The Design and Implementation of FIT: a Flexible Instrumentation Toolkit

De Bus, Bruno
bdebus@elis.ugent.be

Chanet, Dominique
dchanet@elis.ugent.be

De Sutter, Bjorn
brdsutte@elis.ugent.be

Van Put, Ludo
lvanput@elis.ugent.be

De Bosschere, Koen
kdb@elis.ugent.be

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

ABSTRACT

This paper presents FIT, a Flexible open-source binary code Instrumentation Toolkit. Unlike existing tools, FIT is truly portable, with existing backends for the Alpha, x86 and ARM architectures and the Tru64Unix, Linux and ARM Firmware execution environments. This paper focuses on some of the problems that needed to be addressed for providing this degree of portability. It also discusses the trade-off between instrumentation precision and low overhead.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Measurement techniques*; D.2.5 [**Software Engineering**]: Testing and Debugging—*tracing;diagnostics*; D.3.4 [**Programming Languages**]: Processors—*code generation;compilers*

General Terms

Experimentation, Performance

Keywords

performance, code abstraction, code compaction

1. INTRODUCTION

Compiler and computer architecture research depends on the analysis of run-time program information, and during the last decades many tools for collecting run-time information have been developed: emulators [1], static instrumentation tools [8, 9], run-time instrumentation tools [3, 5] and hardware monitoring tools [11]. This paper presents FIT, a Flexible Instrumentation Toolkit for static instrumentation.

While in theory a program can be instrumented at any point in a compiler tool chain, in practice it is simplest and most useful to add instrumentation code in a post-compilation executable code rewriting step. Most importantly, this ensures that the instrumentation does not influence the compiler code generation. Moreover, it enables the instrumentation of precompiled library code.

Several link-time or post-link-time executable code rewriting systems have been developed in the past [7], but they all fail with respect to at least one usability criterium. Our new binary code instrumentation toolkit FIT combines the following usability properties:

- *Portability* FIT at the moment has back-ends for the Alpha, x86 and ARM architectures, and for the Tru64-Unix, Linux and ARM Firmware Suite execution environments. Support for the IA64 and MIPS32 architectures is under construction.
- *Extensibility* FIT's user interface is implemented as a collection of wrappers on top of the open-source executable code editing framework Diablo. To extend FIT, it suffices to add more wrappers around the existing analyses and transformations.
- *Precision* FIT can guarantee that all addresses occurring in the original program remain unchanged.
- *Tunability* When full precision is not needed, FIT can disable it to minimize the instrumentation overhead.

In the remainder of this paper, Section 2 first discusses the shortcomings of existing related work. Section 3 then presents FIT, focusing on how FIT overcomes the previously mentioned shortcomings. In Section 4, the performances of different precisions of instrumentation are compared, and conclusions are drawn in Section 5.

2. RELATED WORK

Before presenting FIT in more detail in the Section 3, this section briefly discusses existing work in the field of binary code instrumentation and the shortcomings thereof. Given the very large number of existing run-time information collection tools, we limit ourselves to one example for each of three classes of instrumentors: Diota [3] for dynamic instrumentation, Simics [4] for emulation and ATOM [9] for static