Lua i LuaJIT

- Stefan Pejić
- Đorđe Kovačević
- RT-RK, Computer Based Systems
- Fakultet tehničkih nauka, Novi Sad
- Radimo na programskom prevodiocu LuaJIT
- Današnje teme:
  - Programski jezik Lua
  - Programski prevodilac LuaJIT

Glavni ciljevi jezika

- Portabilnost
- Jednostavnost
- Kompaktnost
- Skriptovanje

Portabilnost

- Podržava **mnogo** arhitektura
- Veoma lako se prilagođava novim arhitekturama
- ANSI C
- LuaJIT nema ove osobine
  - Koristi asembler, noviji C standard i raznovrsne GCC ekstenzije

Jednostavnost

- Dinamički tipiziran jezik
  - Nema eksplicitnih tipova
- Interpretiran
  - Nema potrebe za prevođenjem
- Jednostavna osnovna sintaksa

Skriptovanje

- Lua je implentirana kao biblioteka
  - Take da je moguće korisiti iz drugih jezika
- Povezivanje sa drugim aplikacijama i jezicima
- Pogodan za ugrađivanje i proširivanje
  - Ugrađivanje: korišćenje Lue u aplikacijama koje nisu napisane u Lui
  - Proširivanje: pravljenje modula (biblioteka) za Luu u nekom drugom programskom jeziku

Upotreba Lue

- Veliki broj frameworka za pisanje igara
  - LÖVE, Corona, MOAI, itd.
  - Stingray 3D, Luxinia, Cafu 3D, itd.
- Igre proširive korišćenjem Lue:
  - The Witcher
  - Far Cry
  - World of Warcraft
  - Google „Lua-scripted video games“

Upotreba Lue

- Adobe Photoshop Lightroom
- VLC media player
- Apache web server
- Wikipedia
- Cisco Adaptive Security Appliance
- I mnogi drugi

Zanimljive osobine Lue

- Tabele
  - Jedan od osnovnih tipova u Lui
  - Mogu se tretirati kao nizovi ili kao mape
- Metatabele
  - Služe za definisanje operacija nad tipovima
- Funkcije
  - Kao i tabele, jedan od osnovnih tipova
- „Closure functions“
  - Anonimne funkcije koje imaju pristup promenljivama roditeljske funkcije

Kako početi?

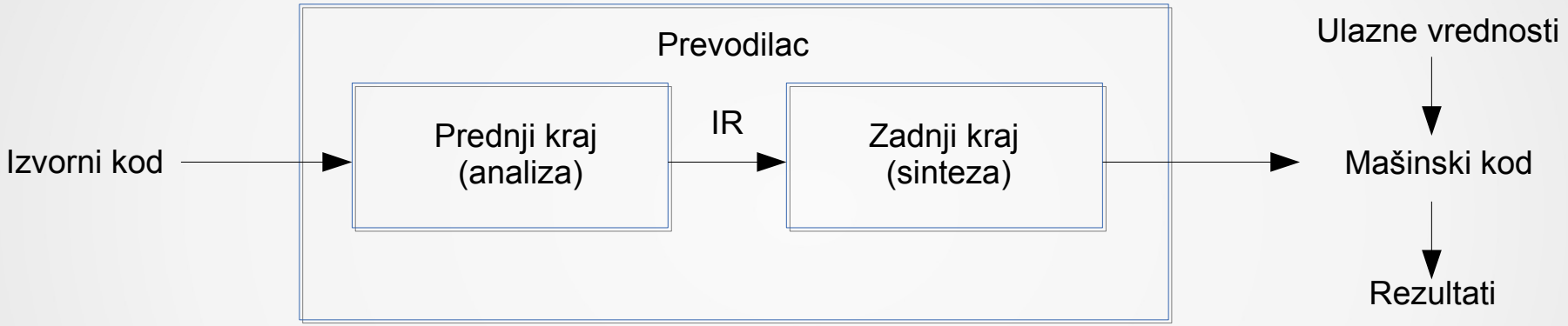
- Knjiga „Programming in Lua“, Roberto Ierusalimschy
- Lua zajednica: www.lua-users.org
- Mailing lista: http://www.lua.org/lua-l.html
- Lua priručnik: http://www.lua.org/manual/5.3/

Kako početi?

