Using Machine Learning to Predict the Sequences of Optimization Passes

Laith H. Alhasnawy, Esraa H. Alwan, and Ahmed B. M. Fanfakh(✉)

Department of Computer Science, College of Science for Women, University of Babylon, Hillah, Iraq
lythhamd0718@gmail.com,
{wsci.israa.hadi,wsci.ahmed.badri}@uobabylon.edu.iq

Abstract. The manual tuning for the sequence of optimization passes in modern compilers was impractical, where this sequence was not general to all benchmark programs in achieving optimal performance. Therefore, the process of selecting a set of passes manually them over a particular program to achieve optimal performance is a very difficult problem. Moreover, choosing the order for these passes will add another problem called phase order problem.

In this paper, the proposed approach provides auto tuning optimization seqaences instead of manual tuning by building a prediction scheme. The proposed framework used machine learning to find a sequence for each program by collecting these passes based on the features of program. K-Nearest Neighbors classifier algorithm (KNN) is used in the prediction process and improved by that the reduction algorithm that work after it. The reduction algorithm eliminated passes that have a negative impact on program execution time.

The proposed approach was evaluated using the LLVM (Low Level Virtual Machine) compiler under Linux Ubuntu. The obtained results showed that this approach outperforms standard optimization level-O2 of LLVM compiler in improving the execution time byan average of 21% through building a prediction scheme by using KNN algorithm. Consequently, the execution time is improved byan average of 23% due to the application of the reduction algorithm on the sequences of optimization passes resulting from KNN algorithm.

Keywords: LLVM · KNN algorithm · Reduction algorithm · Optimization seqaence · Optimization passes

1 Introduction

Modern compilers are portioned into three parts: the front end, middle end and back end. The frontend creates an intermediate representation (IR) for the source program after testing its syntactic and semantic rightness. The middle end which represents the optimizer is a critical part. It applies a sequence of conversions on the IR in order to optimize it to get better execution time, code size, and power consumption. The backend converts the optimized IR to target assembly code. The optimizer part arranged as a sequence of optimization passes that represents analysis passes followed by transformation passes.

A. M. Al-Bakry et al. (Eds.): NTICT 2020, CCIS 1183, pp. 139–156, 2020.
https://doi.org/10.1007/978-3-030-55340-1_10