Opeth

Lua VM Bytecode Optimizer

Nymphium

February 12, 2017 at tsukuba.lua

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●Intro

こんにちは、びしょ〜じょです。

- ここの大学に４年滞在中の３年生
- Lua/MoonScriptをよく書く
- ライトなメタラー

● 流れ

1. tsukuba.pm というイベントで Lua のバイトコード解析 [*1]
2. あまり最適化されてないことが判明
3. optimizer 作るか

[*1] http://nymphium.github.io/pdf/tsukubapm3-luavm.html

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●Lua is . . . . . .

- 弱い動的型付けなスクリプト言語
- 文法が簡単、予約語も 22 個と少ない

●Lua is . . . . . .

- 弱い動的型付けなスクリプト言語
- 文法が簡単、予約語も22個と少ない
- 関数がファーストクラス
  - ナウい関数型プログラミングも可能

●Lua is . . . . . .

- 弱い動的型付けなスクリプト言語
- 文法が簡単、予約語も22個と少ない
- 関数がファーストクラス
    - ナウい関数型プログラミングも可能
- 唯一のデータ構造 table
    - 簡単に言うと連想配列
    - オブジェクトは全部キーにも要素にも
    - メタテーブルで色々拡張

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
- Optimizations
  - Constant Folding
  - Constant Propagation
  - Dead-Code Elimination
  - Function Inlining
  - Unreachable Block Removal
  - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●Implementations

- (PUC-Lua)
  - リオデジャネイロ·カトリカ大学開発の、いわゆる本家
  - 軽量、組み込みで広く活躍
- LuaJIT
  - だいぶ速い。FFI モジュールなども提供
- LuaJ、Rembulan
  - JVM 実装。
- その他色々

●Implementations

- (PUC-Lua)
  - リオデジャネイロ·カトリカ大学開発の、いわゆる本家
  - 軽量、組み込みで広く活躍
- LuaJIT
  - だいぶ速い。FFI モジュールなども提供
- LuaJ、Rembulan
  - JVM 実装。
- その他色々

●Implementations

その他

- llix
  - 拙作。例外処理構文を追加
- TypedLua
  - 型アノテーション、型定義ファイルなど。トランスパイラ
  - GSoC で募集してたり [*2]
- Ravi
  - LLVM+Lua の文法+α。別言語
- Terra
  - multi-stage programming
- MoonScript
  - altLua 的なモノ。

[*2] https://summerofcode.withgoogle.com/archive/2016/organizations/4733835644239872/

●Implementations

その他

- llix
  - 拙作。例外処理構文を追加
- TypedLua
  - 型アノテーション、型定義ファイルなど。トランスパイラ
  - GSoC で募集してたり [*2]
- Ravi
  - LLVM+Lua の文法+α。別言語
- Terra
  - multi-stage programming
- MoonScript
  - altLua 的なモノ。ちょっとコントリビュート

[*2] `https://summerofcode.withgoogle.com/archive/2016/organizations/4733835644239872/`

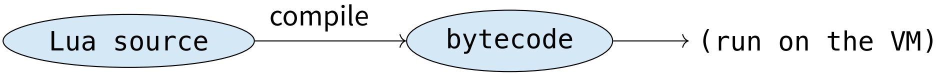
Opeth, Lua VM Bytecode Optimizer

- Intro
    - Lua is . . . . . .
    - Implementations
        - Other
    - About Lua VM
- Opeth
    - Demo
- Implementation
    - Architecture
    - Bytecode
    - Dataflow analysis
        - Control Flow Graph
    - Optimizations
        - Constant Folding
        - Constant Propagation
        - Dead-Code Elimination
        - Function Inlining
        - Unreachable Block Removal
        - Unused Resource Removal
- Benchmark
- Tools
    - lvis
    - moonstep
    - lasmc
- Conclusion
- appendix

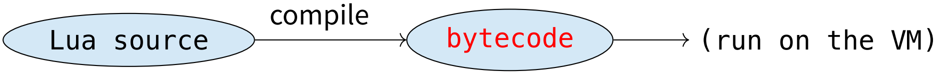
●About Lua VM

- PUC-Lua
- レジスターベース (Lua 5.0〜)
- 関数呼び出しはレジスターウィンドウ
- 47 個の命令 (Lua 5.3)

