Fast JIT Code Generation for x86-64 with LLVM

Viktor Pavlu, Andreas Krall[1,2]

Institute of Computer Languages, Vienna University of Technology, Argentinierstrasse 8/E185, 1040 Wien, Austria

ABSTRACT

Our work focuses on investigating novel ways of efficient processor simulation using just-in-time compilation techniques. We can automatically generate a cycle-accurate simulator from a processor description that captures hardware structure and instruction set. The simulator employs an adaptive two-level just-in-time compilation scheme based on LLVM to attain high simulation speeds.

As the main source of slowdown during simulation we identified the LLVM code generator. We reduced compilation time in our own experimental code generator by an order of magnitude compared to LLVM's original backend. Current work aims at leveraging instruction descriptions already available in LLVM to extend the coverage of our fast JIT code generator.

KEYWORDS: code generation; instruction selection; just-in-time compilation; simulation

1 Introduction

Embedded systems have become a prevalent part of our everyday life and it is very unlikely that this trend is going to decline anytime soon. To meet the increasing performance requirements of modern embedded systems, application specific instruction-set processors (ASIPs) often provide a viable trade-off between computing power and opposing metrics such as power consumption or chip area. With today's short production cycles and processors custom-built for a single application the need for simulators is further amplified. At the same time, application specific instruction-sets demand a high level of flexibility from the simulators; for each new ASIP a corresponding simulator is required.

Previously our group had developed a processor description language that captures the instruction set and hardware structure of a processor. Using those processor descriptions, various design tasks can be automated, e. g., we can automatically customize a compiler backend, generate hardware models and we can generate a cycle-accurate simulator for the modeled processor. This effectively allows to shorten development turnaround times.

[1]Authors' e-mail addresses: {vpavlu, andi}@complang.tuwien.ac.at

[2]The work described here is supported by the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) under contract P21842, *Optimal Code Generation for Explicitly Parallel Processors*, `http://www.complang.tuwien.ac.at/epicopt/`.

Currently, the simulators use a mixed approach based on two levels of dynamic compilation via the LLVM just-in-time (JIT) compiler. In the first level, the code is compiled with all optimizations turned off. The second level identifies hot traces and recompiles them with optimizations enabled. Most time is spent in first-level compilation. Our work concentrates on reducing that compile-time.

2 LLVM JIT Compiler

Even with optimizations turned off, LLVM executes almost 1.5 million instructions in the JIT compiler for a minimal basic block that consists of a single return instruction. We have implemented an experimental x86-64 code emitter that generates code directly from LLVM intermediate representation (IR) to investigate the potential of bypassing the existing backend that is not optimized for JIT compiling. By using our experimental emitter we can reduce the time spent in the JIT roughly by an order of magnitude for that trivial basic block. We expect this speedup to grow with larger basic blocks as the regular backend aims at quality code while the experimental codegen emits machine code equivalents of LLVM IR directly.

The main drawback of this experimental code emitter is that the IR to machine code mapping is hand-coded instead of generated from instruction descriptions readily available in LLVM. The coverage of IR therefore remains limited if one does not want to duplicate the existing x86-64 backend by hand. The nature of the LLVM instruction descriptions, however, prevents us from directly using them in the fast code emitter. While the descriptions themselves are collected in a single place, the interpretation of the instruction descriptions – the process that yields the machine code we want to emit – is distributed over the whole backend.

Our ongoing research aims at leveraging the existing instruction descriptions to extend the coverage of the fast JIT backend for LLVM. We do this by extracting parametrizable chunks of machine code for each LLVM IR instruction from the existing backend. The chunks are collected at compile time. At runtime – when the fast JIT backend compiles the simulation code – the code emitter only has to iterate over IR instructions in a single pass assembling the already prepared machine code chunks. A second pass then resolves relocations.

Instruction selection is strongly reduced in complexity in this fast JIT backend. Register allocation and scheduling are left out completely as our primary concern is to reduce time spent in the JIT compiler. The quality of the code being generated is of little concern as regions executed more often will eventually be recompiled with the second-level JIT that produces optimized code.

3 Project status

The EPICOpt project started in October 2009 and is scheduled to last until 2012. The work described above is still in progress but first experimental results confirm the viability of our approach. By drastically reducing the time spent compiling mostly cold regions of code, we expect to gain a substantial increase in overall simulation speed. This will greatly add to the attractiveness of cycle-accurate simulators generated from processor descriptions.