Machine Learning of Polyhedral Compilation

Prakshal Nandi

Master of Science

School of Informatics

University of Edinburgh

2022

Abstract

A significant amount of time is spent executing loops for most compute-bound applications. By leveraging the polyhedral model, PolyGym[**?**] has presented the scheduling of a particular class of loops having regular structure as a Markov Decision Process. Given the wide range of the programs presented to the compiler, learning profitable schedules is a complex task. Deep Reinforcement Learning methods such as Deep Q-Learning have recently successfully solved complex problems. Building on top of PolyGym, the proposed research advances the emerging field of Machine Learning for Compiler Optimizations by implementing and evaluating established Deep Reinforcement Learning algorithms on PolyBench data sets [**?**].

Research Ethics Approval

This research does not involve any human participants or personal data. All data are collected based on simulation experiments on benchmark suites; hence no ethical concerns are associated with the project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Prakshal Nandi*)

Acknowledgements

Any acknowledgements go here.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Reinforcement Learning(RL) has recently been seen as a promising approach for compiler optimization [**?**][**?**]. A Reinforcement Learning agent makes the *observations* within the *environment*, and takes *actions* to receive certain *rewards*. The goal is to maximize the rewards over a period of time. On the other side, to achieve optimizations related to data locality, memory management, communication, and automatic parallelization, the space of the possible transformations available to a compiler is vast. Applying Machine Learning(ML) algorithms to find a good strategy in this space is challenging. Polyhedral Compilation is one type of compiler optimization technique applicable to a particular class of nested loops. It can explore geometric dependency properties of kernel statements in different iterations and apply transformations to improve the schedule of the loop execution. In compiler optimization, machine learning is yet to achieve the level of success in other fields such as Computer Vision or Natural Language Processing. Even though it has been an object of research for more than two decades, machine learning guided techniques are still not widely utilized in industrial compilers.

This project proposal is structured as follows: Following the discussion regarding the problem statement, research objective, novelty, and significance of the project in Section 1, Section 2 provides an overview of the background and literature review. Section 3 focuses on project methodology, limitations, risks, and ethical considerations. Section 4 explains how the evaluation of the developed methods will be carried out. Section 5 discusses expected outcomes, and Section 6 outlines the research timeline, milestones, and deliverables.

1.2 Objectives

The aim of the project is to analyze how Reinforcement Learning can help improve Polyhedral Compilation. This analysis will be performed with the help of Deep Neural Networks and other hybrid meta-heuristics. The research hypothesis is that Reinforcement Learning algorithms can be generalized to learn Polyhedral Compilation.

It is not always clear how to represent the actual problem in a Reinforcement Learning environment. However, this part has already been covered in Polygym[**?**]; hence it is not going to be the focus of this research. The main objectives for this project are trying to find answers to these questions:

- Can we implement a Deep Reinforcement Learning algorithm that can help select the best actions related to polyhedral transformations?
- Can we design reward functions that improve the training performance to learn profitable schedules?
- Can we explore different RL algorithms and find the most suitable technique by evaluating their performance?

1.3 Structure

Chapter 2

Background

2.1 Polyhedral Compilation

The iteration space of a nested loop can be represented as an n-dimensional geometric model called a polytope. The optimization through Polyhedral Compilation is achieved in three different stages, moving from code to polyhedron model, applying program transformations within the model, and finally regenerating the code from the model. This model gives the framework to transform the loops, where the array references and loop bounds are affine functions of loop iterators and parameters. The polyhedral model is broadly applicable due to the high presence of affine loops in compute-bound applications[**?**]. In the kernel example shown in Figure 1, geometric interpretations such as two-dimensional polygons can be used to visualize dependencies between statements. Based on these dependencies, conclusions can be made regarding the preferred execution time of each statement.