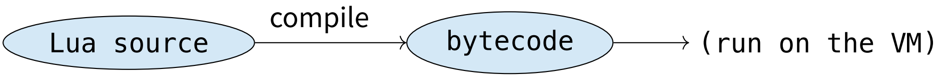
Lua VM, register-based Virtual Machine



Lua VM, register-based Virtual Machine



Lua VM, register-based Virtual Machine



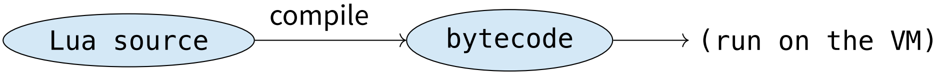
積極的に最適化が行われない

```lua
local x = 3
local y = 5
print(x + y)
```

⇒
compile

```
LOADK    0 0
LOADK    1 1
GETTABUP 2 0 -3
ADD      3 0  1
CALL     2 2  1
RETURN   0 1
```

Lua VM, register-based Virtual Machine



積極的に行われない

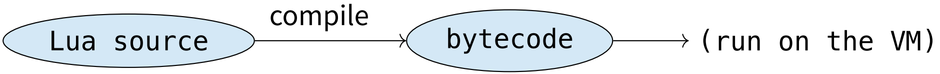
```
local x = 3
local y = 5
print(x + y)
```

⇒
compile

```
LOADK     0  0
LOADK     1  1
GETTABUP  2  0  -3
ADD       3  0   1
  LL      2  2   1
 TURN     0  1
```

コンパイル時に値が ↑
分かる (定数化可能)

Lua VM, register-based Virtual Machine



積極的に最適化か

足し算の結果が分かれば
この定数はいらない →

```
local x = 3
local y = 5
print(x + y)
```

⇒
compile

```
LOADK     0  0
LOADK     1  1
GETTABUP  2  0  -3
ADD       3  0   1
 LL       2  2   1
 TURN     0  1
```

コンパイル時に値が ↑
分かる (定数化可能)

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

Q. Do you know a metal band, Opeth?

Q. Do you know a metal band, Opeth?

Figure: Opeth、新譜出すってよ

つくった optimizer

- https://github.com/Nymphium/opeth
- 『情報特別演習 II』[*3] という通年の講義で制作
- コマンドラインから使用可能
- モジュールとしても使える

[*3] http://www.coins.tsukuba.ac.jp/syllabus/GB13312_GB13322.html

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

デモ

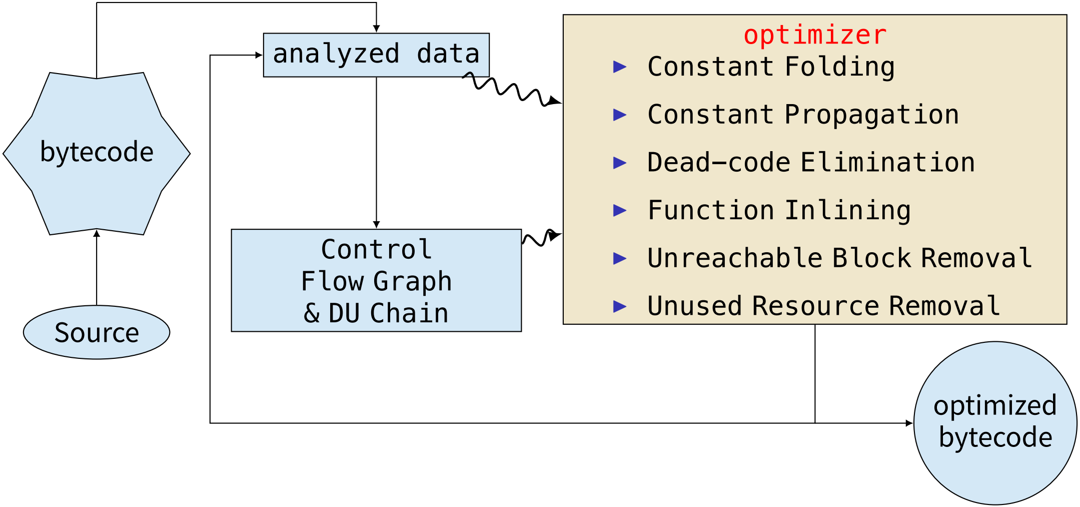
●Demo

```
optimizer = require'opeth.opeth'

f = -> ......
......

g = optimizer f
g! -- wow
```

