An LLVM back-end for HiPE, the native code compiler of Erlang/OTP

Christos Stavrakakis, Yiannis Tsiouris

November, 2011

Abstract

Existing open-source compilers are based on old code generation technology, with code bases that are difficult to learn and hard to change, and share no code between each other. The Low Level Virtual Machine (LLVM) is a state-of-the-art compiler infrastructure providing a set of reusable components that implement the best known techniques focusing on compile time and performance of the generated code. The goal of LLVM is to provide modular components for building high quality compilers for many different languages.

This thesis describes the architecture, design decisions and implementation details of a new back end for HiPE, the native code compiler of Erlang/OTP, that targets the LLVM infrastructure. One of HiPE's intermediate representation, called Register Transfer Language (RTL), was found to have a very straightforward translation to LLVM assembly. However, there were a few subtle points, such as the calling convention, the exception handling mechanism and garbage collection, that needed to be handled in order to retain Application Binary Interface (ABI) compatibility with the Erlang Run-Time System (ERTS) and integrate our work in the existing Virtual Machine architecture. For these reasons we patched the LLVM Code Generator and imposed rules on the generated binary code.

In the evaluation we detail the current complexity and performance of the new LLVM back end for the AMD64 architecture. The run-time performance was found to be comparable with HiPE and signifficantly faster than BEAM virtual machine and Erjang, a virtual machine for Erlang based on the Java Virtual Machine (JVM). The complexity of the LLVM back end proved to be far simpler; especially, if you take into consideration that, with rather plain extensions, it can support all hardware architectures that HiPE currently targets. Various performance improvements are planned for future work.

Keywords

Erlang, HiPE compiler, native code compilation, LLVM framework, back end, high-level assembly, intermediate representation transformation, compile-time optimizations