Julia

2010s proposition for scientific (and other) programming

Kamil Ziemian
kziemianfvt@gmail.com

Seminar of Field Theory Department
23 November 2018

Background information

Basics

- Julia has started as project in 2009, first release 0.1 in 2012, version 1.0 8 August 2018. Current version (20 November 2018) 1.0.2.
- It is free, open software project.
- Created by Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah, most of them work at MIT at some stages of their life.
- They aim was to created modern language suited to numerical computing and scientific use.
- Today one of main center of development of the language is Julia Lab at MIT, with Alan Edelman as it current head.
- They co-founded Julia Computing *to develop products that make Julia easy to use, easy to deploy and easy to scale*.

Why not learn Julia?

Reasons not to learn

- You are happy with tools you have now.
- You don't find libraries you need.
- You don't have time/want learn new programming language.
- You don't want to learn new style of programming.
- Since release of **Julia v1.0**, which clean language a lot, ecosystem of external packages still need some adjustments.
- There is not enough good learning materials for version 1.x.
- Still not enough user base.
- Ecosystem (packages, IDEs, debugger,...) is not as mature as for other languages/enivroments.
- Low "stackoverflow effect": how likely answer for your question is under the top link in Google results.

Why and why not learning Julia?

More reasons to not learn

- Since Julia is more general purpose language than MATLAB, Mathematica, R, etc., makes finding some things a little bit harder.
- No serious project/company use it. Please wait few minutes.
- Non-trivial performance engineering. You must learn how to do that.

Reasons to learn

- Allows High productivity of writing code (compilers can make more for us, than 30 years ago).
- Allows good readability of written code (Python or MATLAB like).
- Most languages we used are really old created in 1990s, 1980s, 1970s, 1960s and 1950s. They were designed for very, very different computers, environments, to solve different problems and at different stage of computer science development.

Why learn Julia?

Reasons to learn

- If there is a mistake in MATLAB, you must live with that and walk around it. In Julia you can correct it yourself.
- You can write code closely to Python way with REPL (shell like thing) and "scripting" or in browser (like IPython Notebook).
- After you write your code, in most cases you can optimize it to, achieve 35%– 105% speed of FORTRAN or C (hopefully in future this gap will became narrower).
- Seamless incorporation of FORTRAN, C, C++, Python, R and Java.
- *Is Julia the next big programming language? MIT thinks so, as version 1.0 lands*, see full from TechRepublic article here. Updating *Pan Tadeusz*: What American invent, Pole will like.
- In Poland we often talking about modernization, so this is good thing to watch what happened at world level.
- New style of programming == new opportunities. You can to some extant write domain specific language.

Who use Julia?

Celeste.jl

Project written entirely in Julia at National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory (Berkeley Lab). Aim: analyzing data from 35% of visible sky gathered by Sloan Digital Sky Survey.

- Peak performance of 1.54 petaflops (10^{15} flops) using 1.3 million threads on 9,300 Knights Landing nodes of the Cori supercomputer at NERSC. They say that only assembler, C, C++ and FORTRAN achieved previously over 1 petaflops performance.
- Loaded approximately 178 terabytes of image data and give parameters estimates for 188 millions stars and galaxies in 14.6 minutes.

It is worth noting that this computation was performed before 2018, so it used 0.x version of the language (probably 0.4 an 0.5), which has unstable syntax.

Who use Julia?

Big players that use or give money to support it

NASA, Intel, IBM, Google, Microsoft, Amazon, Apple, Alibaba.com, Ford, Facebook, Oracle. And few more.

List of institutions, to give some scope

MIT Robot Locomotion Group, USA Federal Aviation Administration, Federal Reserve Bank of New York, The Brazilian National Institute for Space Research, AOT Energy (energy trading), PSR (global electricity and natural gas consulting, analytic and technology firm), University of Auckland Electric Power Optimization Centre (The Milk Output Optimizer), Voxel8 (3D printing), University of Copenhagen, Peking University and Imperial College London collaboration on *An Anthropocene Map of Genetic Diversity* (Science, vol 343, issue 6307, pp. 1532-1535), Path BioAnalytics (medicine).

See for more cases and details here link and here link.

