Ordering Optimization Passes with Reinforcement Learning for Instruction Count Reduction

Calvin Higgins
Department of Computer Science and Statistics
University of Rhode Island
Kingston, United States
calvin_higgins2@uri.edu

***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 passes. In this work, we minimize IR instruction count by ordering LLVM optimization passes with deep Q-learning, a standard reinforcement learning method. We attain an TODO% reduction in instruction count compared to `-O0` (% regression from `-Oz`) across six representative benchmark suites.**

***Index Terms*—TODO**

I. Introduction

The goal of a compiler is simple: find an optimal translation of high-level code into machine code. Many modern compilers rely on the LLVM compiler infrastructure which provides a unified intermediate representation (think architecture-independent assembly language) as well as associated optimization and code-generation tools [**?**]. These compilers typically begin with a relatively naive translation from high-level code into LLVM intermediate representation (IR), and iteratively refine it with transformations known as optimization passes. Different optimization pass orders can yield binary with drastically different sizes and runtime performance.

The phase-ordering problem involves determining the best order to apply a fixed set of optimization passes. Historically, expert compiler engineers have designed new optimization pass orders for every architecture. However, as hardware rapidly evolves and becomes increasingly heterogeneous, maintaining optimization pass orders imposes a growing engineering burden. Automation can alleviate this burden, allowing compiler experts to focus on more critical tasks [**?**]. Unfortunately, as LLVM exposes hundreds of distinct passes, and typical orders apply hundreds of passes, automatically finding good orders is extremely challenging. Deep reinforcement learning has successfully explored other difficult search spaces, such as video games, demonstrating promise for compiler optimization [**?**].

By formulating the phase-ordering problem as a Markov decision process, standard reinforcement learning techniques apply. A Markov decision process is a four tuple (S, A, T, R) where S is a set of states, A is a set of actions, $T(s, a)$ is a transition function mapping from a state s and an action a to a new state, and $R(s, a)$ is the change in state cost upon applying action a to state s. In the phase ordering problem, the set of states S is an equivalence class of programs, the action space A is the set of all optimization passes, the transition function $T(s, a)$ outputs the program s after applying optimization pass a, and the reward function $R(s, a) = C(s) - C(T(s, a))$ where $C(s)$ is a cost function such as binary size, instruction count, or execution time. Unfortunately, not only does the large action space A lead to a combinatorial explosion of possible optimization sequences, but rewards are also sparse as effective sequences are rare, and evaluating the transition and reward functions is slow as they require compiler invocations.

Accordingly, any approach to the phase-ordering problem ought to be sample efficient. CompilerDream [**?**], a state-of-the-art phase-ordering policy, adapted DreamerV2 [**?**], a general purpose reinforcement learning method to compiler optimization. However, although the Dreamer methods dominate nearly all reinforcement learning methods across a diverse set of domains, they are notably overshadowed by EfficientZero [**?**] in low sample regimes [**?**]. Motivated by its sample efficiency, we aim to apply EfficientZero to the phase-ordering problem. In this work, we conduct preliminary experiments with deep Q-learning [**?**], hoping to gain experience and insights before implementing EfficientZero.