- Postoji više implementacija Lue:
  - Lua (zvanični kompajler) i LuaJIT su najpoznatiji kompajleri za Luu
  - http://www.lua.org/download.html
  - http://luajit.org/download.html
- Postoji nekoliko verzija Lue
  - 5.1, 5.2 i 5.3 su najčešće implementirane
  - Programi pisani za različite verzije Lue su nekada nekompatibilni

Programski prevodilac LuaJIT

- Programski prevodilac



- LuaJIT programski prevodilac za jezik Lua
  - Nezavisna realizacija interpretera Lue sa prevodiocem tipa JIT

Šta je interpreter?

- Prevodi kod u toku izvršavanja
- Direktno izvršava operacije definisane u izvornom programu nad ulaznim podacima
- Mane:
  - za svako novo izvršavanje ponavlja kompletno prevođenje
  - ukupno zahteva više vremena za izvršavanje ulaznog programa
- Prednosti:
  - koristi mnogo manje memorije
  - bolju dijagnostiku grešaka i interaktivno ispitivanje

Šta je prevodilac tipa JIT?

- Just-In-Time
- Prevodi deo koda (međukoda) tokom samog izvršavanja programa u formu čije izvršavanje je brže
- U većini programa postoje kritični delovi koda koji se izvršavaju uzastopno više puta
- Ideja:
  - Prevesti kritični deo koda samo jednom i sačuvati ga u memoriji
  - Iskoristiti preveden kod svaki naredni put kada treba da se izvrši isti segment

Programski prevodilac LuaJIT

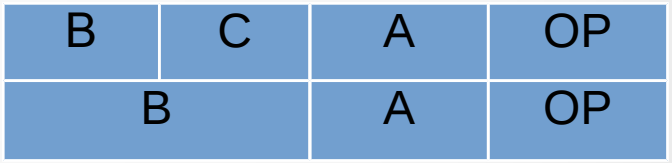
- Koristi se u igrama, grafičkim programima, numeričkim simulacijama, mrežnim i namenskim uređajima...
- LuaJIT se distribuira samo kao paket sa izvornim kodom
- Podržane arhitekture: x86, x64, PPC, ARM, ARM64, MIPS, MIPS64

Programski prevodilac LuaJIT

- Visoke performanse
  - Neuporedivo mala upotreba memorije
  - Znatno brže izvršavanje koda napisanog u Lui
- Fleksibilnost
  - Kompatibilan sa svim verzijama Lue do verzije 5.1
- Modul za operacije nad bitima
- FFI (Foreign Function Interface) modul

LuaJIT interpreter

- Napisan u asembleru (DynASM)
- Interpretira bajt kod (Bytecode)
  - Bajt kod instrukcija se zameni odgovarajućim mašinskim kodom
- Sopstveni format bajt koda
  - Dva formata instrukcija
  - Razlikovanje po tipu operanada (pr. ADDVN)



LuaJIT interpreter

Lua code:

```
-----------------------
local x = 5

for i = 1,100 do
  x = x + i
end

io.write(x)
```

Bytecode:

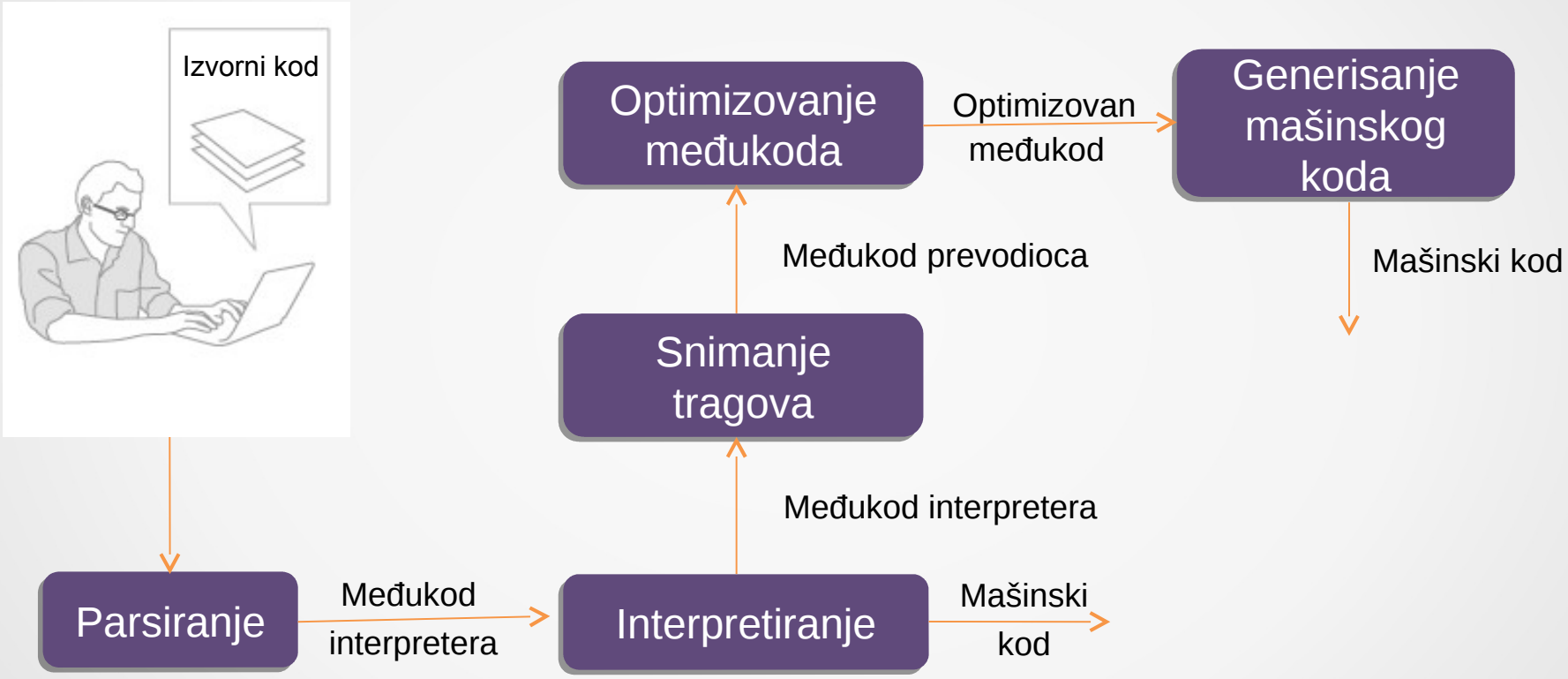
```
-----------------------------------
0001    KSHORT   0   5
0002    KSHORT   1   1
0003    KSHORT   2 100
0004    KSHORT   3   1
0005    FORI     1 => 0008
0006 => ADDVV    0   0   4
0007    FORL     1 => 0006
0008 => GGET     1   0      ; "io"
0009    TGETS    1   1   1  ; "write"
0010    MOV      2   0
0011    CALL     1   1   2
0012    RET0     0   1
```

LuaJIT interpreter

```
case BC_KSHORT:
  |  // RA = dst, RC = int16_literal
  |  sxth RCw, RCw
  |  add TMP0, RC, TISNUM
  |  str TMP0, [BASE, RA, lsl #3]
  |  ldr INSw, [PC], #4
  |  add TMP1, GL, INS, uxtb #3
  |   decode_RA RA, INS
  |  ldr TMP0, [TMP1, #GG_G2DISP]
  |   decode_RD RC, INS
  |  br TMP0
  break;
```

```
lj_BC_KSHORT:
    .long 0x13003f9c,0x8b180388,0xf83b7a68,0xb84046b0
    .long 0x8b300ec9,0xd3483e1b,0xf9479928,0xd3507e1c
    .long 0xd61f0100
```

Prevođenje tipa JIT



Prevođenje tipa JIT

- LuaJIT je prevodilac tipa *tracing JIT*
  - Vrši formiranje koda bazirano na tragovima u toku izvršavanja
- Realizovano u programskom jeziku C
- Generiše IR instrukcije na osnovu bajt koda
- IR instrukcije su:
  - 64 bita
  - Kod operacije i 2 operanda
  - Svaka ima tip povratne vrednosti
  - Jedinstveno referencirane svojim položajem u nizu

IR kod

Lua code:

```
----------------------
local x = 5

for i = 1, 100 do
  x = x + i
end

io.write(x)
```

IR code:

```
---------------------------------
0001     int SLOAD  #3    I
0002 >   int SLOAD  #2    T
0003 >+  int ADDOV  0002  0001
0004   + int ADD    0001  +1
0005 >   int LE     0004  +100
0006 ------ LOOP ------------
0007 >+  int ADDOV  0004  0003
0008   + int ADD    0004  +1
0009 >   int LE     0008  +100
0010     int PHI    0004  0008
0011     int PHI    0003  0007
```

