Pontifícia Universidade Católica
do Rio de Janeiro

What about Pallene?

Roberto Ierusalimschy

Lua Workshop 2022

Scripting and Performance

“If it's slow, rewrite it in C”

Easier said than done...

- Data mismatch
- Language mismatch

Data Mismatch

- Data has to be transferred between the two languages.
- Data has to be converted between the two languages.
- This process can kill any gains in performance due to a faster language.

Language Mismatch

- Big differences between Lua and C.
- It can be expensive to convert code from Lua to C.
- It can be hard to predict whether it is worth converting.

What about JITs?

- Don't change the language.
- Can achieve quite good performance.
- Hard to implement, port, and maintain.
- Optimization killers.

Programmers can go to great lengths to appease a JIT. At what point does it become a good idea to switch to a typed language?

Pallene as Scripting

- Pallene has been designed to act as a system counter–part of Lua in a scripting architecture.
- To reduce language mismatch, it is a typed subset of Lua.
- To reduce data mismatch, it operates directly on Lua data.

Pallene as Gradual Typing

- Pallene is a typed subset of Lua.

```
local function addqueen (N:integer, a:{integer}, i:integer)
  if i > N then
    printsolution(N, a)
  else
    for c = 1, N do
      if isplaceok(a, i, c) then
        a[i] = c
        addqueen(N, a, i + 1)
      end
    end
  end
end
```

Pallene as Gradual Typing?

- Types allow a simple ahead–of–time (AoT) compilation approach.
- The subset includes only what is simple to type and can generate efficient code.
  - no metamethods, variadic functions, coroutines
  - tables only in *tamed* forms (e.g., arrays and records)

Pallene as Gradual Typing?

Pallene:

```
local function id (a:{float}) : {float}
  return a
end
```

Lua:

```
local a = id({})              -- Ok
local a = id({3.5, 4.3})      -- Ok
local a = id({nil, 4.0})      -- ?
local a = id(3.5)             -- ?
local a = id({“x”, “y”})      -- ?
```

Data Representation

- Reuses Lua's data structures
- Reuses Lua's garbage collector
- Direct access to Lua data. Code bypasses the C API, as the compiler ensures correctness

How Can Types Help?

- Performance
- Simplicity

Types and Performance

- tag-checking is cheap
- boxing/unboxing is expensive
- Types guide box/unbox and allows local values to be manipulated unboxed

Types and Simplicity

- No need to deduce types
- No need to recover from wrong assumptions
  - (that also helps performance; there is only the fast path.)

Some Design Principles

- Same selling points from Lua.
- Simple AOT compiler.
- Very simple type system.
- Good on the borders.
- Gradual guarantee.

Selling points

Portable, Small size, Simple, Emphasis on scripting

Simple AOT compiler

Generates C code that can be loaded by the standard Lua interpreter, as a C module.

Keeps the same standards of portability as Lua itself.

Simplifies the implementation.

Simple type system

The primary goal of the type system is to help the compiler, in particular to allow *unboxing*!

Everything else can be handled with any, the dynamic type. Or, better yet, kept in Lua.