There are lies, big lies and benchmarks

Compression of different languages

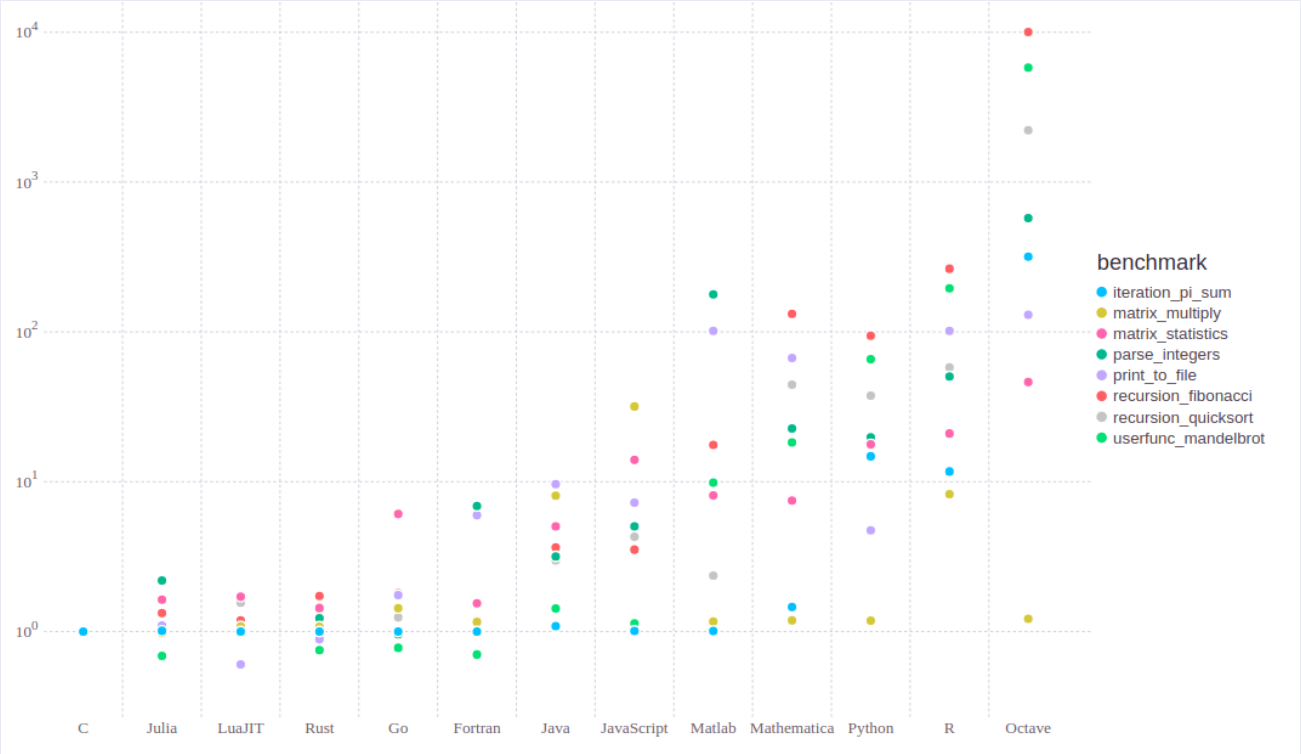


Figure: From page *Julia Micro-Benchmarks*, https://julialang.org/benchmarks/.

Warning! Python code use numpy libraries that is written 52.9% in C (state of GitHub repository at 23 November 2018).

Solving numerical PDEs, how good Julia can be?

Example: Kuramoto-Sivashinsky equation in $1+1$ dimensions

Kuramoto-Sivashinsky equation is nonlinear (more precisely: semilinear) partial differential equation, which in $1+1$ dimension takes form

$$\frac{\partial u(t,x)}{\partial t} + \frac{\partial^4 u(t,x)}{\partial x^4} + \frac{\partial^2 u(t,x)}{\partial x^2} + u(t,x)\frac{\partial u(t,x)}{\partial x} = 0. \tag{1}$$

Full $1+3$ dimensional version of this equation was proposed to *model the diffusion instabilities in a laminar flame front*.

There are lies, big lies and benchmarks

This benchmarks were crated by John F. Gibson, Dept. Mathematics and Statistics, University of New Hampshire, main author of **Channelflow**: *a set of high-level software tools and libraries for research in turbulence in channel geometries* written in C++, link. Full list of people that contribute to benchmarks is here.

Solving numerical PDEs, how good Julia can be?

They are outdated

They are from middle 2017, so they using Julia 0.5 (?) and as such they are outdated. For this reason I don't present any code, only results. John F. Gibson said that he will update them to Julia 1.x, but they were not ready for this presentation (or I don't notice update in last week).

