Why I Love Lua

Lua is very fast

I took the Münchhausen-Algorithm from Munchausen numbers and tested it against Python, PyPy, Lua, Luajit, NodeJS, Java and C, with the little change not to search to 5k but to 50m. I took the comparable non-optimized algorithms.

22.29s C 23.29s Luajit 26.33s PyPy 54.30s Java 92.94s NodeJS 159.93s Lua 416.55s Python

So, I think we can say Lua is fucking fast for a script language. You might argue that PyPy is comparable, but PyPy really sucks big balls if I switch the algorithm and it is not comparable on the Benchmarksgame or anywhere. So, yes.

Lua is incredible fast and often is called the fastest script level High level language language on the market. It usually is not this tight to C, this is, I guess, because here are happening a lot of complex floating point operations and not just the byte crunching operations of the language itself.

But the algorithm was short and easy and I could access some comparable algorithms for this, that I can refer to on a single site, so, if you don't like it, make your own. But it is matching my usual perception of Lua performance.

Lua is a very fast language. Even in non-JIT it does even come near the JIT compiling NodeJS and it is just factor 3 behind Java, which is also JIT compiling and even not a script language anymore. I just wanted to put it here to show that Java is not a fast language.

Why is Lua so fast?

Well, calling Luajit yields this message:

> luajit

LuaJIT 2.0.5 – Copyright (C) 2005-2017 Mike Pall. http://luajit.org/

JIT: ON CMOV SSE2 SSE3 SSE4.1 AMD fold cse dce fwd dse narrow loop abc sink fuse

As you can see, this language has one of the best JIT compilers of all languages you may find, with supporting chipset features up to SSE4.1, folding, loop optimizing and many more features, that you mostly will miss on the other JIT engines.

Performance: x86/x64

Because Lua was the language, which invented this and the rest copied it, but they did bad. Why? Because their programmers are bad? No. It's because Lua was designed from the very beginning as a very simple and easy to embed and implement language with just a few features, that she manages to combine in a genius way, so that you get all what you get from other High level language, but at much lower cost. The idea of C cast into a High level language.

Lua only knows one abstract datatype: the table.

And it makes everything to a kind of table and the way they are doing it, is genius, well, not really because if you are an Assembly programmer you know that for the machine everything is a table and it ends up being a table anyway. The CPU loves tables, likes them, optimizes around them on hardware level and you really do good to organize your data in tables.

Which is exactly what Lua does.

It makes the thing much easier for Lua that there are just 21 language atoms, means original language tokens, that are reserved words. This means you can implement this language just with very few code. Few code means tight code, which means the stuff remains in the CPU cache much better than of larger languages and this you can see on it's performance.

Additional to that, most High level language languages have a stack design, like a Keller automate, while Lua is organized as a register machine, which is corresponding to the actual processor design of our hardware. This, too, gives a huge advantage.

And additionally to this, Lua has a unique and genius FFI interface to directly call libC or any other library of the Linux environment, additional to be able to define very easily it's own libraries, classes if you want or any other data structures in C and makes it easy for C to manipulate them, work with them and give the results back.

If you want the full power of Lua, use the libraries, that are done in C or ASM. The easy and available implementation level for Lua is one of it's strengths that keeps it's code compact and very powerful. Indeed the more you add machine-code-level libraries the faster the language gets, until you can't see any difference to pure C anymore. Which does not mean that Lua beats C! It embraces it and it is so fast, because it embraces C. This is ***teamwork!*** And it is so exceptional, because most High level language want to be "the best, the cleanest, the language with maximum features and syntactic sugar". Lua just wants to give you everything any of the other High level language can and it gives you this power for cheap. Really cheap. She's not a diva, she's just that cheap and easy brat that is fun to be around.

While most other High level language rely on modules that are written only in this language and any other way is highly discouraged (Python, Java, NodeJS, …), Lua embraces the work with native libraries and also provides a package installer system named Luarocks, that can provide these libraries to the community and compiles on all target systems I tested without much problems.

To jump directly into the powerful code of C or C++ is something other High level language can't do, can't do this easily and without sweat.

Lua eats just 3k of code if you embed it into your application. It just uses 120 kb for the normal core in form of it's system library or 400k in form of it's JIT system library, if you chose so. Size to speed, that's always the deal.

In comparison, there is just no library this small, Python just eats 15 mb for the calling programs alone, the system library horde is so huge, that I got sick while trying to count, it must be hundreds of MB all together. And PyPy is even on top of that, that's typical for languages like that, NodeJS starts with the caller alone at 12 mb and I won't start with Java, okay? Hope you have a swap-space installed.

Lua doesn't compete with C. You can't replace C with Lua. But you can boost C applications through Lua and yield all that High level language power that you might want for flexible configuration and change of dataflow.

As a script language it does absolutely fantastic in your system, it is well behaving and well playing with the other kids, it can call foreign library functions, is fast, has a very well controllable garbage collector (one of the good/bad things about High level language, that can break your application in the worst case), but you don't have to break your head about that.

It is fully controllable and it will usually not fuck up your app. You can of course always evade any hassle by moving some implementation level of your Lua algorithm into a C library, where it will run perfectly.

All this said, I think, there are some points about why Lua is so fast. And why it will remain fast and why it is one of the fastest High level language around. And why the whole concept of it does good, because it respects it's limits, it leaves you open a door to the cellar, to leave High level language whenever you need or whenever you want or to call other parts of the system, reach out and loot some Python code or call Perl or shell or system commands, pipelines or network functions, down to calling the GPU with OpenGL or OpenCL or whatever you like.

You will never beat smart C code with Lua, but that's not the purpose. It doesn't fight C like all the others, it embraces C. And if you don't write C yourself, you can easily find someone who does this for you and shares his library, like all the other libraries that are already out there.

It is not the best High level language language and it doesn't try to be. It is the cheapest High level language language. So you get the maximum bang for the buck with Lua. It was designed for use in small embedded systems like Nintendo DS. And it was very successful there. You can find almost every algorithm you might want already coded in Lua on pages like Rosettacode or other algorithm collections.

They might not be always the best possible, but they are good enough. Lua does well in the Benchmarksgame, it always did well and they even didn't test Luajit for that, which is the actual APEX of Lua.

And available for all big platforms.

it is the simplicity of the language that makes it fast. There is also LuaJit that many people confuse for Lua when talking about speed. LuaJit is a branch of Lua 5.1 syntax that includes one of the most advanced JIT optimizers ever made. Because of Lua's simplicity LuaJit can optimize code to some really unbelievably tight assembly routines and knock your socks off.

More complex languages have amazing JIT optimizers, but at this point in time I haven't seen anything that is quite as good as LuaJit's. I don't think it is impossible for any language to achieve these speeds, but it may take the developers increasingly more time as language complexity increases.

Lus is a very simple language. One of the main differences between Lua and other well known interpreted languages like Python or Ruby is that Lua doesn't offer the same level of introspection, and doesn't need the same amount of indirection, including need to check for hooks, that those languages require. Another difference is that Lua was never proposed as general use language; its underlying VM support fewer primitives, making it simpler, lighter and faster.