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 October 2001, Erlang/OTP was exclusively a byte-code interpreter for 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+ (or higher) running Solaris 8, 9 or 10
  - x86 based machines running Linux, FreeBSD or Solaris
  - x86_64 based machines running Linux or FreeBSD
  - PowerPC (32 and 64-bits) machines running MacOS X or Linux
  - ARM

HiPE Compiler: Design Goals

A native code compiler for Erlang

- Allows *flexible*, *user-controlled* compilation of Erlang programs to native machine code
- *Fine-grained:* Compilation unit was (till R15B) just a single function. Nowadays, it's a whole module.

Desiderata:

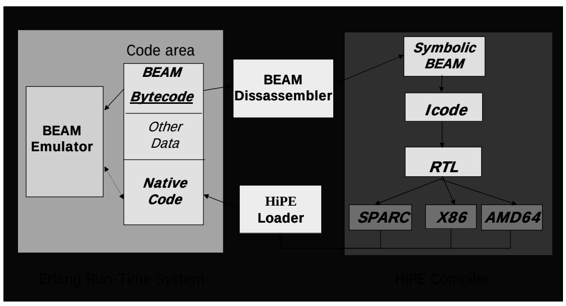
- Reasonable compilation times
- Acceptable sizes of object code

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

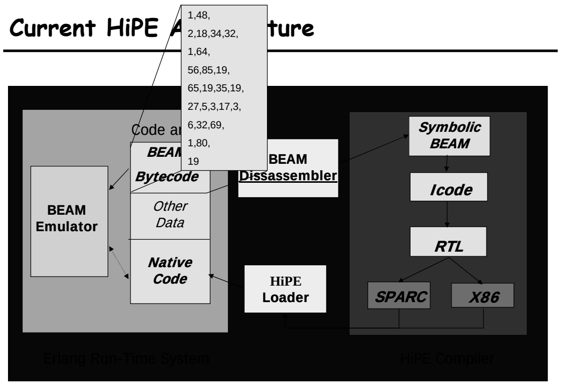
One can roughly expect a decrease in portability and increase in performance and implementation effort for choices lower in the list.

Current HiPE Architecture



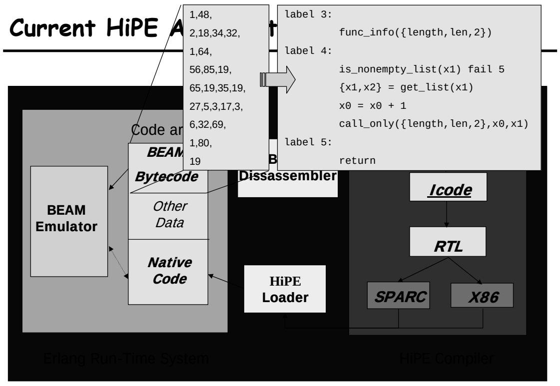
A HiPE-enabled Erlang/OTP system

Current HiPE Architecture



A HiPE-enabled Erlang/OTP system

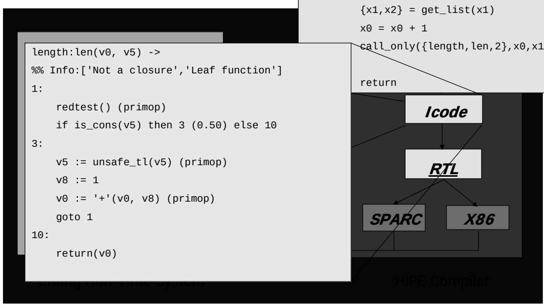
Current HiPE Architecture




```
label 3:
        func_info({length,len,2})
label 4:
        is_nonempty_list(x1) fail 5
        {x1,x2} = get_list(x1)
        x0 = x0 + 1
        call_only({length,len,2},x0,x1)
label 5:
        return
```