If you want to see code yourself

All code in all languages used for creating this benchmarks is available on GitHub johnfgibson/julia-pde-benchmark.
Warning! Python code use numpy libraries that is written 52.9% in C (state of GitHub repository at 23 November 2018).

Solving numerical PDEs, how good Julia can be?

Algorithm

The KS-CNAB2 benchmark algorithm is a simple numerical integration scheme for the KS [Kuramoto-Sivashinsky] equation that uses Fourier expansion in space [All codes call this same Fourier Transform external library.], collocation calculation of the nonlinear term $u(t,x)u_x(t,x)$, and finite-differencing in time, specifically 2nd-order Crank-Nicolson Adams-Bashforth (CNAB2) timestepping.

More details on algorithms can be found here.

Result

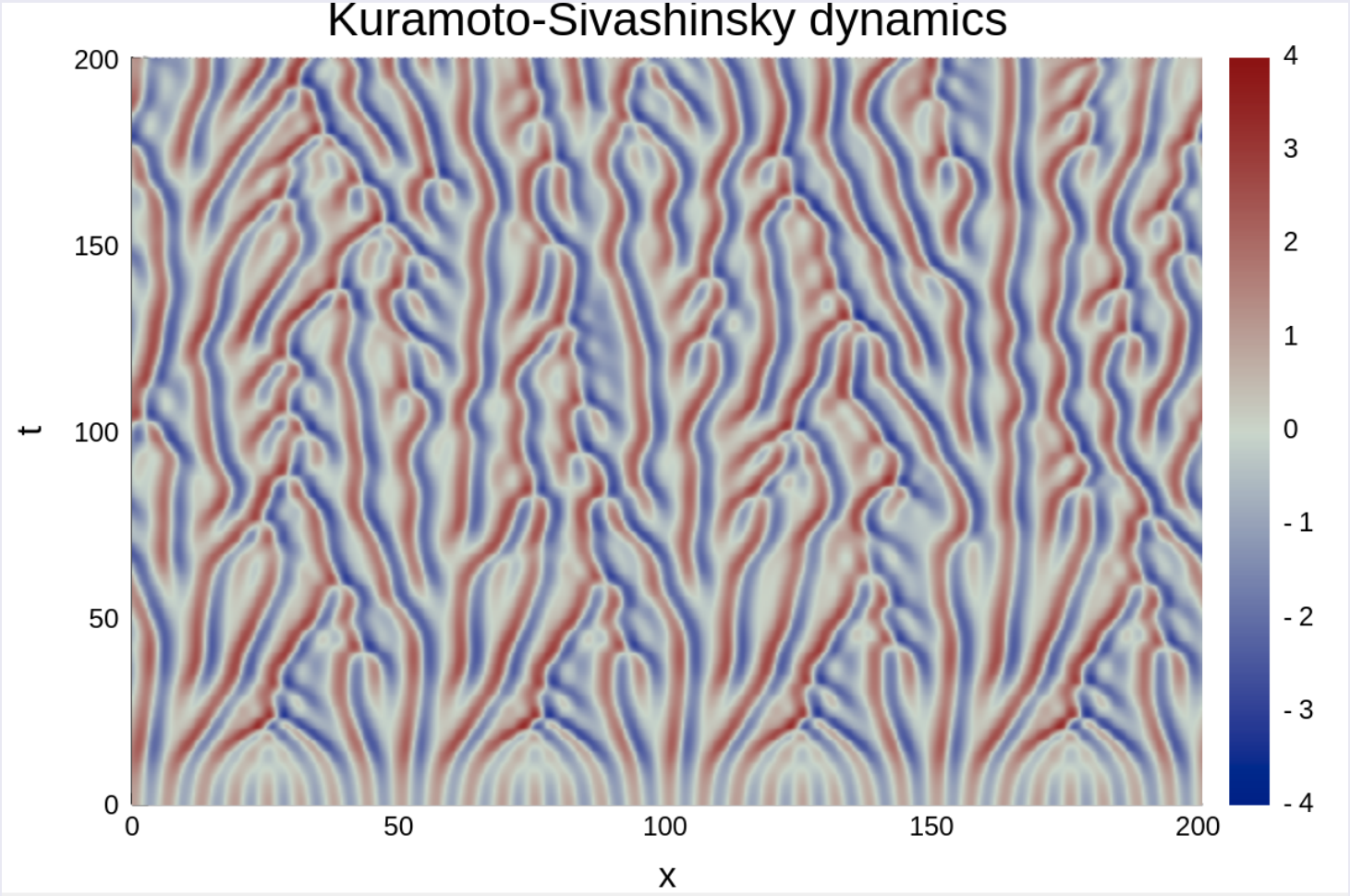


Figure: Kuramoto-Sivashinsky heat evolution in 1 dimension.