Byte code:

```
--------------------------
---- TRACE 1 start test.lua:4
0006  ADDVV    0   0   4
0007  FORL     1 => 0006
```

IR kod

IR code:

```
---------------------------------

---- TRACE 1 IR
0001     int SLOAD  #3    I
0002 >   int SLOAD  #2    T
0003 >+  int ADDOV  0002  0001
0004   + int ADD    0001  +1
0005 >   int LE     0004  +100
0006 ------ LOOP ------------
0007 >+  int ADDOV  0004  0003
0008   + int ADD    0004  +1
0009 >   int LE     0008  +100
0010     int PHI    0004  0008
0011     int PHI    0003  0007
```

Machine code:

```
---------------------------------
---- TRACE 1 mcode 72
0058ff9c  mov   x0, #65529, lsl #16
0058ffa0  ldr   w27, [x19, #8]
0058ffa4  ldr   x28, [x19]
0058ffa8  cmp   x0, x28, lsr #32
0058ffac  bne   0x0058fff0   ->0
0058ffb0  adds  w28, w28, w27
0058ffb4  bvs   0x0058fff0   ->0
0058ffb8  add   w27, w27, #1
0058ffbc  cmp   w27, #100
0058ffc0  bgt   0x0058fff4   ->1
->LOOP:
0058ffc4  mov   x26, x28
0058ffc8  adds  w28, w27, w26
0058ffcc  bvs   0x0058fff8   ->2
0058ffd0  add   w27, w27, #1
0058ffd4  cmp   w27, #100
0058ffd8  ble   0x0058ffc4   ->LOOP
0058ffec  b     0x0058fffc   ->3
---- TRACE 1 stop -> loop
```

Interpreter vs. JIT

Interpreter:

```
Dump of assembler code for function lj_BC_ADDVV:
=> 0x00434308 <+0>:  ubfx   x17, x16, #24, #8
   0x0043430c <+4>:  and    x28, x28, #0xff
   0x00434310 <+8>:  ldr    x0, [x19,x17,lsl #3]
   0x00434314 <+12>: ldr    x1, [x19,x28,lsl #3]
   0x00434318 <+16>: cmp    x25, x0, lsr #32
   0x0043431c <+20>: b.ne   0x434350 <lj_BC_ADDVV+72>
   0x00434320 <+24>: cmp    x25, x1, lsr #32
   0x00434324 <+28>: b.ne   0x434350 <lj_BC_ADDVV+72>
   0x00434328 <+32>: adds   w0, w0, w1
   0x0043432c <+36>: b.vs   0x435eb8 <lj_vmeta_arith_vv>
   0x00434330 <+40>: add    x0, x0, x24
   0x00434334 <+44>: str    x0, [x19,x27,lsl #3]
   0x00434338 <+48>: ldr    w16, [x21],#4
   0x0043433c <+52>: add    x9, x22, w16, uxtb #3
   0x00434340 <+56>: ubfx   x27, x16, #8, #8
   0x00434344 <+60>: ldr    x8, [x9,#3888]
   0x00434348 <+64>: ubfx   x28, x16, #16, #16
   0x0043434c <+68>: br     x8
   0x00434350 <+72>: ldr    d0, [x19,x17,lsl #3]
   0x00434354 <+76>: ldr    d1, [x19,x28,lsl #3]
   0x00434358 <+80>: cmp    x25, x0, lsr #32
   0x0043435c <+84>: b.ls   0x435eb8 <lj_vmeta_arith_vv>
   0x00434360 <+88>: cmp    x25, x1, lsr #32
   0x00434364 <+92>: b.ls   0x435eb8 <lj_vmeta_arith_vv>
   0x00434368 <+96>: fadd   d0, d0, d1
   0x0043436c <+100>:str    d0, [x19,x27,lsl #3]
   0x00434370 <+104>:b      0x434338 <lj_BC_ADDVV+48>
Dump of assembler code for function lj_BC_FORL:
=> 0x00435480 <+0>:  lsr    x0, x21, #1
   0x00435484 <+4>:  and    x0, x0, #0x7e
   0x00435488 <+8>:  add    x0, x0, #0xeb0
   0x0043548c <+12>: ldrh   w1, [x22,x0]
   0x00435490 <+16>: subs   x1, x1, #0x2
   0x00435494 <+20>: strh   w1, [x22,x0]
   0x00435498 <+24>: b.cc   0x437060 <lj_vm_hotloop>
```

