Whole-Program Linear-Constant Analysis with Applications to Link-Time Optimization

Ludo Van Put
lvanput@elis.ugent.be

Dominique Chanet
dchanet@elis.ugent.be

Koen De Bosschere
kdb@elis.ugent.be

Department of Electronics
and Information Systems
Ghent University

Abstract

Current link-time optimization techniques can reduce the power consumption and code size of embedded software [2]. Due to a lack of information, the stack frames of procedures are left untouched by link-time program optimizers. In this paper we present a practical whole-program linear-constant analysis [9] that allows to analyze the stack layout of a procedure. The analysis deals with the peculiarities of link-time program representation, namely the lack of high-level information and the huge size of the control flow graph. Even on a complete linux kernel, our analysis is practical in terms of computation time. The collected information consists of restricted affine equations between two registers, but it enables optimizations complementary to existing link-time optimization techniques. On a set of ARM benchmarks, the number of store operations decreases by up to 7% while the execution time, program size and power consumption are all further improved. This paper discusses both the practical issues of applying whole-program linear-constant propagation as well as its use in program optimization and understanding.

1 Introduction

An important application of static analysis is the detection of affine relations that hold between the variables of a program. The information provided by such an analysis is used in optimizing compilers and program verification. A lot of work in this field deals with the theory of the analyses and the classification of the abstracted programs they work on (see [6] for an overview). This paper discusses the implementation and evaluation of interprocedural linear-constant propagation applied on realistic programs.

Link-time transformation of programs has been successfully used for program compaction and program optimization [2, 3]. The technique benefits from the whole-program overview at link time to apply optimizations that are complementary to compiler optimizations. The drawback of link-time transformation is the absence of high-level information on the program, forcing analyses and optimizations to be very conservative. Due to the size of whole-program control flow graphs, all analyses and optimizations must be carefully engineered to be practical in their use.

At link time, the information on variables as they exist in the source code or at compile time is discarded. Therefore, the analysis presented in this paper will detect simple affine relations that hold between the processor registers at every point in the program. Each of the affine relations contains at most two registers, which is a specialization of the relations that are used in the work by Karr [5]. The binary relations that we evaluate are similar to those in the linear-constant propagation by Sagiv et al.[9], but they are less general. Since our analysis operates on the level of machine instructions, even simple relations can capture enough information to enable effective program optimizations and at the same time keep the running time and memory usage practical.

In the next section, we discuss the characteristics of the program representation at link time. In Section 3, we explain our data flow analysis and the different operations involved. In Section 4 we give some applications that use the computed information and we discuss the results of the optimizations. We discuss related work in Section 5 and draw conclusions in Section 6.

2 Link-time program representation

To motivate the design of our analysis, explained in the next section, we highlight the most important characteristics of the program representation that we are working on at link time. We have implemented our analysis using Diablo, a framework for link-time binary rewriting[1]. For a

[1] http://www.elis.ugent.be/diablo