Update on Ravi

Dibyendu Majumdar
A new compiler for Ravi

• In this talk I will present the work I have been doing on a new compiler for Ravi. I will cover following areas:

• Benefits of the new compiler framework
  • C stack allocation of temporaries,
  • Ability to JIT or AOT compile
  • Two languages in one – Ravi and embedded C!

• Challenges faced

• Future directions
Previous Approach

Lua Parser → Byte Code → Translate to C → Generate machine code (MIR)
New Approach

- Compiler project is independent of Ravi
- However final C code generated assumes it’s the Ravi VM – various internal structures of Lua are referenced
Simple Example – Old Version

\[
f = \text{function}(a: \text{integer}) \text{ local } b: \text{integer} = a + a; \text{ return } b \times 10 \text{ end}
\]

**Generated code snippet**

```plaintext
ra = R(1);
rb = R(0);
rc = R(0);
setivalue(ra, ivalue(rb) + ivalue(rc));
ra = R(2);
rb = R(1);
setivalue(ra, ivalue(rb) \times 10);
```
Simple Example – New Version

\[ f = \text{function}(a: \text{integer}) \text{ local } \ b: \text{integer} = a + a; \text{ return } b * 10 \text{ end} \]

**Generated code snippet (local b eliminated)**

```plaintext
// ADDii {local(a, 0), local(a, 0)} {Tint(1)}
{ raviX__i_1 = ivalue(R(0)) + ivalue(R(0)); }

// MOVi {Tint(1)} {Tint(0)}
raviX__i_0 = raviX__i_1;

// MULii {Tint(0), 10 Kint(0)} {Tint(1)}
{ raviX__i_1 = raviX__i_0 * 10; }
```
Goals

• Create a standalone lexer and parser that are re-usable
• Translate to an Intermediate Representation – not byte code
• C code generation is not generic as requires knowledge of the VM details
• Generated C functions look like Lua functions but have no bytecode
Reusable lexer and parser

• Hope is that anyone that wants to play with this should be able to do so easily
• Work in progress parser AST dump that should be readable directly in Lua (a DSL)
Example of AST dump

```
ExprFunction {
  function_id = 0000002193E970B30,
  parent_function_id = 0000000000000000,
  type_code = closure,
  Type_name = '\',
  is_vararg = true,
  is_method = false,
  need_close = false,
  upvalues = {
    SymUpvalue {
      target_symbol_id = 0000002193E970B00,
      target_variable_name = '_ENV',
      target_type_code = table,
      target_type_name = '\',
    },
  },
  main_block =
    Scope {
      scope_id = 0000002193E970B90,
      function_id = 0000002193E970B30,
      parent_scope_id = 0000000000000000,
      symbols = {
      },
      need_close = false,
    },
};

StatReturn {
  ExprFunction {
    function_id = 0000002193E970D20,
    parent_function_id = 0000002193E970B30,
    type_code = closure,
    Type_name = '\',
    is_vararg = false,
    is_method = false,
    need_close = false,
    args = {
      SymLocalRef { name='a', symbol_id = 0000002193E970F90 }
    },
    main_block =
      Scope {
        scope_id = 0000002193E970D80,
        function_id = 0000002193E970B20,
        parent_scope_id = 0000002193E970B90,
        symbols = {
          SymLocal {
            symbol_id = 0000002193E970F90,
            name = 'a',
            type_code = integer,
            type_name = '\',
            modified = false,
            function_argument = true,
            escaped = false,
            scope_id = 0000002193E970E0,
          }
        },
      },
  },
};
```

Linear IR

- Looks more like traditional IR with basic blocks

define Proc1
L0 (entry)
  CLOSURE {Proc%2} {T(0)}
  RET {T(0)} {L1}
L1 (exit)
define Proc%2
L0 (entry)
  TOINT {local(a, 0)}
  ADDii {local(a, 0), local(a, 0)} {Tint(1)}
  MOVi {Tint(1)} {Tint(0)}
  MULii {Tint(0), 10 Kint(0)} {Tint(1)}
  RET {Tint(1)} {L1}
L1 (exit)
Embedding C in Ravi code

```c
C__decl [[
    typedef struct {
        unsigned char d;
        unsigned char m;
        short y;
        int  serial;
    } Date;
]]

DateFunctions = {}

function DateFunctions.make_date(d: integer, m: integer, y: integer)
    local date = C__new('Date', 2)

    C__unsafe(date, d, m, y) [[
        Date *dateptr = (Date *) date.ptr;
        dateptr->d = (unsigned char)d;
        dateptr->m = (unsigned char)m;
        dateptr->y = (short)y;
        y -= m <= 2;
        int era = (y >= 0 ? y : y - 399) / 400;
        unsigned yoe = (unsigned)(y - era + 480); // [0, 399]
        unsigned doy = ((153 * (m + (m > 2 ? -3 : 9)) + 2) / 5 + d - 1); // [0, 365]
        unsigned doe = yoe + 365 + yoe / 4 - yoe / 100 + yoe / 400; // [0, 146096]
        dateptr->serial = era * 146097 + (int)doy - 719468 + 25569; // +25569 adjusts the serial number to match Excel
    ]]

    return date
end
```
Embedding C in Ravi code

```ravel
function DateFunctions.print_date(date: any)
    local d: integer
    local m: integer
    local y: integer
    local j: integer
    C_unsafe(date, d, m, y, j) [[
        Date *dateptr = (Date *) date.ptr;
        d = dateptr->d;
        m = dateptr->m;
        y = dateptr->y;
        j = dateptr->serial;
    ]]
    print(d,m,y,j)
end

function DateFunctions.get_day(date: any)
    local v: integer
    C_unsafe(date, v) [[
        Date *dateptr = (Date *) date.ptr;
        v = dateptr->d;
    ]]
    return v
end
```
Embedding C in Ravi code

• Since we generate C code as intermediary – why not allow user to embed C
• How to do it safely?
• Integrate an embedded full C parser
• Enforce certain restrictions
  • Function calls not allowed
  • Pointers in structs not allowed
Optimizing code

• New parser and IR was designed so that we can write optimization passes
• However not much has been done so far
• Experimental feature – detect and replace constant upvalues with their values
Removing constant upvalues

```plaintext
define Proc1
L0 (entry)
    MOVi {1 Kint(0)} {local[a, 0]}
    CLOSURE {Proc2} {T(0)}
    RET {T(0)} {L1}
L1 (exit)
define Proc2
L0 (entry)
    RET {Upval(0, Proc1, a)} {L1}
L1 (exit)
```
Future directions

• Immediate goal is to stabilize the existing features for a production quality release, along with a small set of batteries (Suravi)

• I would like to add type inference but it’s a hard problem

• An alternative approach would be to detect types at runtime and specialize functions based on input types
Challenges

• It is a personal project in spare time, so hard to make progress quickly
Links

- https://github.com/dibyendumajumdar/ravi-compiler
- https://github.com/dibyendumajumdar/ravi
- https://github.com/dibyendumajumdar/Suravi