Chart with linear scale

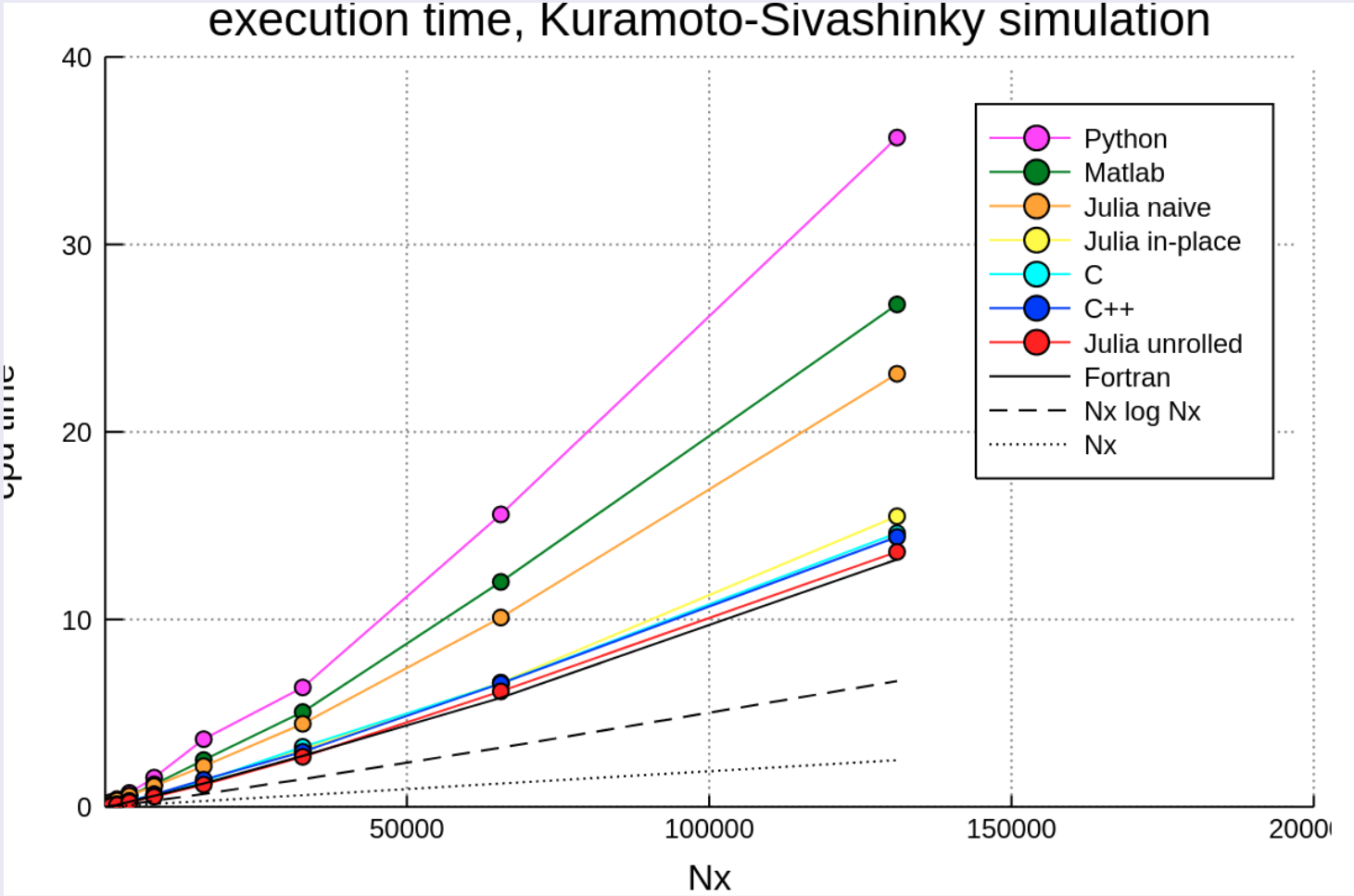


Figure: Results for N_x points on x axis, CPU time in seconds.

