Julia Programming Language

Arsalan Macknojia

July 15th, 2020

Introduction

Numerous developers and scientists have the conviction that when it comes to programming languages, conventional low-level, pre-compiled languages, for example, C and C++ are in every case quicker, more powerful, and distinctly efficient. Julia, a moderately new, Just-in-Time compiled language, refutes these articulations and proves that these convictions are not valid in every case.

Julia's development was started in 2009 at MIT, with the assistance of Lead engineers Alan Edelman, Jeff Bezanson, Stefan Karpinski, and Viral Shah. They wanted to create an open-source, high-level language with the speed of C, dynamism of Ruby, and convenience of Python. Something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab and as good at gluing programs together as the shell. A language that is exceptionally simple to adapt yet powerful enough to support complex numerical and scientific computation. [1]

Out of this vision, after 3 years of research and development, Julia was released in 2012. As of July 2020, Julia has released version 1.4.2 and reached a state of “fully baked”. Ever since its initial release, Julia has been picking up heaps of footing and it is currently ranked as the 36th most popular programming language as per TIOBE index [2]. Although the adoption of Julia has been incredible, it is still far from begin considered as a mainstream programming language.

Julia Uses

Julia was originally built from ground up to tackle machine learning, data mining, and statistical computation. A significant portion of use cases for Julia incorporates climate modelling, autonomous vehicles, artificial intelligence, finance, and bioinformatics.

Over the years, Julia has managed to pull in some prominent clients, ranging from investment manager BlackRock, which utilizes it for time-series analytics, to the British insurer Aviva, which utilizes it for risk estimations. In 2015, the Federal Reserve Bank of New York utilized Julia to make models of the United States economy, noting that Julia made model estimation was almost 10 times faster than its past MATLAB implementation.

Furthermore, Julia is used by the Climate Modelling Alliance as the sole execution language for its cutting-edge worldwide atmosphere model. It is also used by NASA; and Brazilian comparable (INPE) for space mission planning/satellite simulation. [3]

Julia Strengths

1. **Fast execution time**
   Julia was designed with performance in mind without compromising the ease of use. Julia's JIT compilation and type-stability through specialization (multiple-dispatch), leads to clear performance gains. It makes the code concise and enables the compiler to turn it into efficient machine instructions. Furthermore, Julia comes with built-in features to accelerate computationally intensive problems, such as distributed computing. This allows efficient use of available computational resources to reduce execution time.

2. **Julia solves the two-language problem**
   Two-language problem is a trade-off that developers normally make while picking a programming language. Language can either be moderately simple for developers to write, or relatively fast for computers to run, yet not both. Developers often prototype algorithms in a user-friendly language, like Python, and later rewrite it in a faster language, like C. Especially those dealing with computationally intensive problems. It's a laborious and error-prone process.

   Julia circumvents this problem by being both a compiled and dynamically typed language. It runs like C but reads like Python. Coding in Julia appears identical to Python, type a line, and get a result. However, in the background, the code is compiled which results in fast execution time.

3. **Julia support scientific syntax**
   Julia excels at numerical and statistical computing. It supports Greek characters, subscripts, and special maths symbols which make formulas looks identical to regular scientific equations. This allows researchers, quants, and scientists to do calculations in a very natural manner and protect against arithmetic mistakes.

```
julia> x = 0.75π
2.356194490192345

julia> 2x
4.71238898038469

julia> 2cos(x)
-1.414213562373095
```

4. **Julia comes with superior parallelism**
   To perform scientific computing efficiently it is important to make full use of the available resources. To achieve this Julia provides built-in primitives for parallel computing at every level. Julia compiler is also capable of generating native code for various hardware accelerators, such as GPUs.

Julia Weaknesses

1. **Limited package ecosystem**
   Julia's ecosystem is relatively new which means finding the right package can be a challenge. Julia currently has upwards of 3,000 packages, including, machine learning, DNA sequence analysis, computational simulations, and mathematical modeling. In comparison, R has more than 10,000 packages, and Python exceeds 130,000 packages. However, Julia does support importing libraries written in other languages such as C and Python in an efficient way. This mitigates the limited package ecosystem problem to a certain extent.

2. **Small developers' community & limited industry adoption**
   Julia is still a new language with a small community of developers and limited industry adoption which means it can be difficult to find answers online. This may not be the case a few years from now since there has been an exponential increase in Julia's popularity in recent years.

Other Interesting Features

Julia offers many interesting features such as macros, multiple dispatch, excellent support for distributed computing, coroutines (green threading), high-order functions, superior package manager, a fantastic REPL, and numerous other features. Following are some of the prominent features of Julia.

1. **Macros**
   Macro is an alternate way to access functions. It maps a sequence of arguments to a returned expression which is substituted directly into the program where the macro is invoked. In Julia, users can define a macro by simply using the "macro" keyword. Users can then call the macro by adding @ symbol before the name of the macro. Macro arguments may include symbols, literal values, and expressions.

```
julia> macro discriminant(a, b, c) return (b^2 - 4a*c) end
@discriminant (macro with 1 method)

julia> @discriminant 4 10 6
4
```

2. **Multiple Dispatch**
   Multiple Dispatch is a prominent Julia feature which invokes different functions based on function's arguments. Julia allows users to create multiple functions to handle different arguments under the same method name using parametric polymorphism. Multiple dispatch is particularly useful for mathematical functions, where arguments can be of many different types.

```
julia> function _int(x) println("Handle Integer!") end
julia> function _float(x) println("Handle Float!") end

julia> multiple_dispatch(x::Int64) = _int(x)
julia> multiple_dispatch(x::Float64) = _float(x)

julia> multiple_dispatch(70)
Handle Integer!

julia> multiple_dispatch(.07)
Handle Float!
```

3. **Parallel Computing**
   Julia comes with staggering support for multi-threading and parallel computing right out of the box. It offers multiple levels of parallelism including coroutines (green threading), multi-threading, and distributed processing. Julia natively implements interfaces to split a process across different cores. In addition, the compiler is also capable of generating native code for various hardware accelerators.

