Fast JIT Code Generation

Tilmann Scheller
Overview

- Introduction
- tiny-llvm-codegen
- SkyEye
- Performance Numbers
- Summary
Introduction

- Traditional LLVM JIT has a relatively high overhead since it's essentially using the same code generator like the static compiler.
- Only useful for really hot code.
- Fast-isel solves part of the problem but overhead still significant.
- It would be nice to just flip a switch and get a different tradeoff in terms of compile time/runtime performance.
tiny-llvm-codegen

- Work is based on tiny-llvm-codegen
- tiny-llvm-codegen is a really simple JIT for LLVM IR targeting x86-32
- Developed by Mark Seaborn in March 2013
- Ported tiny-llvm-codegen to x86-64
- Added basic support for the AMD64 System V ABI
Extremely simple translator
Very small (about 2000 LOC)
No register allocation
No instruction selection
No instruction scheduling
Just translating every LLVM IR instruction one by one
All values go into memory
Example

```c
define i64 @foo(i64 %a, i64 %b) {
  %1 = add i64 %b, %a
  ret i64 %1
}
```

```assembly
define i64 @foo(i64 %a, i64 %b) {
  %1 = add i64 %b, %a
  ret i64 %1
}
```
- Supported: Integer operations
- Missing: Floating-point operations, Vector operations
- No performance tuning yet
- Probably lots of low hanging fruit
- Supports i1, i8, i16, i32, i64
SkyEye

- Open Source full system simulator
- Supports a wide range of different architectures: ARM, PowerPC, MIPS, x86, SPARC, ColdFire, Blackfin
- Does interpretation as well as dynamic binary translation with LLVM (using a fork of the libcpu project)
- Can run an ARM Android 2.2 build
SkyEye Overview

ARM

Interpreter

Translate to LLVM IR

Optimize LLVM IR

tiny-llvm-codegen

LLVM JIT

x86-64

x86-64
Test workload

- Simulating a Samsung S3C6410X SoC with an ARM11 core
- Booting an ARMv6 Linux 3.0 kernel
- This requires about 150 million instructions
- Produces 33MB of optimized bitcode
Test workload

- Compiling the 33MB of bitcode offline:
  - 3.3 seconds with tiny-llvm-codegen
  - 67 seconds with llc
- JITing every basic block which is executed at least twice to compare the performance of both JITs
- Booting the kernel on the simulated system: about 3x faster when using tiny-llvm-codegen (24 sec vs. 76 sec)
- Measured on an Intel Core i7-4770K
Summary

- Ported tiny-llvm-codegen to x86-64
- Successfully compiles a substantial amount of LLVM IR
- Performance numbers look promising
- Future:
  - Support the remaining LLVM IR instructions
  - Performance tuning
  - Add support for another architecture
  - Add a simple register allocator?
Thank you.
References

- http://github.com/mseaborn/tiny-llvm-codegen
- http://skyeye.sourceforge.net
- http://libcpu.org