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

•Architecture



- Constant Folding
- Constant Propagation
- Dead–code Elimination
- Function Inlining
- Unreachable Block Removal
- Unused Resource Removal

Figure: optimization image

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
- Optimizations
  - Constant Folding
  - Constant Propagation
  - Dead-Code Elimination
  - Function Inlining
  - Unreachable Block Removal
  - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●Bytecode

Lua VM 5.3 のバイトコードを操作したい ⇒ バイトコードの document はない

●Bytecode

Lua VM 5.3 のバイトコードを操作したい ⇒ バイトコードの document は ない

⇒ 自分で読み解くしかない

●Bytecode

有志の非公式ドキュメント

- Lua VM 5.3 instructions (bytecode ではない)[*4]
- Lua VM 5.1 reference[*5]

[*4] https://github.com/dibyendumajumdar/ravi/blob/master/readthedocs/lua_bytecode_reference.rst

[*5] http://luaforge.net/docman/83/98/ANoFrillsIntroToLua51VMInstructions.pdf

[*6] https://www.lua.org/source

●Bytecode

有志の非公式ドキュメント

- Lua VM 5.3 instructions (bytecode ではない)[*4]
- Lua VM 5.1 reference[*5]

Lua VM bytecode を読むためのツール

- `luac -l -l luac.out`
- `xxd -g 1 luac.out | nvim - -R`

[*4] https://github.com/dibyendumajumdar/ravi/blob/master/readthedocs/lua_bytecode_reference.rst

[*5] http://luaforge.net/docman/83/98/ANoFrillsIntroToLua51VMInstructions.pdf

[*6] https://www.lua.org/source

●Bytecode

有志の非公式ドキュメント

- Lua VM 5.3 instructions (bytecode ではない)[*4]
- Lua VM 5.1 reference[*5]

Lua VM bytecode を読むためのツール

- `luac -l -l luac.out`
- `xxd -g 1 luac.out | nvim - -R`
- ソースコード[*6]

[*4] https://github.com/dibyendumajumdar/ravi/blob/master/readthedocs/lua_bytecode_reference.rst

[*5] http://luaforge.net/docman/83/98/ANoFrillsIntroToLua51VMInstructions.pdf

[*6] https://www.lua.org/source

●Bytecode

有志の非公式ドキュメント

- Lua VM 5.3 instructions (bytecode ではない)[4]
- Lua VM 5.1 reference[5]

Lua VM bytecode を読むためのツール

- `luac -l -l luac.out`
- `xxd -g 1 luac.out | nvim - -R`
- ソースコード[6]

簡単に言うと**気合**

[4] https://github.com/dibyendumajumdar/ravi/blob/master/readthedocs/lua_bytecode_reference.rst

[5] http://luaforge.net/docman/83/98/ANoFrillsIntroToLua51VMInstructions.pdf

[6] https://www.lua.org/source

Bytecode

```
print("hello, world!")
```

```
 $ luac -l -l luac.out

main <hello.lua:0,0> (4 instructions at
     0x16e79e0)
0+ params, 2 slots, 1 upvalue, 0 locals,
      2 constants, 0 functions
  1 [1] GETTABUP  0 0 -1 ; _ENV "print"
  2 [1] LOADK     1 -2  ; "hello, world!"
  3 [1] CALL      0 2 1
  4 [1] RETURN    0 1
constants (2) for 0x16e79e0:
  1 "print"
  2 "hello, world!"
locals (0) for 0x16e79e0:
upvalues (1) for 0x16e79e0:
  0 _ENV  1     0
```