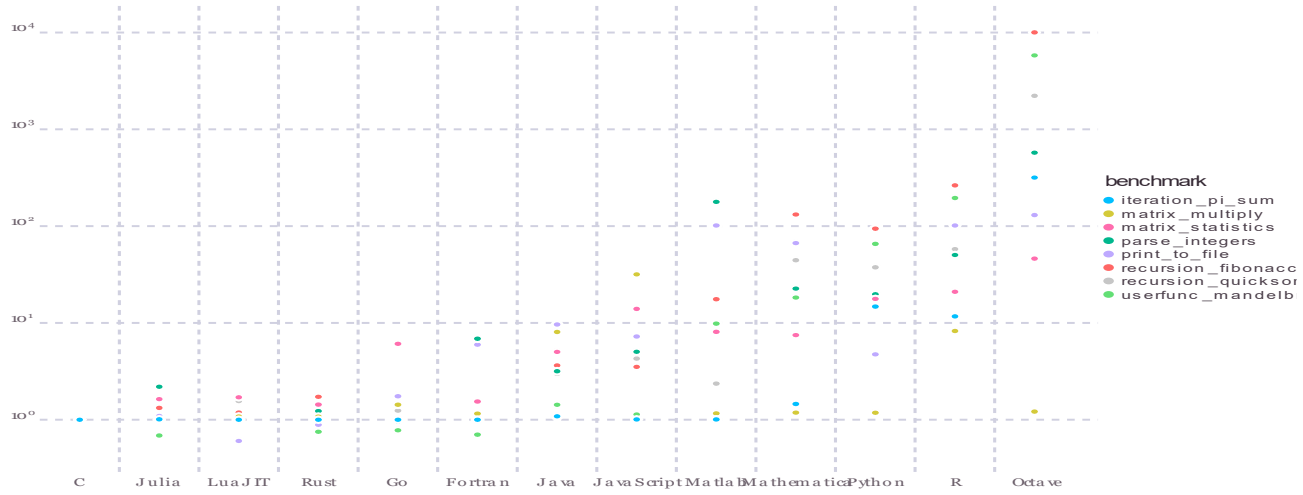
Comparison with Other Languages

Julia was designed to combine the interactivity of high-level languages, like Python with the speed of compiled languages, like C. In comparison, Julia has several advantages and numerous differences.

C is a low-level, statically typed, compiled language that makes it very powerful, but it comes with very complex syntax. To master C, users must understand the low-level implementation of the program such as memory management, pointers, etc. Without a firm understanding of these concepts, users struggle to develop an efficient and performance-oriented application. In contrast, Julia is a dynamically typed, compiled language that makes it flexible and just as fast as C. It abstracts out low-level details and offers a significantly simple syntax to write code. Furthermore, it comes with an automatic memory management system (garbage collector) which eliminates the need to manually manage memory.

Python is the third most popular language as of July 2020 based on the TIOBE index. [2] It is extremely popular among data scientists and machine learning professionals and is extensively used for Artificial Intelligence. Its a dynamically typed, high-level, interpreted language, which is known for its simplicity and flexibility, but it lacks execution speed. Julia, in contrast, is both easy to write and fast to execute. It is magnitudes faster than Python and provides excellent support for big data analytics by performing complex tasks such as cloud computing and parallelism. Unlike Python, Julia is a new language with a significantly smaller community of developers, resulting in limited packages. However, it does provide good support to use external libraries written in other programming languages.

Following is the benchmark result taken from the official Julia website. These micro-benchmarks are not comprehensive, but it does compare Julia's performance on a range of problems with other programming languages. [4]



Conclusion

To conclude, Julia has shown a lot of potential to be the language of the future. Its technical superiority makes it easy to learn, flexible to write, and blazing fast to execute. Julia is currently the 36th most popular language which is a huge milestone given that it's a relatively new language. Despite having small developers' community and limited packages ecosystem, it has managed to pick up the consideration of significant organizations and enterprises, such as NASA, and Federal Reserve Bank of New York. [2] It is still far from being considered a mainstream language but the exponential increase in its popularity is a clear indication that Julia has met creators' vision, of being a fast, flexible, and easy to use programing language which addresses multiple programming paradigms.

References

1. Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012, February 12). Why We Created Julia [Web log post]. Retrieved July 20, 2020, from https://julialang.org/blog/2012/02/why-we-created-julia/
2. Index, T. (2020, July 04). Latest news. Retrieved July 21, 2020, from https://www.tiobe.com/tiobe-index/
3. Conservation through Coding: 5 Questions with Viral Shah [Web log interview]. (2019, March 04). Retrieved July 20, 2020, from https://science.nasa.gov/earth-science/applied-sciences/making-space-for-earth/5-questions-with-viral-shah
4. Jeff Bezanson, S. (n.d.). The Julia Language. Retrieved July 21, 2020, from https://julialang.org/benchmarks/