What is ErLLVM?

• A project aiming to provide multiple back-ends for the High Performance Erlang (HiPE) native code compiler of Erlang/OTP ...

• … using the Low Level Virtual Machine (LLVM) compiler infrastructure …

• … in order to improve the performance of Erlang applications …

• … and ease the maintenance of its native code compiler.
This talk

• Overview and design
  – HiPE native code compiler
  – LLVM compiler infrastructure

• Architecture and implementation of ErLLVM
  – LLVM extensions
  – New HiPE component

• Performance evaluation
  – vs. BEAM
  – vs. HiPE

• Current status and future work
HiPE: High Performance Erlang

- Project at Uppsala University started in 1997
- Developed the native code compiler for Erlang
  - Integrated into Erlang/OTP since 2001
- A mature compiler that is robust and produces reasonably efficient code
- Back ends for
  - SPARC V8+
  - x86 and x86_64 (AMD64)
  - PowerPC and PowerPC64
  - ARM
LLVM: Low Level Virtual Machine

• Collection of industrial strength compiler technology
  • Language-independent optimizer and code generator
    • Many optimizations, many targets, generates good code
  • Clang C/C++/Objective-C front end
    • Designed for speed, reusability, compatibility with GCC
  • Debuggers, “binutils”, standard libraries
    • Providing pieces of low-level tools, with many advantages

• High-level portable LLVM assembly
  • RISC-like instruction set; static type system; SSA form
  • Three forms: human-readable, on-disk, in-memory
Why LLVM?

• Used as a static or JiT compiler and for static analysis
• State-of-the-art software with very active community of developers
• A new compiler = glue code + any components not yet available. Allows choice of the right components for the job, e.g. register allocator, scheduler, optimization order.
• Supports many architectures: x86, x86_64, ARM, PowerPC, SPARC, Alpha, MIPS, Blackfin, CellSPU, Mblaze, MSP430, XCore, …
• Open source with a BSD-like License and many contributors: industry, research groups, individuals.
HiPE Architecture in Erlang/OTP

Erlang Run-Time System

BEAM Emulator

Memory
- BEAM Bytecode
- Other Data
- Native Code

BEAM Disassembler

HiPE Loader

HiPE Compiler

Symbolic BEAM
- Icode
- RTL

SPARC

X86

... PowerPC

Kostis Sagonas

ErLLVM: http://erllvm.softlab.ntua.gr
Structure of HiPE's Back-ends

- Register allocation
  - Many choices; default iterated register coalescing
- Frame management
  - Check for stack overflow
  - Set up frame
  - Create stack descriptors
  - Add “special” code for tailcalls
- Code linearization
- Assembly
Why use LLVM as a Back-end?

• Curiosity: perform a research experiment
• Easier maintenance of existing back-ends
  – One instead of six
  – Small-sized, straightforward code
  – Outsource implementation and further optimization
• Get more back-ends “for free” (well, almost…)
• Improve performance
  – Outsource target-related optimizations
HiPE Architecture in ErLLVM

• Use existing HiPE Loader and ERTS support
  – Be ABI compatible!

Kostis Sagonas
ErLLVM: http://erllvm.softlab.ntua.gr
The LLVM Component

- **RTL**
  - LLVM back end
  - LLVM assembler
  - LLVM Assembly
  - LLVM back end
  - LLVM assembler
  - LLVM Bitcode
  - LLVM optimizer
  - LLVM Bitcode
  - LLVM compiler
  - Native Assembly
  - LLVM-GCC assembler
  - Object Code
  - Object file parser
  - Binary Code

---

- **hipe_rtl2llvm** Create human-readable LLVM assembly (.ll)
- **llvm-as** Human-readable assembly (.ll) → LLVM bitcode (.bc)
- **opt** Optimization Passes, supports standard groups (-O1, -O2, -O3) (.bc → .bc)
- **llc** Bitcode (.bc) → Native assembly (.s), impose rules about memory model, stack alignment, etc.
- **llvm-gcc** Create object file (.s → .o)
- **elf64_format** Extract executable code and relocations

Kostis Sagonas

ErLLVM: http://erllvm.softlab.ntua.gr
Subtle Points of Using LLVM

- Calling convention
  - VM “special” registers, args and return values
  - callee-/caller-save registers, callee pops args

- Explicit frame management
  - In-lined code for stack overflow checks in assembly prologue

- Stack descriptors
  - Exception handling
  - Precise garbage collection
Current Status of ErLLVM

• Patches to LLVM
  – Custom calling convention & register pinning
  – GC plugin to write GC information in object file
    • Use elf_format to parse .o file and extract the info
  – Function pass to emit custom prologue

• New HiPE component on top of R15B “maint”
  – Support for x86 and x86_64
  – Support for accurate GC: mark stack slots not live when variables that “inhabit” them are no longer live
  – About 5000 LOC

• Very robust and ready to use in production!
ErLLVM's Performance vs. BEAM

BEAM / ErLLVM (Small)

Runtime Speedup

Benchmark

Kostis Sagonas

ErLLVM: http://erllvm.softlab.ntua.gr
ErLLVM's Performance vs. BEAM

BEAM / ErLLVM (Shootout)

Benchmark

Runtime Speedup

Kostis Sagonas

ErLLVM: http://erllvm.softlab.ntua.gr
ErLLVM's Performance vs. BEAM

BEAM / ErLLVM (Misc.)

Runtime Speedup

Benchmark

Kostis Sagonas
ErLLVM's Performance vs. HiPE

HiPE / ErLLVM (Small)

Runtime Speedup

Benchmark

Kostis Sagonas

ErLLVM: http://erllvm.softlab.ntua.gr
ErLLVM's Performance vs. HiPE

HiPE / ErLLVM (Shootout)

Runtime Speedup

Benchmark
ErLLVM's Performance vs. HiPE

The graph compares the runtime speedup of HiPE and ErLLVM across various benchmarks. The benchmarks include `bin_to_term_bm`, `bs_bm`, `bs_simple_bm`, `bs_sum_bm`, `call_bm`, `call_tail_bm`, `float_bm`, `freq_bm`, `fun_bm`, and `Average`. The graph shows the performance metrics for two platforms: `amd64` and `x86`.
Now what?

Demo time!
Current Status: Pros

- **Complete & robust**: handles all Erlang programs
- **ABI compatible**: 
  - smooth integration with BEAM and HiPE code
- **Performance**: 
  - much better than BEAM 
  - almost as good as HiPE
- Smaller and simpler code base for the back-ends
- Possibility to target more architectures
- LLVM back-end improvements now also improve performance of Erlang applications!
Current Status: Cons & Future Work

• Cons:
  – Need to download and install custom LLVM
  – Slightly longer compilation times
    • Erlang LLVM bindings to the rescue??

• Future Work
  – Push LLVM patches upstream
  – ARM
  – Improve GC support
Where can I find ErLLVM?

ErLLVM is a project aiming at providing multiple back ends for the High Performance Erlang (HiPE) with the use of the Low Level Virtual Machine (LLVM) infrastructure.

Currently, ErLLVM supports the AMD64 and x86 architectures.
Users are now welcome!

- Install following the instructions at:
  
  http://erllvm.softlab.ntua.gr

- Grab code from github
- Test and measure!
- Report experiences
- Contribute to the project