•Bytecode

```
print("hello, world!")
```

```
 $ luac -l -l luac.out

main <hello.lua:0,0> (4 instructions at
      0x16e79e0)
0+ params, 2 slots, 1 upvalue, 0 locals,
       2 constants, 0 functions
  1 [1] GETTABUP  0 0 -1 ; _ENV "print"
  2 [1] LOADK     1 -2  ; "hello, world!"
  3 [1] CALL      0 2 1
  4 [1] RETURN    0 1
constants (2) for 0x16e79e0:
  1 "print"
  2 "hello, world!"
locals (0) for 0x16e79e0:
upvalues (1) for 0x16e79e0:
  0 _ENV  1     0
```

```
$ xxd -g 1 luac.out
00000000: 1b 4c 75 61 53 00 19 93 0d 0a 1a 0a 04 08 04 08 .LuaS...........
00000010: 08 78 56 00 00 00 00 00 00 00 00 00 00 00 28 77 .xV...........(w
00000020: 40 01 0b 40 68 65 6c 6c 6f 2e 6c 75 61 00 00 00 @..@hello.lua...
00000030: 00 00 00 00 00 00 02 02 04 00 00 00 06 00 40 00 ..............@.
00000040: 41 40 00 00 24 40 00 01 26 00 80 00 02 00 00 00 A@..$@..&.......
00000050: 04 06 70 72 69 6e 74 04 0e 68 65 6c 6c 6f 2c 20 ..print..hello, 
00000060: 77 6f 72 6c 64 21 01 00 00 00 01 00 00 00 00 00 world!..........
00000070: 04 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 ................
00000080: 01 00 00 00 00 00 00 00 01 00 00 00 05 5f 45 4e ............._EN
00000090: 56                                              V
```

•Bytecode

```
print("hello, world!")
```

```
 $ luac -l -l luac.out

main <hello.lua:0,0> (4 instructions at
      0x16e79e0)
0+ params, 2 slots, 1 upvalue, 0 locals,
       2 constants, 0 functions
  1 [1] GETTABUP  0 0 -1 ; _ENV "print"
  2 [1] LOADK     1 -2  ; "hello, world!"
  3 [1] CALL      0 2 1
  4 [1] RETURN    0 1
constants (2) for 0x16e79e0:
  1 "print"
  2 "hello, world!"
locals (0) for 0x16e79e0:
upvalues (1) for 0x16e79e0:
  0 _ENV  1     0
```

```
$ xxd -g 1 luac.out
00000000: 1b 4c 75 61 53 00 19 93 0d 0a 1a 0a 04 08 04 08 .LuaS...........
00000010: 08 78 56 00 00 00 00 00 00 00 00 00 00 00 28 77 .xV...........(w
00000020: 40 01 0b 40 68 65 6c 6c 6f 2e 6c 75 61 00 00 00 @..@hello.lua...
00000030: 00 00 00 00 00 00 02 02 04 00 00 00 06 00 40 00 ..............@.
00000040: 41 40 00 00 24 40 00 01 26 00 80 00 02 00 00 00 A@..$@..&.......
00000050: 04 06 70 72 69 6e 74 04 0e 68 65 6c 6c 6f 2c 20 ..print..hello,
00000060: 77 6f 72 6c 64 21 01 00 00 00 01 00 00 00 00 00 world!..........
00000070: 04 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 ................
00000080: 01 00 00 00 00 00 00 00 01 00 00 00 05 5f 45 4e ............._EN
00000090: 56                                              V
```

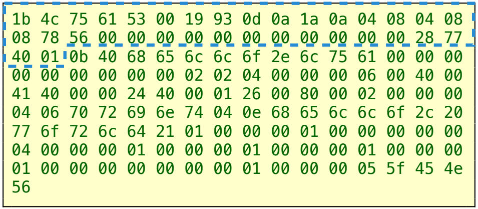
???

•Bytecode

```
1b 4c 75 61 53 00 19 93 0d 0a 1a 0a 04 08 04 08
08 78 56 00 00 00 00 00 00 00 00 00 00 00 28 77
40 01 0b 40 68 65 6c 6c 6f 2e 6c 75 61 00 00 00
00 00 00 00 00 00 02 02 04 00 00 00 06 00 40 00
41 40 00 00 24 40 00 01 26 00 80 00 02 00 00 00
04 06 70 72 69 6e 74 04 0e 68 65 6c 6c 6f 2c 20
77 6f 72 6c 64 21 01 00 00 00 01 00 00 00 00 00
04 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
01 00 00 00 00 00 00 00 01 00 00 00 05 5f 45 4e
56
```

