On the Side-Effects of Code Abstraction

Bjorn De Sutter
brdsutte@elis.ugent.be

Hans Vandierendonck
hvdieren@elis.ugent.be

Bruno De Bus
bdebus@elis.ugent.be

Koen De Bosschere
kdb@elis.ugent.be

Electronics and Information Systems (ELIS) Department
Ghent University, Sint-Pietersnieuwstraat 41
9000 Gent, Belgium

ABSTRACT

More and more devices contain computers with limited amounts of memory. As a result, code compaction techniques are gaining popularity, especially when they also improve performance and power consumption, or at least not degrade it. This paper quantifies the side-effects of code abstraction on performance using extensive measurements and simulations on the SPECint2000 benchmark suite and some additional C++ programs. We show how to use profile information in order to obtain almost all the code size reduction benefits of code abstraction, yet experience almost none of its disadvantages.

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors—*code generation;compilers;optimization*; E.4 [**Coding and Information Theory**]: Data Compaction and Compression—*program representation*

General Terms

Experimentation, Performance

Keywords

performance, code abstraction, code compaction

1. INTRODUCTION

More and more devices contain computers with limited amounts of memory. Examples are PDAs, set top boxes, wearables, mobile and embedded systems in general. The limitations on memory size result from considerations such as space, weight, power consumption and production cost. As a result, the last decade has witnessed a growing research into the automated generation of smaller programs using compaction and compression techniques.

As the field of program compaction and compression matures, side-effects of these techniques are more and more taken into account. The goals for code compaction are gradually shifting. Whereas reducing program size was for a long time the main objective, the focus has now changed to reducing program size while maintaining or even improving performance and lowering power consumption.

Code abstraction is a technique by which a program fragment that occurs multiple times in a program is abstracted into a separate procedure. The original occurrences of the fragment are replaced by a call to the abstracted procedure. Code abstraction techniques have proven very efficient in reducing code sizes of programs in general, and of C++ programs in particular, leading to code size reductions of up to 35% and more [9].

Unfortunately, code abstraction also introduces a significant amount of run-time overhead: more instructions will be executed, in particular more procedure calls and returns. This influences performance-related design criteria such as instruction counts and instruction cache hit rates. As a result, embedded system developers in practice often discard code abstraction because design criteria like performance and power consumption are even more important than code size.

In this paper we present the first extensive empirical study of the side-effect of code abstraction. With this study, we refute the performance-related arguments for not applying code abstraction. We discuss and extensively quantify the side-effects of code abstraction on execution speed, instruction cache behavior, code schedule quality and branch prediction. Previously we suggested to avoid performance degradation by using profile information while still obtaining significant code size reductions [9]. The main contribution of this paper is the rigid validation of that suggestion.

LCTES'03, June 11–13, 2003, San Diego, California, USA.

2. CODE ABSTRACTION

Code abstraction is the replacement of a multiple occurring code fragment by a single copy. The latter forms the body of a new procedure, and each original occurrence of the fragment is replaced by a call to that procedure. This technique can be seen as the inverse of inlining.