2.2 Deep Reinforcement Learning

The concept of Reinforcement Learning is one of the oldest in the field of Machine Learning, but it found huge success when DeepMind researchers used it to play Atari games outperforming humans [**?**]. In March 2016, DeepMind created AlphaGo, which later on beat the world champion in the game of Go [**?**]. They achieved it by applying methods of Deep Learning on Reinforcement Learning, specifically using techniques such as *Deep Q-Networks* (DQNs) and *Policy Gradients*. The value of an action a can be explained as the total reward received by executing it in a state s. In Q-Learning, the agent uses these state-action values to make a decision. However, this approach

might struggle when the number of states is vast. This challenge can be overcome by approximating Q-Values of required state-action pairs, using a range of parameters. Deep Neural Networks (DNN) have proven very effective for this purpose. Using such Deep Q-Networks(DQN) for approximating the Q-Learning is called Deep Q-Learning.

2.3 PolyGym

PolyGym provides an environment of Polyhedral optimizations for Reinforcement Learning, adhering to OpenAI Gym interface [**?**], which is a toolkit for solving RL problems. Suppose the actions are chosen to directly represent the transformations, such as loop fusion or tiling. In that case, the model may learn the profitable schedules only for a particular set of loops or domains. Instead, PolyGym takes a more granular approach[1]. For constructing the space of valid schedules, it uses the existing formulation of Farkas' lemma from discrete geometry. In the next step, a Markov decision process(MDP) can explore this space to find a profitable schedule, independent of the loop being optimized. Hence, this makes it a two-stage technique, the construction of schedule space and its exploration, using two different Markov Decision Processes with different states and actions. This framework, however, still lacks a training agent, although just by using this simple heuristic, PolyGym is able to find transformations that lead to a speed up of 3.39x over LLVM Or and 1.34x over best isl transformation [**?**]. As shown in Figure 2, it should be noted that PolyGym does not use any agent, but an agent can use the environment defined by PolyGym for learning.

2.4 Literature Review

Previous surveys have explored the relationship between Machine Learning and Compiler Optimizations [**?**][**?**]. Several studies have also begun to examine how RL can be applied to the field of compiler optimizations. In 2014, Emani *et al.* demonstrated a new approach for dynamically mapping parallel programs to varying system resources [**?**]. They used Markov Decision Process for online thread scheduling and an offline model trained based on the system environment and the structure of the code. Similar to PolyGym, CompilerGym extends the Gym interface to provide a reinforcement learning environment to researchers for compiler optimizations [**?**]. A recent study by Mammadli *et al.* presented an excellent example of using Deep Reinforcement Learning

[1] https://github.com/PolyGym/polygym

for finding effective sequences of phase-ordering, which is a long-standing problem for compiler generators [**?**]. However, finding profitable loop transformations is a much more fine-drawn problem than finding the optimum number of threads or distributing the work on different cores. In an investigation related to current Single Instruction, Multiple Data (SIMD) architectures, Neurovectorizer [**?**] addresses the challenge to find a good vectorization strategy with the help of Deep Reinforcement Learning. MLGO [**?**] attempted to leverage Reinforcement Learning and Evolutionary Algorithms for making decisions about function in-lining for improving code size. However, the evaluation for the code size could be less challenging than the evaluation of the heuristics focusing on the program's execution speed due to the uncertainty involved in the time taken during each execution.

All these methods, however, did not consider polyhedral optimizations. In 1993, Feautrier presented the theoretical basis of the polyhedral formulation for single and multidimensional schedules [**?**][**?**]. After more than twenty years, Pouchet *et al.* proposed an iterative optimization method for systematic exploration of the schedule search space [**?**][**?**]. Building on top of these methods, Ganser *et al.* demonstrated improvements in optimization with the use of genetic algorithms in 2017 [**?**]. One year later, they showed a reduction in benchmarking efforts by using surrogate performance models, where they trained the models based on the program transformation evaluated in the previous iterative optimizations, at the cost of 15% degradation in speed-up [**?**]. This set of methods followed the iterative approach with benchmarking, making them time-consuming. In contrast, methods such as isl[**?**] and Pluto[**?**] generated the models directly to find a good compilation strategy in less time. However, this approach has certain limitations. The optimization performed is still dependent on the type of loop being analyzed, and its performance can not beat the time taking iterative methods. Hence, the proposed research with Reinforcement Learning fits very well between these two approaches.

Chapter 3

Methodology

Chapter 4

Evaluation

Chapter 5

Results and Discussions

Chapter 6

Conclusions

6.1 Unresolved Issues

6.2 Future Work

Bibliography

Appendix A

First appendix

A.1 First section