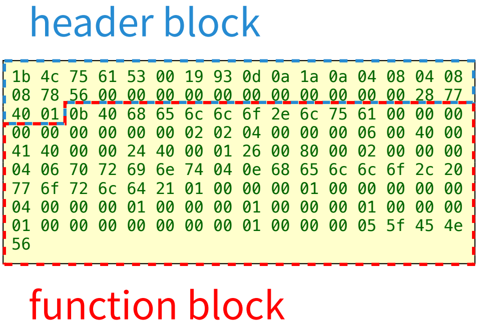
●Bytecode

header block



```
1b 4c 75 61 53 00 19 93 0d 0a 1a 0a 04 08 04 08
08 78 56 00 00 00 00 00 00 00 00 00 00 00 28 77
40 01 0b 40 68 65 6c 6c 6f 2e 6c 75 61 00 00 00
00 00 00 00 00 00 02 02 04 00 00 00 06 00 40 00
41 40 00 00 24 40 00 01 26 00 80 00 02 00 00 00
04 06 70 72 69 6e 74 04 0e 68 65 6c 6c 6f 2c 20
77 6f 72 6c 64 21 01 00 00 00 01 00 00 00 00 00
04 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
01 00 00 00 00 00 00 00 01 00 00 00 05 5f 45 4e
56
```

●Bytecode



header block

```
1b 4c 75 61 53 00 19 93 0d 0a 1a 0a 04 08 04 08
08 78 56 00 00 00 00 00 00 00 00 00 00 00 28 77
40 01 0b 40 68 65 6c 6c 6f 2e 6c 75 61 00 00 00
00 00 00 00 00 00 02 02 04 00 00 00 06 00 40 00
41 40 00 00 24 40 00 01 26 00 80 00 02 00 00 00
04 06 70 72 69 6e 74 04 0e 68 65 6c 6c 6f 2c 20
77 6f 72 6c 64 21 01 00 00 00 01 00 00 00 00 00
04 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
01 00 00 00 00 00 00 00 01 00 00 00 05 5f 45 4e
56
```

function block

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●Dataflow analysis

- Control Flow Graph (CFG)
  - プログラムの流れをグラフで表したもの
- Define-Use / Use-Define Chain (DU/UD Chain)
  - 変数の定義、使用を調べる
  - 役割としては SSA、A 正規形

●Dataflow analysis

```
local b = true

if b then
 print("hello")
else
 print"world"
end
```

Dataflow analysis

```
local b = true

if b then
  print("hello")
else
  print"world"
end
```

⇓

```
LOADBOOL   0  1   0
TEST       0  0
JMP        0  4
GETTABUP   1  0  -1
LOADK      2  1
CALL       1  2   1
JMP        0  3
GETTABUP   1  0  -1
LOADK      2  2
CALL       1  2   1
RETURN     0  1
```

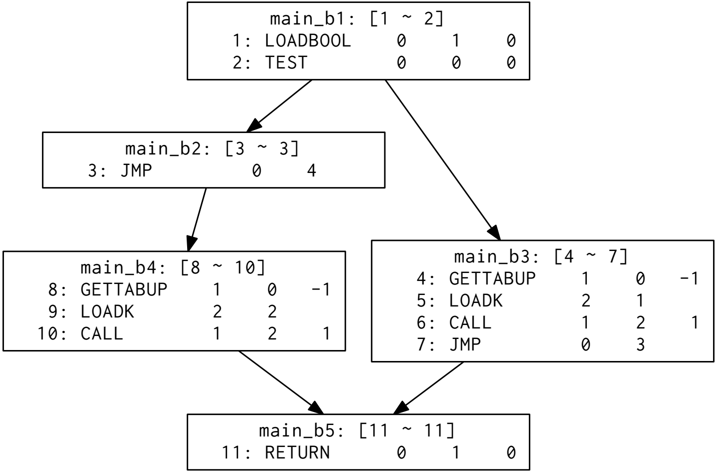
●Dataflow analysis

```
local b = true

if b then
  print("hello")
else
  print"world"
end
```

⇓

```
LOADBOOL   0  1   0
TEST       0  0
JMP        0  4
GETTABUP   1  0  -1
LOADK      2  1
CALL       1  2   1
JMP        0  3
GETTABUP   1  0  -1
LOADK      2  2
CALL       1  2   1
RETURN     0  1
```

