Finding a Good Global Sequence using Multi-Level Genetic Algorithm

Zeyd S. Alkaaby[1] zeydsaeed@gmail.com

Esraa H. Alwan[2] isr.phd@gmail.com

Ahmed B. M. Fanfakh[3] afanfakh@gmail.com

[123] Department of Computer science
Collage of science for women
University of Babylon

Abstract

Trying all the optimization sequences manually to find out a one that give the best performance is not practical solution. Therefore, it is essential to layout a schema which is able to introduce an optimization sequence with better performance for a given function.

In this work, multi-levels genetic algorithm has been used to find a good optimal sequence. Our method has three levels. In the first level, the programs search space is divided into three groups and try to find a good sequence for each program in group. These good sequences for each program will be used as initial seed to find good sequence for all programs in that group. This process will be repeated for all three groups to find good sequence for each one. Then, these good sequences from three groups will be used as a seed for initial population to the Third level. Genetic algorithm will use the resulting sequences to find out one good optimal sequence for all these groups. LLVM compiler framework has been used to validate the proposed method. Experiments that have been implemented on the generated good sequence for different benchmarks show the effectiveness of the proposed method. Overall, it achieves better performance compare with the -O2 flag.

I. Introduction

Modern compilers introduce a massive number of optimization passes targeting different code segments of an application. These optimization passes can transform the code segment which might be a basic block, a function or the whole program to optimize one (Purini,2013). The optimization can be applied through the whole life of the program, in another word, at different compilation stages (Almagor,2004). Although the purpose of optimization is producing better code (speed or code size), however there is no grantee that the resulting code is doing better than the original one. Typical optimizer is made up of a set of analysis passes followed by transformation passes. The analysis passes responsible for collecting information about the