A HiPE-enabled Erlang/OTP system

Current HiPE Architecture



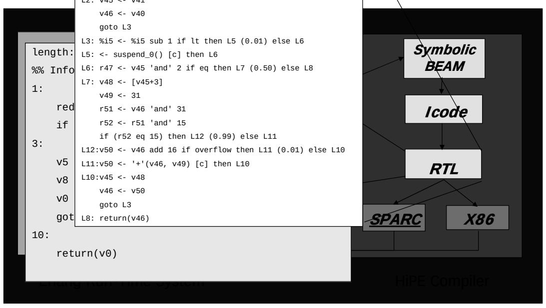

```
label 3:
        func_info({length,len,2})
label 4:
        is_nonempty_list(x1) fail 5
        {x1,x2} = get_list(x1)
        x0 = x0 + 1
        call_only({length,len,2},x0,x1)

        return
```

```
length:len(v0, v5) ->
%% Info:['Not a closure','Leaf function']
1:
    redtest() (primop)
    if is_cons(v5) then 3 (0.50) else 10
3:
    v5 := unsafe_tl(v5) (primop)
    v8 := 1
    v0 := '+'(v0, v8) (primop)
    goto 1
10:
    return(v0)
```

A HiPE-enabled Erlang/OTP system

Curren

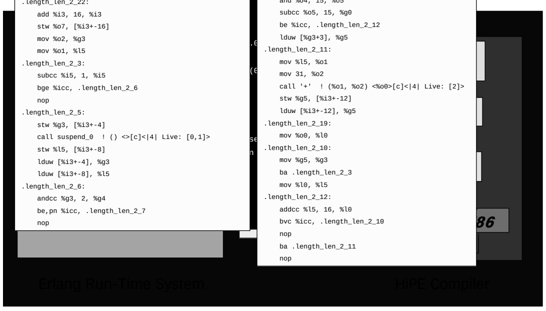



```
{length,len,2}(v40, v41) ->
.DataSegment
.DL0: [{length,len,2}]
.CodeSegment
L2: v45 <- v41
    v46 <- v40
    goto L3
L3: %i5 <- %i5 sub 1 if lt then L5 (0.01) else L6
L5: <- suspend_0() [c] then L6
L6: r47 <- v45 'and' 2 if eq then L7 (0.50) else L8
L7: v48 <- [v45+3]
    v49 <- 31
    r51 <- v46 'and' 31
    r52 <- r51 'and' 15
    if (r52 eq 15) then L12 (0.99) else L11
L12:v50 <- v46 add 16 if overflow then L11 (0.01) else L10
L11:v50 <- '+'(v46, v49) [c] then L10
L10:v45 <- v48
    v46 <- v50
    goto L3
L8: return(v46)
```

```
length:
%% Info
1:
    red
    if
3:
    v5
    v8
    v0
    got
10:
    return(v0)
```

A HiPE-enabled Erlang/OTP system

C




```
.section    ".text"
    .align 4
    .global length_len_2
.section    ".data"
.length_len_2_dl_0: .word 0 ! .term [{length,len,2}]
.section    ".code"
length_len_2:
.length_len_2_22:
    add %i3, 16, %i3
    stw %o7, [%i3+-16]
    mov %o2, %g3
    mov %o1, %l5
.length_len_2_3:
    subcc %i5, 1, %i5
    bge %icc, .length_len_2_6
    nop
.length_len_2_5:
    stw %g3, [%i3+-4]
    call suspend_0  ! () <>[c]<|4| Live: [0,1]>
    stw %l5, [%i3+-8]
    lduw [%i3+-4], %g3
    lduw [%i3+-8], %l5
.length_len_2_6:
    andcc %g3, 2, %g4
    be,pn %icc, .length_len_2_7
    nop
```

```
.length_len_2_8:
    mov %l5, %o0
    lduw [%i3+-16], %o7
    jmpl %o7+8, %g0 ! (%o0)
    sub %i3, 16, %i3
.length_len_2_7:
    and %l5, 31, %o4
    and %o4, 15, %o5
    subcc %o5, 15, %g0
    be %icc, .length_len_2_12
    lduw [%g3+3], %g5
.length_len_2_11:
    mov %l5, %o1
    mov 31, %o2
    call '+'  ! (%o1, %o2) <%o0>[c]<|4| Live: [2]>
    stw %g5, [%i3+-12]
    lduw [%i3+-12], %g5
.length_len_2_19:
    mov %o0, %l0
.length_len_2_10:
    mov %g5, %g3
    ba .length_len_2_3
    mov %l0, %l5
.length_len_2_12:
    addcc %l5, 16, %l0
    bvc %icc, .length_len_2_10
    nop
    ba .length_len_2_11
    nop
```

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 component (currently about 100,000 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

Machine-specific parts

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.