⇒



```
      main_b1: [1 ~ 2]
 1: LOADBOOL     0     1     0
 2: TEST         0     0     0
```

```
      main_b2: [3 ~ 3]
 3: JMP          0     4
```

```
      main_b4: [8 ~ 10]
 8: GETTABUP     1     0    -1
 9: LOADK        2     2
10: CALL         1     2     1
```

```
      main_b3: [4 ~ 7]
 4: GETTABUP     1     0    -1
 5: LOADK        2     1
 6: CALL         1     2     1
 7: JMP          0     3
```

```
      main_b5: [11 ~ 11]
11: RETURN       0     1     0
```

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●Optimizations

- Constant Folding
- Constant Propagation
- Dead-Code Elimination
- Function Inlining
- Unreachable Block Removal
- Unused Resource Removal

●Constant Folding

1. 演算命令のオペランドの型を調べて
2. `table`、`userdata`以外なら
3. 値を取ってきて
4. 演算をおこない
5. 即値命令にswap

●Constant Folding

1. 演算命令のオペランドの型を調べて
2. `table`、`userdata`以外なら
3. 値を取ってきて
4. 演算をおこない
5. 即値命令にswap

●Constant Folding

1. 演算命令のオペランドの型を調べて
2. `table`、`userdata`以外なら ⇐ メタメソッドを考慮
3. 値を取ってきて
4. 演算をおこない
5. 即値命令にswap

●Constant Propagation

1. MOVE命令が参照してる register の定義位置を見て
2. LOADKなら MOVEを LOADKにする

●Constant Propagation

1. MOVE命令が参照してる register の定義位置を見て
2. LOADKなら MOVEを LOADKにする

- ▸ 単体では速度改善なさそう
- ▸ LOADKへの依存が減るので、他の最適化を有利に進められる

●Constant Propagation

1. MOVE命令が参照してる register の定義位置を見て
2. LOADKなら MOVEを LOADKにする

- ▸ 単体では速度改善なさそう
- ▸ LOADKへの依存が減るので、他の最適化を有利に進められる
  (今回の実装では) いまいちぱっとしない

●Dead-Code Elimination

1. `LOADK`、`MOVE`、`CLOSURE`、`LOADNIL`が生成する registr の使用を調べ
2. 0 個の場合命令を消す

●Dead-Code Elimination

1. `LOADK`、`MOVE`、`CLOSURE`、`LOADNIL`が生成するregistrの使用を調べ
2. 0個の場合命令を消す

- DU/UD Chainのわかりやすい使用例

●Function Inlining

1. CALL命令が引っ張ってくるclosureを見て
2. 再帰関数でなければ展開

●Function Inlining

1. `CALL`命令が引っ張ってくる closure を見て
2. 再帰関数でなければ展開

- register window の使用を抑えられる

●Function Inlining

1. CALL命令が引っ張ってくる closure を見て
2. 再帰関数でなければ展開

- register window の使用を抑えられる
- 実は頼みの綱

●Function Inlining

1. CALL命令が引っ張ってくるclosureを見て
2. 再帰関数でなければ展開

- register windowの使用を抑えられる
- 実は頼みの綱
- バグがヤバいア

●Unreachable Block Removal

1. 後続ブロックを持たない基本ブロックを丸々削除
2. だけ

- ▸ 速くはならないがバイトコードのサイズ縮小に貢献

●Unused Resource Removal

1. constant list、prototype list から不要なものを削除
2. だけ

- ▸ 速くはならないがバイトコードのサイズ縮小に貢献

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- **Benchmark**
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●Benchmark

```
local function pow(i)
 return i * i
end

local a = {}

for i = 1, 10000000 do
 a[i] = pow(i)
end
```

●Benchmark

```
local function pow(i)
 return i * i
end

local a = {}

for i = 1, 10000000 do
 a[i] = pow(i)
end
```

⇒

```
......
FORPREP     2 4
MOVE        6 0
MOVE        7 5
CALL        6 2 2
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

•Benchmark

```lua
local function pow(i)
 return i * i
end

local a = {}

for i = 1, 10000000 do
 a[i] = pow(i)
end
```

⇒

```
......
FORPREP     2 4
MOVE        6 0
MOVE        7 5
CALL        6 2 2
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

⇓

```
......
FORPREP     2 4
MOVE        7 5
MUL         8 7 7
MOVE        6 8
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

•Benchmark

