Work-in-Progress: Searching Optimal Compiler Optimization Passes Sequence for Reducing Runtime Memory Profile using Ensemble Reinforcement Learning

Juneseo Chang
Seoul National University
Seoul, South Korea

Daejin Park
Kyungpook National University
Daegu, South Korea

ABSTRACT

The order in which compiler optimization passes are applied has a significant impact on program performance. However, widely used compiler optimization options use handpicked sets of optimization passes, optimized for specific benchmarks. In this paper, we propose an ensemble reinforcement learning (RL) model that optimizes LLVM transform passes sequence to reduce the runtime memory profile, which is an important consideration in resource-constrained embedded systems. We developed an LLVM intermediate representation (IR) analysis pass to extract static program features. The extracted features are processed with PCA for dimension reduction. We also generated datasets using a random program generator, and clustered them according to the PCA results of their extracted features. The ensemble RL model was trained on each clustered dataset. Experiments showed that the proposed model reduced 37% more memory profile than the standard optimization option.

CCS CONCEPTS

• **Computer systems organization** → **Embedded software**; • **Computing methodologies** → **Reinforcement learning**.

KEYWORDS

Code optimization, reinforcement learning, embedded system

ACM Reference Format:
Juneseo Chang and Daejin Park. 2023. Work-in-Progress: Searching Optimal Compiler Optimization Passes Sequence for Reducing Runtime Memory Profile using Ensemble Reinforcement Learning. In *International Conference on Embedded Software (EMSOFT '23), September 17–22, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3607890.3608460

1 INTRODUCTION

Compiler-level code optimization is a crucial but difficult task. These optimizations are applied as transform passes sequence, while the optimization performance depends on which transform passes are applied in which order. To facilitate such optimizations, compiler engineers handpicked sets of transform passes, regardless of the target program. Therefore, a performance gap exists between the ideally-optimized and compiler-optimized programs. To narrow this, recent studies use machine learning to identify the optimal transform passes sequence for an arbitrary input program [1].

Among various optimization purposes, we focus on peak runtime memory profile reduction, which is an important consideration in embedded systems. To the best of our knowledge, there is no memory profile optimization by transform passes at this point, since the transform passes that can reduce memory usage are too sparse compared to the entire search space.

In this paper, we present an ensemble RL model that optimizes the transform passes sequence to reduce the runtime memory profile for an input program. We developed an LLVM IR analysis pass for extracting program features and used PCA to reduce dimensions. We utilized a dataset derived from a random program generator and clustered based on the PCA results. The ensemble RL model was trained on each clustered dataset. All source codes and datasets are made public for future studies[1].

2 PROPOSED MODEL

2.1 Feature Extraction and Dataset

Feature Extraction. To represent program characteristics, program feature $\boldsymbol{f}$ should be extracted. We developed an LLVM IR analysis pass that extracts 50 features, including the number of various types of instructions and basic blocks (Fig. 1-A).

Dataset. Benchmark suites used in the literature are too simple to reduce memory profile by applying a transform passes sequence. This implies that generating complex, diverse datasets is crucial. Therefore, we utilized Csmith [2] to generate diverse valid random C codes S, with code sizes greater than 300kb (Fig. 1-A). The generated code was divided into 150 training datasets and 67 test datasets.

Transform Pass Candidates. To select appropriate transform pass candidates for the RL model action, we examined 50 LLVM transform passes' impact on the runtime memory profile of our dataset. 32 transform passes T that reduced memory profile compared to the `-O0` were selected. Full lists of features, datasets, and selected transform passes are available in our code repository.

2.2 Dimension Reduction and Clustering

PCA. Although the extracted feature $\boldsymbol{f}$ represents essential characteristics of a program, its dimension is too large to be used for the RL model. Therefore, we applied PCA, which is a widely used feature reduction technique, that also helps model training when there are only a small number of diverse training samples (Fig. 1-B). In our setup, $\boldsymbol{f} \in \mathbb{N}^{50}$ was abstracted to $PCA(\boldsymbol{f}) = \boldsymbol{p} \in \mathbb{R}^{15}$, where the 15 PCA components accounted for 99% of the variances.

[1] https://github.com/jschang0215/RL-TransformPass-Optimization

EMSOFT '23, September 17–22, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0291-4/23/09.
https://doi.org/10.1145/3607890.3608460

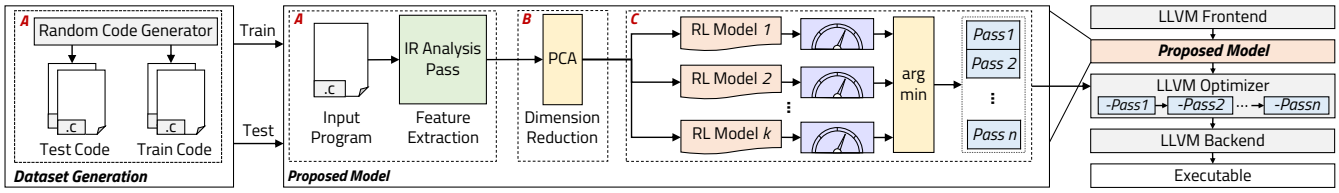

Figure 1: The proposed model extracts the program feature, applies PCA, and feeds to the ensemble RL model.