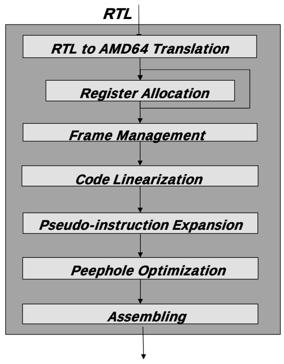
HiPE Compiler: SPARC back-end

- Parameter-passing in registers (up to 16)
- Register allocation based on choice between a Briggs-style graph coloring, iterated register coalescing, optimistic coalescing, or a linear scan algorithm [SPE'03]
  - Iterated coalescing default on x86 and AMD-64
  - Linear scan default on SPARC and PowerPC
- Cache-conscious code linearization
- Garbage collection:
  - Based on *two-generational copying*
  - Aided by *stack descriptors* (live-variable maps)
  - Performs *generational stack collection.*

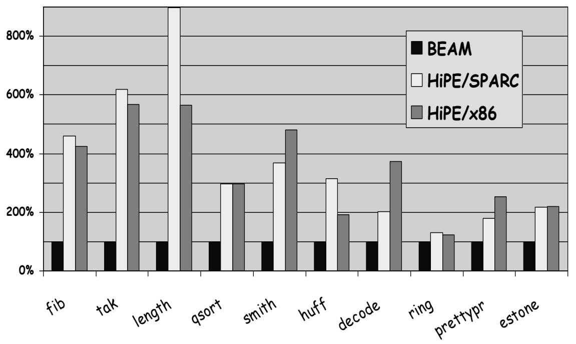
HiPE Compiler: x86 and AMD-64 backends

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

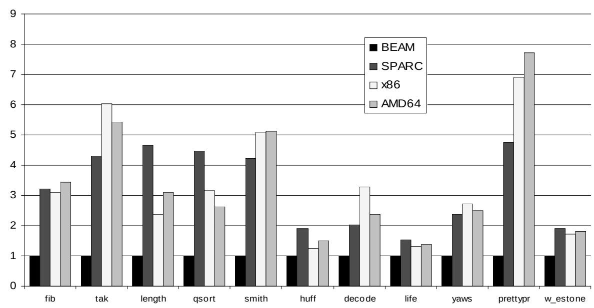
Backend Passes



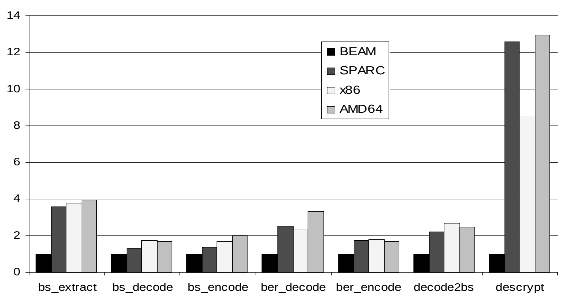
Performance of HiPE on SPARC & x86 (Feb 2002)



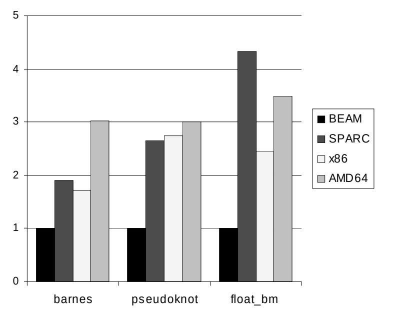
Performance Comparison on more platforms



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