```
local function pow(i)
  return i * i
end

local a = {}

for i = 1, 10000000 do
  a[i] = pow(i)
end
```

⇒

```
......
FORPREP     2 4
MOVE        6 0
MOVE        7 5
CALL        6 2 2
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

⇓

```
......
FORPREP     2 4
MOVE        7 5
MUL         8 7 7
MOVE        6 8
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

•Benchmark

```
local function pow(i)
  return i * i
end

local a = {}

for i = 1, 10000000 do
  a[i] = pow(i)
end
```

⇒

```
......
FORPREP     2 4
MOVE        6 0
MOVE        7 5
CALL        6 2 2
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

⇓

```
......
FORPREP     2 4
MOVE        7 5
MUL         8 7 7
MOVE        6 8
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

●Benchmark

```
local function pow(i)
  return i * i
end

local a = {}

for i = 1, 10000000 do
  a[i] = pow(i)
end
```

⇒

```
......
FORPREP     2 4
MOVE        6 0
MOVE        7 5
CALL        6 2 2
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

⇓

```
......
FORPREP     2 4
MOVE        7 5
MUL         8 7 7
MOVE        6 8
SETTABLE    1 5 6
FORLOOP     2 -5
......
```

⇐

1.4倍の高速化

ぶっちゃけ function inlining 以外微妙……

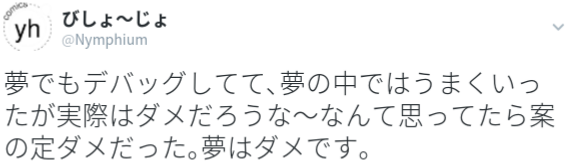
もう少し何かいいケースがあればあるいは……

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
- Optimizations
  - Constant Folding
  - Constant Propagation
  - Dead-Code Elimination
  - Function Inlining
  - Unreachable Block Removal
  - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

今回のマズイ点

- ツールなさすぎ
- 気合では解決できない
- 興味が逸れる



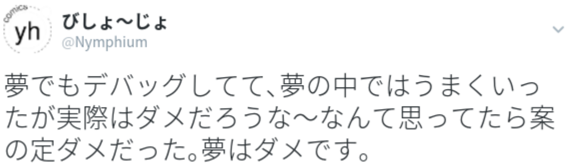
びしょ〜じょ
@Nymphium

夢でもデバッグしてて、夢の中ではうまくいったが実際はダメだろうな〜なんて思ってたら案の定ダメだった。夢はダメです。

●Tools

今回のマズイ点

- ツールなさすぎ
- 気合では解決できない
- 興味が逸れる



びしょ〜じょ
@Nymphium

夢でもデバッグしてて、夢の中ではうまくいったが実際はダメだろうな〜なんて思ってたら案の定ダメだった。夢はダメです。

ツール制作で英気を養う👈

Opeth, Lua VM Bytecode Optimizer

Intro

- Lua is . . . . . .
- Implementations
  - Other
- About Lua VM

Opeth

- Demo

Implementation

- Architecture
- Bytecode
- Dataflow analysis
  - Control Flow Graph

Optimizations

- Constant Folding
- Constant Propagation
- Dead-Code Elimination
- Function Inlining
- Unreachable Block Removal
- Unused Resource Removal

Benchmark

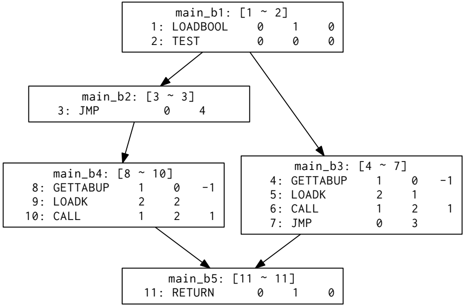
Tools

- lvis
- moonstep
- lasmc

Conclusion

appendix

- これの描画ツール ⇒
- ちょうど Graphviz の Lua binding(嘘)[7] 作ってた
- 目 grep から急速文明化



```
main_b1: [1 ~ 2]
 1: LOADBOOL     0     1     0
 2: TEST         0     0     0

main_b2: [3 ~ 3]
 3: JMP          0     4

main_b4: [8 ~ 10]
 8: GETTABUP     1     0    -1
 9: LOADK        2     2
10: CALL         1     2     1

main_b3: [4 ~ 7]
 4: GETTABUP     1     0    -1
 5: LOADK        2     1
 6: CALL         1     2     1
 7: JMP          0     3

main_b5: [11 ~ 11]
11: RETURN       0     1     0
```