Good on the borders

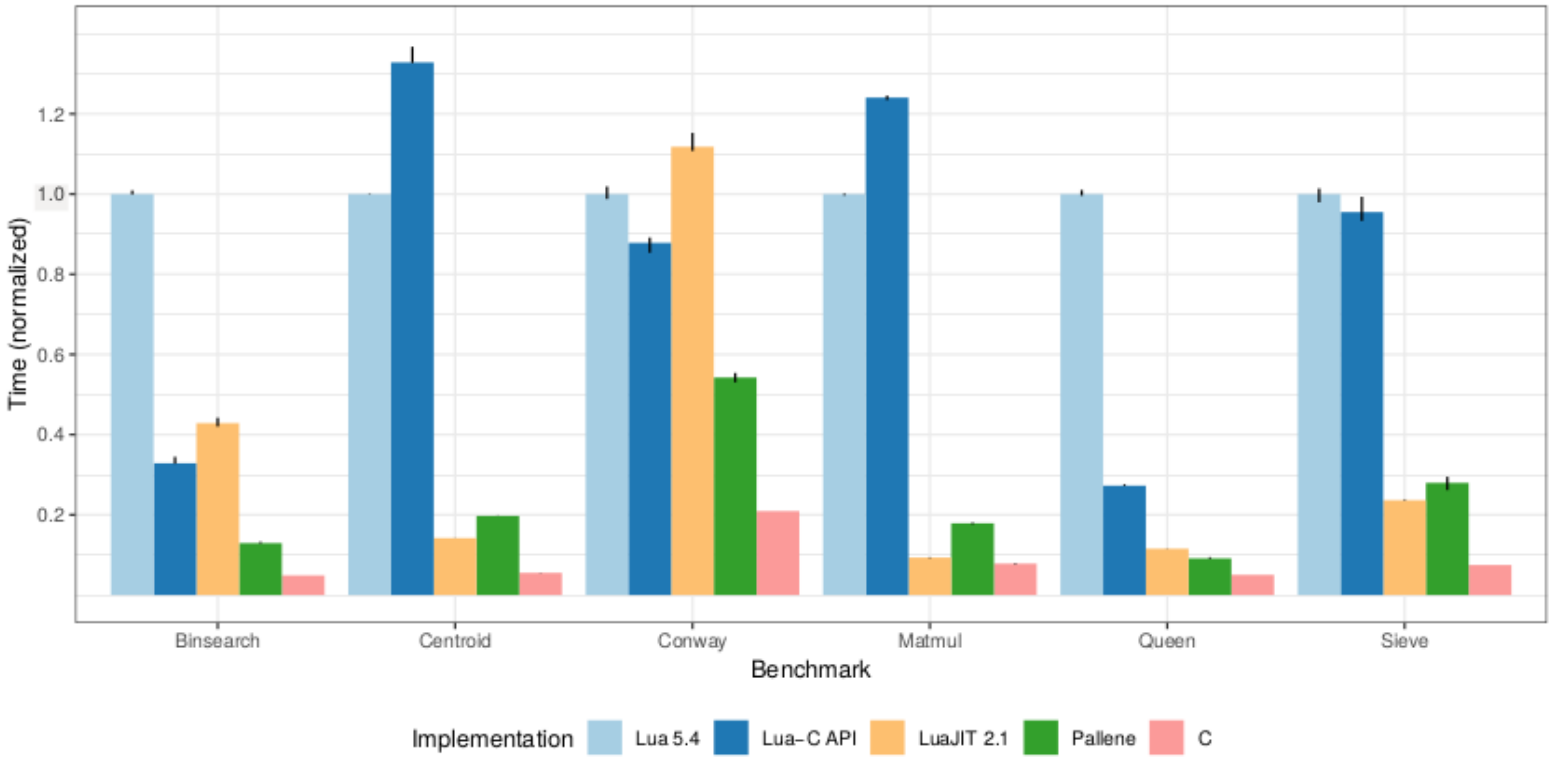
Real programs are seldom fully translated. Only the performance–critical parts need optimizations.

The change of one single function from Lua to Pallene should not worsen the performance.

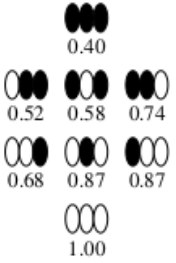
Gradual Guarantee

Pallene functions should have the same semantics of their translation to Lua (by removing type annotations), except for type errors.

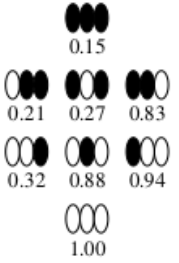
Hard to ensure to the letter in a real language like Lua. Simple solution is to remove features from Pallene, which makes it less expressive.


Time (normalized)
1.2
1.0
0.8
0.6
0.4
0.2
Binsearch
Centroid
Conway
Matmul
Queen
Sieve
Benchmark
Implementation
Lua 5.4
Lua−C API
LuaJIT 2.1
Pallene
C

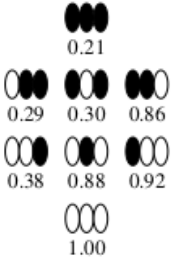
31



(a) Spectral Norm



(b) Queens



(c) Nbody

(d) Stream Sieve

Conclusions

- Performance is an always–present concern for dynamic languages.
- A companion language is an approach for improving the performance of Lua that seems compatible with its selling points.