HiPE
High Performance Erlang

A brief overview of the compiler

Open Source Erlang (Erlang/OTP)

- Part of Ericsson's Open Telecom Platform (OTP).
- Implemented and commercially supported by Ericsson, but the source code is free and available on-line (`www.erlang.org`).
- Till recently (Oct. 2001), Erlang/OTP was exclusively a byte-code interpreted system based on a virtual machine:
  - JAM (stack-based) - not supported anymore;
  - BEAM (register-based) – current VM.

HiPE: High Performance Erlang Compiler

- HiPE is a native code compiler on top of BEAM, written in Erlang.
- HiPE is fully and tightly integrated within Open Source Erlang/OTP (starting with Release 8B)
- Compiler for the complete Erlang language
- Back-ends for:
  - SPARC V8+
  - x86-based machines running Linux or Solaris
  - AMD-64
  - PowerPC (32-bits)

HiPE Compiler: Design Goals

A "just-in-time" native code compiler for Erlang

- Allows *flexible*, *user-controlled* compilation of Erlang programs to native machine code
- *Fine-grained:* Compilation unit is a single function.

Desiderata:

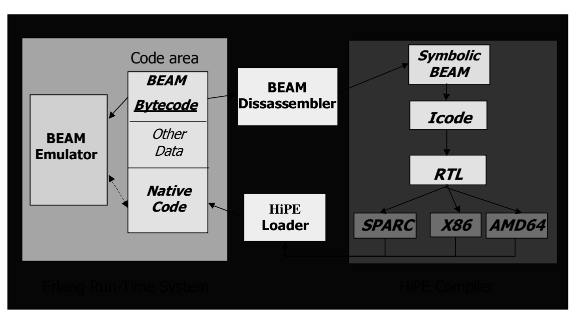
- Reasonable compilation times
- Acceptable sizes of object code

Alternatives to Bytecode Interpretation

- Compile to another "similar" language with a more mature implementation (e.g., Scheme).
- Compile to a sufficiently low-level and fast language such as C.
- Use C-- as a portable assembly language.
- Use a retargetable code generator as ML-RISC.
- Compile to the gcc back-end.
- Compile directly to native code.

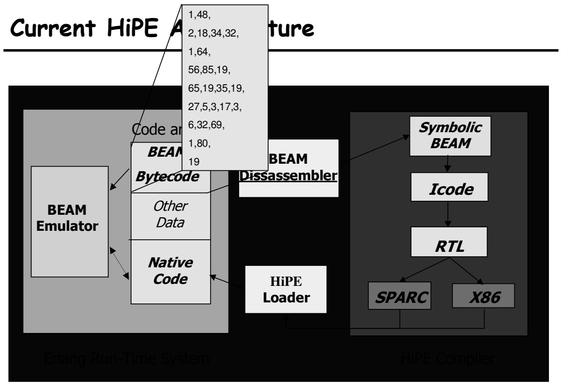
One can roughly expect a decrease in portability and increase in performance and implementation effort for choices lower in the list.