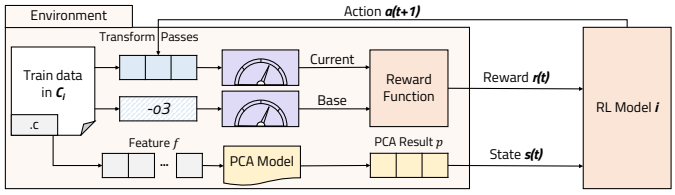

Figure 2: RL environment and training process.

Clustering. Building a model that can cope with a diverse dataset is challenging. Accordingly, we clustered the training dataset into k clusters, $C_1, C_2, \ldots, C_k$, according to the PCA results of the training dataset with the KMeans clustering. The number of clusters was determined as $k = 4$ using the Elbow method.

2.3 Ensemble RL Model

RL Environment. The goal of the RL model is to output the transform passes sequence $\boldsymbol{o}$ that best reduces memory profile, based on the PCA result $\boldsymbol{p}$. We defined the action space of the RL model as $A = \{a \mid a \in T^n\}$; that is, the action a is a $n = 12$-dimensional vector with each entry representing a transform pass in order. The PCA result $\boldsymbol{p}$ is considered as the state s in the observation space. The reward is defined as $r = (m_s - m_o)/m_0$, where m_s and m_0 represent peak memory profile when applying the current action and LLVM `-O3`, respectively (Fig. 2). For learning algorithms, we used PPO, A3C, and PG algorithms, which are best suited for training discrete actions. We implemented the training process using RLlib, a scalable and open-source RL library.

Ensemble RL. Prior works have employed a single RL model; however, capturing the behavior of various applications with a one-size-fits-all model is challenging and inefficient. Consequently, we speculated that the ensemble approach would be effective, in which each RL model R_i is trained with clustered dataset C_i ($1 \leq i \leq k$). When the model is deployed, the PCA result $\boldsymbol{p}$ of the target program's extracted feature is used as the RL model's input. Among the outputs $\boldsymbol{o}_i \in T^n$ of each RL model R_i, the output $\boldsymbol{o}$ that best minimizes the memory profile was chosen (Fig. 1-C). When multiple outputs equally minimize the memory profile, the output $\boldsymbol{o}$ that minimizes execution time was selected.

3 EXPERIMENTAL RESULTS

We implemented ensemble models that take advantage of clustered datasets, as well as non-ensemble models trained with each learning algorithm. We evaluated by comparing each model on 67 test datasets from Section 2.1 with the LLVM `-Os` option, since `-Os` best reduced the memory profile among all standard options. The

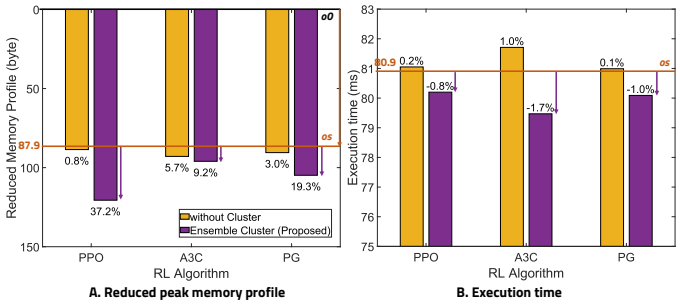

Figure 3: Memory profile, Execution time compared to `-Os`.

ensemble model that exploits clustered dataset trained with PPO reduced 37% memory profile than the `-Os` option (Fig. 3-A), while also reducing the execution time (Fig. 3-B).

4 CONCLUSION AND OUTLOOK

In this study, we developed an ensemble RL model that exploits clustered datasets to find the transform passes sequence that best reduces peak runtime memory profiles. Our model reduced 37% more memory profile compared to the LLVM `-Os` option. We expect the proposed model to enable embedded system developers to easily improve the program's memory profile, by simply running the RL model at the compilation stage without any "human-in-the-loop" processes. In our follow-up study, we plan to in-depth analyze the RL model behavior, and evaluate our method on real-world, memory-consuming embedded software.

ACKNOWLEDGMENTS

This study was supported by the BK21 FOUR project (4199990113966), the Basic Science Research Program (NRF-2018R1A6A1A03025109, 20%), (NRF-2022R1I1A3069260, 20%) funded by the Ministry of Education, and (2020M3H2A1078119) by Ministry of Science and ICT. This work was partly supported by IITP grant funded by MSIT (No. 2021-0-00944, 30%), (No. 2022-0-01170, 20%), (No. RS-2023-00228970, 30%), and (No. RS-2022-00156389, 10%).

REFERENCES

[1] Ameer Haj-Ali, Qijing (Jenny) Huang, and et al. 2020. AutoPhase: Juggling HLS Phase Orderings in Random Forests with Deep Reinforcement Learning. In *MLSys*, Vol. 2. 70–81.

[2] Xuejun Yang, Yang Chen, and et al. 2011. Finding and Understanding Bugs in C Compilers. In *PLDI* (San Jose, California, USA). ACM, New York, NY, USA, 283–294.