Dynamc compilation

Fabian Gruber

19. Juni 2012

Goals

- Write a JIT for a very simple Lisp like language
- There already is an interpreter for that language
- The JIT and the interpreter should interoperate

The language

Reader syntax

integers	1, 5, 23, 1_000_000
floating point numbers	1.5, 3.14, 5., .7
symbols	true, false, +, *
strings	"hello world"
characters	\a, \b, \c, \tab \newline
lists	(), (1 2 3), (1 (2 3) 4)
quotation	'5, '"foo", 'symbol, '(a b c)

Special forms

function application	(+ 5 6)
quotation	(quote (1 2 3)), (quote 5)
if-then-else	(if true 5 7)
expression sequence	(do (println "hi") 5)
variable binding	(let (a 5) (b 6) (+ a b)
	(let* (a 5) (b (+ a 1) (+ a b)
lambda expression	(lambda (a b c) (* a (+ b c)))
global variable	(define pi 3.14)

Refs

- All data types are immutable
- Except refs
- (define foo (ref))
- (set foo 42)
- (get foo) $\rightarrow$ 42

Main components

- **Reader**
- **Analyzer**
- **Interpreter/JIT**

Reader

- Parses strings/files into S-expressions
- Simple hand written recursive descent parser

Analyzer

- 'Parses' S-expressions into AST
- Performs some simple checks on the source
- Also gathers some type information

Interpreter/JIT

- Meta circular interpreter evaluates expressions into values
- When functions are called often enough the JIT is invoked
- The JIT performs some more analysis and optimization

Values

- All values are allocated on the heap
- Every value starts with a 8 byte tag.
- The tag identifies a values type and a functions arity

Functions

For every function there is an object containing

- a pointer to the executable code,
- a pointer to the functions AST,
- a counter for how often the function has been called

Closures

Functions can be nested, thus we need to support closures.
Closure objects contains:

- A type/arity tag (like any other object)
- A pointer to the executable code
- A pointer to the shared function object
- An array of the values the closure captures

Features of the JIT

- Tail call removal for self calls
- Inlining of small functions
- Remove null & type check before function calls in simple cases

A short Demo

Summary

Have I met all my goals?

Have I met all my goals?
Not everything worked out as planned

Have I met all my goals?

Not everything worked out as planned

- Write a JIT for a very simple Lisp like language

Have I met all my goals?

Not everything worked out as planned

- Write a JIT for a very simple Lisp like language

  Ok

Have I met all my goals?

Not everything worked out as planned

- Write a JIT for a very simple Lisp like language

  Ok

- There already is an interpreter for that language

Have I met all my goals?

Not everything worked out as planned

- Write a JIT for a very simple Lisp like language

  Ok

- There already is an interpreter for that language

  Didn't gather enough information, it had to be rewritten

Have I met all my goals?

Not everything worked out as planned

- Write a JIT for a very simple Lisp like language

  Ok

- There already is an interpreter for that language

  Didn't gather enough information, it had to be rewritten

- The JIT and the interpreter should interoperate

Have I met all my goals?

Not everything worked out as planned

- Write a JIT for a very simple Lisp like language

  Ok

- There already is an interpreter for that language

  Didn't gather enough information, it had to be rewritten

- The JIT and the interpreter should interoperate

  The JIT can only call compiled functions

Have I met all my goals?
Not everything worked out as planned

- Write a JIT for a very simple Lisp like language
  Ok
- There already is an interpreter for that language
  Didn't gather enough information, it had to be rewritten
- The JIT and the interpreter should interoperate
  The JIT can only call compiled functions
  $\Rightarrow$ functions are compiled on demand

Other problems

- I Planned to use LLVM as my code generator

Other problems

- I Planned to use LLVM as my code generator
  - Too heavy weight for this small project

Other problems

- I Planned to use LLVM as my code generator
  - Too heavy weight for this small project

Other problems

- I Planned to use LLVM as my code generator
  - Too heavy weight for this small project
- Now I use libjit

Other problems

- I Planned to use LLVM as my code generator
  - Too heavy weight for this small project
- Now I use libjit
  - libjit's tail call optimizer is broken

Other problems

- I Planned to use LLVM as my code generator
  - Too heavy weight for this small project
- Now I use libjit
  - libjit's tail call optimizer is broken
  - It seems that lijit is no longer being developed

Other problems

- I Planned to use LLVM as my code generator
  - Too heavy weight for this small project
- Now I use libjit
  - libjit's tail call optimizer is broken
  - It seems that lijit is no longer being developed
  - Had to repair it

Other problems

- I Planned to use LLVM as my code generator
  - Too heavy weight for this small project
- Now I use libjit
  - libjit's tail call optimizer is broken
  - It seems that lijit is no longer being developed
  - Had to repair it

Trivia

- 5000 lines of C++ code
- Requires a C++11 compiler
- 50 lines of C code for libjit patch
- Only external dependency is libjt

Thank you for your attention

Questions?