI: 10.1109/ACCESS.2022.3142240.