JIT:

```
0058ff9c  mov   x0, #65529, lsl #16
0058ffa0  ldr   w27, [x19, #8]
0058ffa4  ldr   x28, [x19]
0058ffa8  cmp   x0, x28, lsr #32
0058ffac  bne   0x0058fff0  ->0
0058ffb0  mov   w28, w28
0058ffb4  adds  w28, w28, w27
0058ffb8  bvs   0x0058fff0  ->0
0058ffbc  add   w27, w27, #1
0058ffc0  cmp   w27, #100
0058ffc4  bgt   0x0058fff4  ->1
->LOOP:
0058ffc8  adds  w28, w27, w28
0058ffcc  bvs   0x0058fff8  ->2
0058ffd0  add   w27, w27, #1
0058ffd4  cmp   w27, #100
0058ffd8  ble   0x0058ffc8  ->LOOP
0058ffdc  b     0x0058fffc  ->3
```

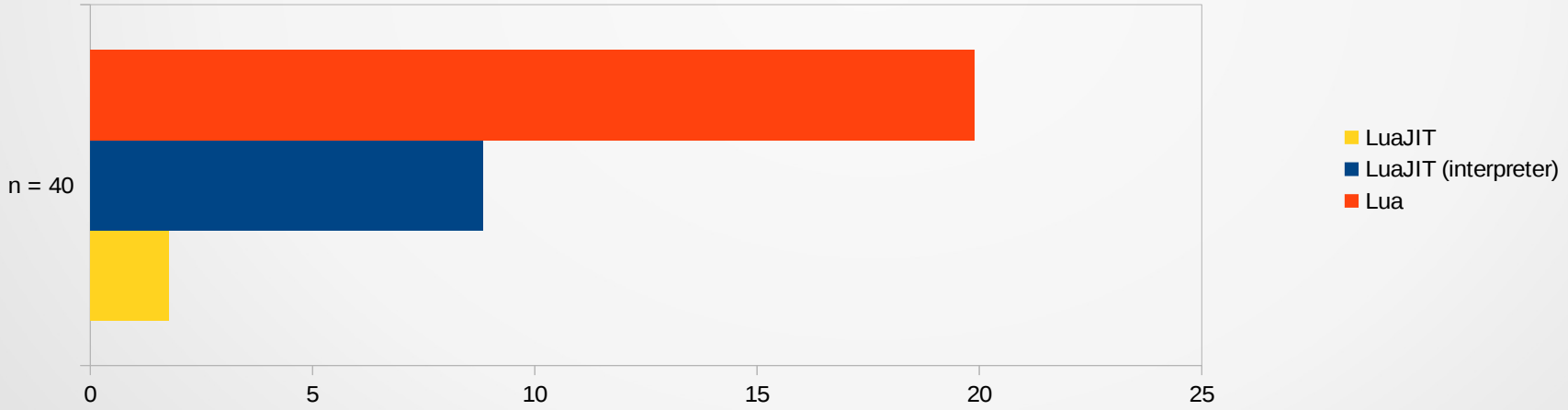
Byte code:

```
---- TRACE 1 start test.lua:4
0006  ADDVV    0   0   4
0007  FORL     1 => 0006
```

Uporedni test

```
local function fib(n)
  if n < 2 then return 1 end
  return fib(n-2) + fib(n-1)
end

local n = tonumber(arg[1]) or 10
io.write(string.format("Fib(%d): %d\n", n, fib(n)))
```



Prilagođavanje LuaJITa

- Arhitekturi MIPS32 bez jedinice za operacije u aritmetici pokretnog zareza (soft-float)
- Arhitekturi ARM64
- Arhitekturi MIPS64
- Koraci:
  - Prilagođavanje interpretera
  - Prilagođavanje prevođenja tipa JIT

Prilagođavanje LuaJITa

- Poznavanje koda LuaJITa
- Set instrukcija za MIPS i ARM arhitekture
- ABI za pomenute arhitekture
- Pronalaženje optimizacija u drugim kompajlerima (gcc, llvm) i njihova primena

Hvala na pažnji!

- Stefan Pejić – stefan.pejic@rt-rk.com
- Đorđe Kovačević – djordje.lj.kovacevic@rt-rk.com
- RT-RK, Computer Based Systems

  www.rt-rk.com