Current HiPE Architecture



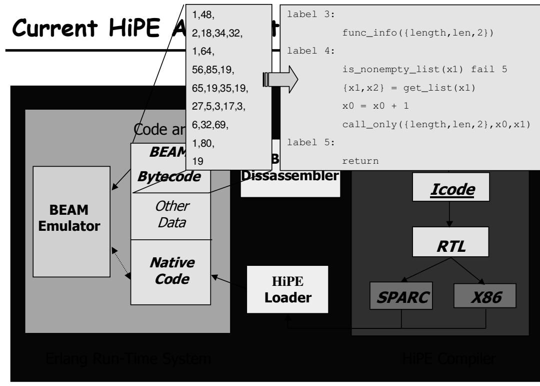
A HiPE-enabled Erlang/OTP system

Current HiPE Architecture



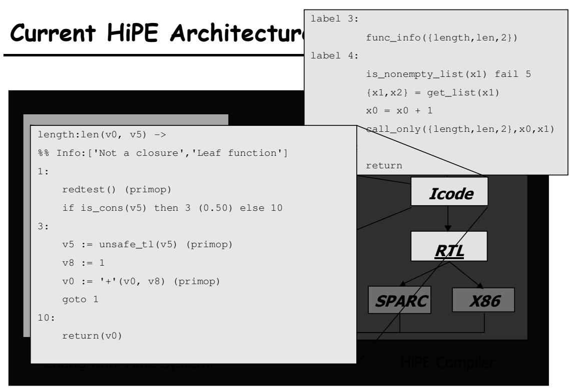
A HiPE-enabled Erlang/OTP system

Current HiPE Architecture



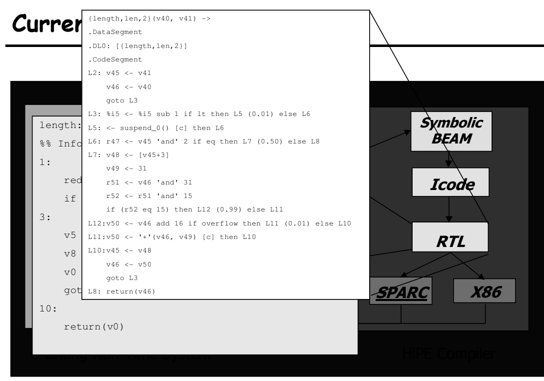
A HiPE-enabled Erlang/OTP system

Current HiPE Architecture

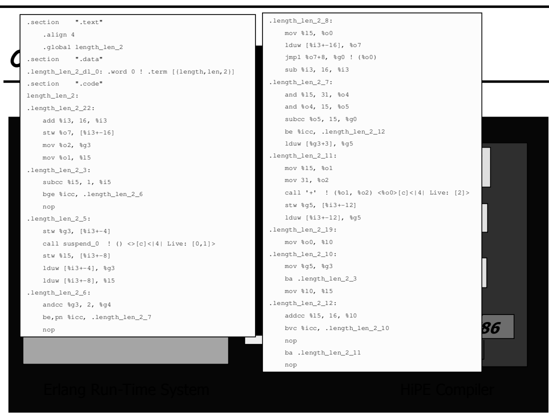


A HiPE-enabled Erlang/OTP system

Current HiPE Architecture



A HiPE-enabled Erlang/OTP system



A HiPE-enabled Erlang/OTP system

Intermediate Representations in HiPE

Icode

- Idealized Erlang assembly language;
- Stack is implicit; unlimited number of temporaries which survive function calls;
- Most of memory management is explicit;
- Process scheduling is implicit.

RTL (Register Transfer Language)

- Generic 3-address target-independent language;
- Tagging is made explicit: RTL has both tagged and untagged registers;
- Data accesses and initializations are turned into loads and stores.

HiPE: Technical Details

- HiPE exists as a new component (currently about 90,000 lines of Erlang code and 15,000 lines of C and assembly code) added to an otherwise mostly unchanged Open-Source Erlang/OTP system.
- HiPE provides its user with a set of profiling tools to identify the hot-code parts of the applications.

HiPE: Runtime System Issues

- Both virtual machine code and native code can happily co-exist in the runtime system
  - To simplify the garbage collector, we use separate stacks for native and interpreted execution
- HiPE optimizes calls to functions which execute in the same mode (no overhead)
- Preserves tail-calls (required feature of Erlang)

The HiPE Runtime System

<u>Machine-specific parts</u>

1. Code for mode-switch interface (in assembly)
2. Glue code for calling C BIFs from native code (in assembly)
3. Code to traverse the stack for GC (in C)
4. Code to create native code stubs & to apply patches to native code during loading (in C)

The HiPE Linker

- When a function f is compiled to native code
  - The bytecode for f is patched so that future calls to f are redirected to its native code
  - If f contains calls to a function g that is not (yet) compiled to native code, a native code-stub for the callee (g) is created to redirect the call to the emulator.
- When a module is reloaded or recompiled, all calls from native code to that module are patched to call the new module
  (in accordance to the hot-code loading semantics)

Optimizations Performed by the HiPE Compiler

- Adaptive pattern matching compilation of construction and matching against binaries.
- Copy & sparse conditional constant propagation, constant folding (partly make up for the absence of types) on Icode and RTL.
- Dead & unreachable code removal on Icode and RTL.
- Partial redundancy elimination on RTL.
- Merging of heap-overflow checks through backward propagation.

