LANCET: A Nifty Code Editing Tool

Ludo Van Put Bjorn De Sutter Matias Madou Bruno De Bus

Dominique Chanet Kristof Smits Koen De Bosschere

Ghent University, Electronics and Information Systems Department
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
{lvanput,brdsutte,mmadou,bdebus,dchanet,ksmits,kdb}@elis.ugent.be

ABSTRACT

This paper presents LANCET, a multi-platform software visualization tool that enables the inspection of programs at the binary code level. Implemented on top of the link-time rewriting framework DIABLO, LANCET provides several views on the interprocedural control flow graph of a program. These views can be used to navigate through the program, to edit the program in a efficient manner, and to interact with the existing whole-program analyses and optimizations that are implemented in DIABLO or existing applications of DIABLO. As such, LANCET is an ideal tool to examine compiler-generated code, to assist the development of new compiler optimizations, or to optimize assembly code manually.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—*graphical environments*; D.3.4 [**Programming Languages**]: Processors—*optimization*

General Terms

Design, Experimentation

Keywords

visualization, optimization, instrumentation, assembler, binary code

1. INTRODUCTION

During the last decades, an extensive range of (graphical) software engineering tools has been developed. These tools most often operate on high level, abstract program representations, such as graphical program blocks (e.g., Labview), screen widgets (Glade), or object-oriented languages (Eclipse, Visual Studio .NET, ...). But despite the ability to work with abstract program representations, there still exist numerous scenarios in which both researchers and developers need to examine assembler code.

A first scenario that comes to mind is the development of compiler optimizations. First, existing code has to be scanned for inefficiencies in order to discover interesting opportunities for new compiler optimizations. Once such an opportunity is found, a compiler researcher might try to apply an the appropriate optimization manually to evaluate its influence on execution speed or power consumption. Obviously this requires the ability to examine and edit the assembler code. If successful, the researcher may start implementing the necessary analyses and the automated transformation in his compiler. At this stage, he needs to verify that the resulting changes to the compiled programs are as intended. Again, assembler code needs to be examined.

Another important scenario is that of an embedded programmer. These days, embedded programs are most often written in higher-level programming languages, of which executable code is generated automatically by compilers. It is not unusual however, that performance critical kernels of an application, such as loops that need to handle streaming media in real time, are optimized manually because the compiler cannot exploit architectural peculiarities. Or that certain (partially) automated optimizations are triggered manually, for example if a developer decides that one hot loop needs to be unrolled while another hot loop, for whatever reason, need not be unrolled.

Finally, sometimes a program does not perform as expected. In that case, a developer needs to be able to study the generated code to get insights in the bottlenecks. Maybe an optimization was unexpectedly not applied by the compiler, or maybe some combination of statements resulted in a particularly bad instruction schedule. In this scenario, it is important that the developer can navigate through the program easily, that he can find the hot spots efficiently, and that he can extract information about those hot spots, such as the results of data flow analysis. The latter can learn the developer why some optimization was not applied.

To the best of our knowledge, there exist no tools today to support the tasks described above on a complete program. To fill that gap, this paper presents LANCET (Lancet is A Nifty Code Editing Tool), a GUI on top of the link-time program editing framework DIABLO. In short, LANCET provides an interface to navigate through a graph representation of a binary program, edit the program, and interact with a wide range of analyses and transformations.

The remainder of this paper is structured as follows. Sec-

PASTE '05 Lisbon, Portugal