Figure: visualise で小学生にも人気

[7] https://github.com/Nymphium/lua-graphviz

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
- Optimizations
  - Constant Folding
  - Constant Propagation
  - Dead-Code Elimination
  - Function Inlining
  - Unreachable Block Removal
  - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●moonstep

- step-by-step execution Lua VM
- gdbを目指した
- つい最近関数呼び出し内を追えるように

●moonstep

- step-by-step execution Lua VM
- gdbを目指した
- つい最近関数呼び出し内を追えるように

  もっとまともなデバッグツール出してほしい

デモ

Opeth, Lua VM Bytecode Optimizer

- Intro
    - Lua is . . . .
    - Implementations
        - Other
    - About Lua VM
- Opeth
    - Demo
- Implementation
    - Architecture
    - Bytecode
    - Dataflow analysis
        - Control Flow Graph
- Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
    - lvis
    - moonstep
    - lasmc
- Conclusion
- appendix

•lasmc

- アセンブリみたいにLua VMの命令を書きたいという願い
- いろいろ機能をたそうとしたら構文がごちゃごちゃになってやる気0
- 意外にもデバッグに貢献

```
main: 0 2
LOADK 0 0 -- load `3`
LOADK 1 1 -- load `5`
LOADK 1 2 -- load `7`
EQ 0 0 1 -- R(0) == R(1) ?
JMP 0 2
LOADK 2 2 -- load `7`
JMP 0 1
LOADK 2 3 -- load `9`
RETURN 0 1
{3 5 7 9} -- constant list
```

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- **Conclusion**
- appendix

●Conclusion

- 最適化器の実装
  - 一部高速化に成功
  - バイトコードの縮小化もぼちぼち
- しんどかった
  - ドキュメントは書こう
- 課題
  - 他の最適化も取り入れたい (for 展開とか)
  - 最適化器の最適化!
    - アルゴリズムが適当すぎ

Opeth, Lua VM Bytecode Optimizer

- Intro
  - Lua is . . . . . .
  - Implementations
    - Other
  - About Lua VM
- Opeth
  - Demo
- Implementation
  - Architecture
  - Bytecode
  - Dataflow analysis
    - Control Flow Graph
  - Optimizations
    - Constant Folding
    - Constant Propagation
    - Dead-Code Elimination
    - Function Inlining
    - Unreachable Block Removal
    - Unused Resource Removal
- Benchmark
- Tools
  - lvis
  - moonstep
  - lasmc
- Conclusion
- appendix

●appendix I

- font

  the King 26 Queen[*8]

[*8] http://www.dfonts.net/the_King__26_Queen_font.font/20679/

appendix I

- Nullstone Corporation, Compiler optimizations, http://www.compileroptimizations.com/.

- Jason D. Davies, Optimizing lua, 2005, https://www.jasondavies.com/optimising-lua/JasonDaviesDissertation.pdf.

- Ikuo Tanaka, Masataka Sasa, Munahiro Takimoto, and Tan Watanabe, コンパイラの基盤技術と実践 – コンパイラ・インフラストラクチャ coins を用いて, 2008.

appendix II

- Dibyendu Majumdar, Lua 5.3 bytecode reference, http://the-ravi-programming-language.readthedocs.io/en/latest/lua_bytecode_reference.html.

- Kein-Hong Man, A no-frills introduction to lua 5.1 vm instructions, 2006, http://luaforge.net/docman/83/98/ANoFrillsIntroToLua51VMInstructions.pdf.

- PUC Rio, source code for lua 5.3, https://www.lua.org/source/5.3/.

●appendix III

- Roberto Ierusalimuschy, Luiz Henrique de Figueiredo, and Waldemar Celes, The implementation of lua 5.0, 2003, `https://www.lua.org/doc/jucs05.pdf`.

- Michael Schroder, Optimizing lua using run-time type specialization, 2012, `https://www.complang.tuwien.ac.at/anton/praktika-fertig/schroeder/thesis.pdf`.