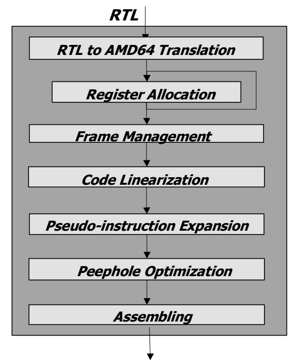
HiPE Compiler: SPARC back-end

- Parameter-passing in registers (up to 16)
- Register allocation based on choice between a Briggs-style graph coloring, iterated register coalescing, or a linear scan algorithm [SPE'03] which is the default.
- Cache-conscious code linearization.
- Garbage collection:
  - Based on *two-generational copying*
  - Aided by *stack descriptors* (live-variable maps)
  - Performs *generational stack collection.*

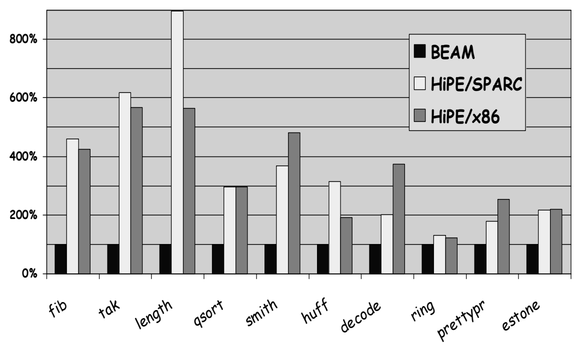
HiPE Compiler: x86 and AMD-64 backends

- Use the native stack of the machine
  - Use %esp as the current process' stack pointer
- Pay attention to register usage
  - Preferred register allocator:
    iterated register coalescing
- Stack-frame minimization
  - Spill-slot coalescing
- Pay attention to branch prediction
  - Use call and ret instructions consistently.

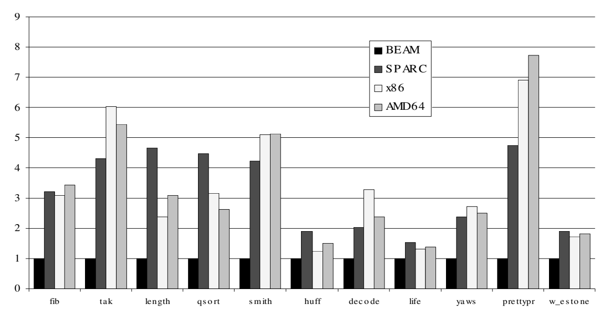
Backend Passes



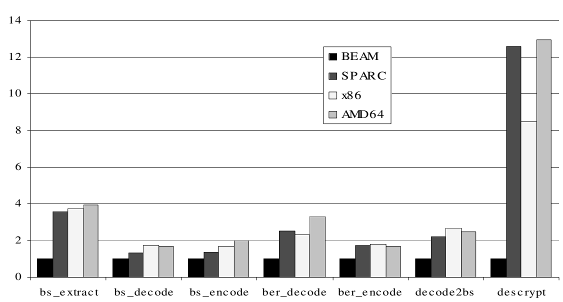
Performance of HiPE on SPARC & x86 (Feb 2002)



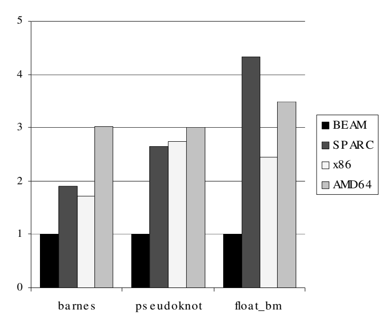
A more up-to-date Performance Comparison



Performance: Speedups (Programs w Binaries)



Performance: Speedups (Programs w Floats)



Space Performance (very rough)

HiPE generates native code that is roughly about 2.5 to 3 times bigger than BEAM bytecode