Chart with logarithmic scale

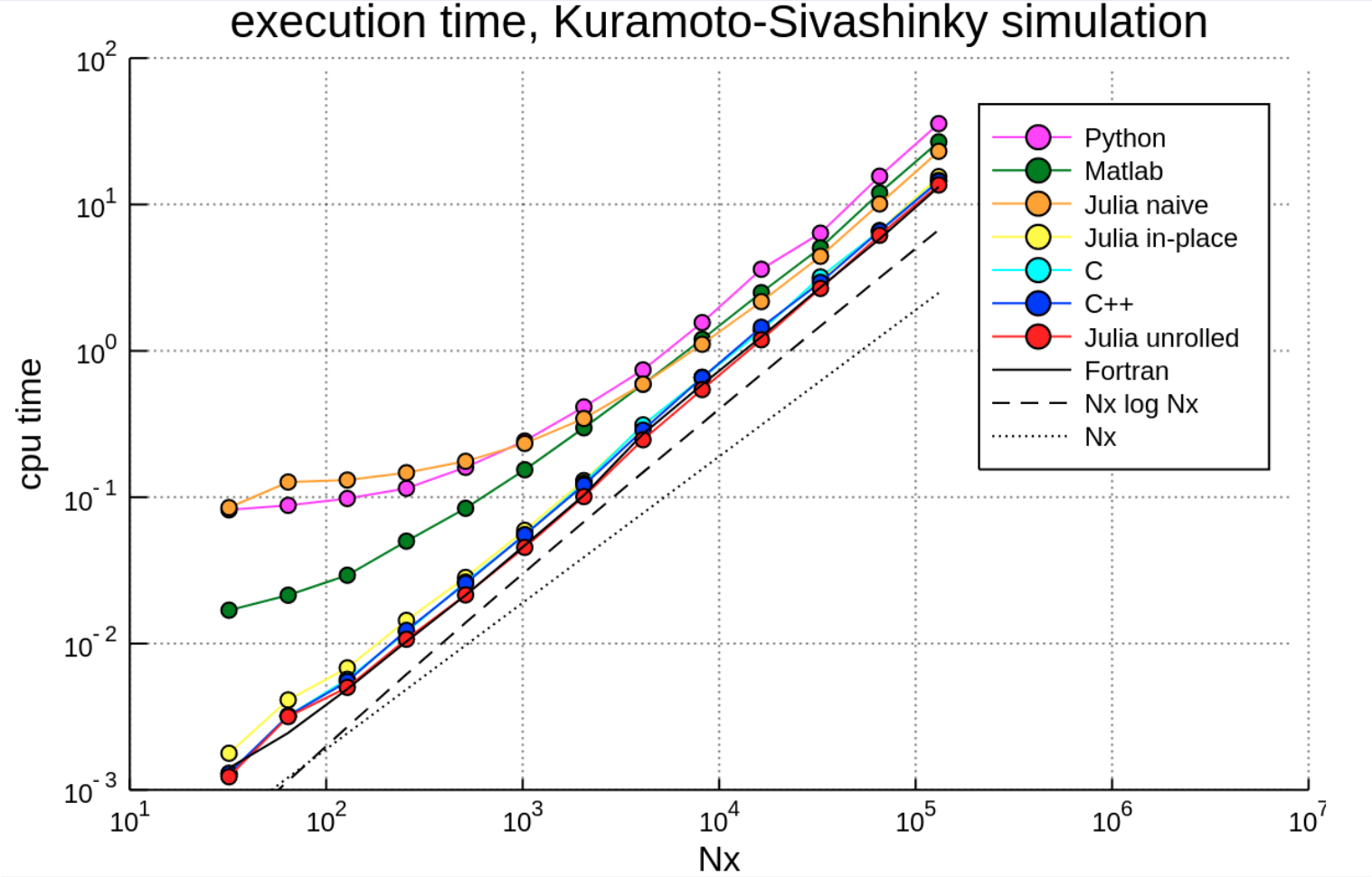


Figure: Results for N_x points on x axis, CPU time in seconds.

Solving numerical PDEs, how good Julia can be?

Time for maximal $N_x = 2^{17} = 131072$

Language	CPU time [s]	Ratio to C
Fortran	13.0	0.90
Julia unrolled	13.6	0.93
C++	14.4	0.99
C	14.6	1.00
Julia in-place	15.5	1.06
Julia naive	23.1	1.58
MATLAB	26.8	1.83
Python	35.7	2.45

Table: From the fastest to the slowest code.

Time vs lines of code

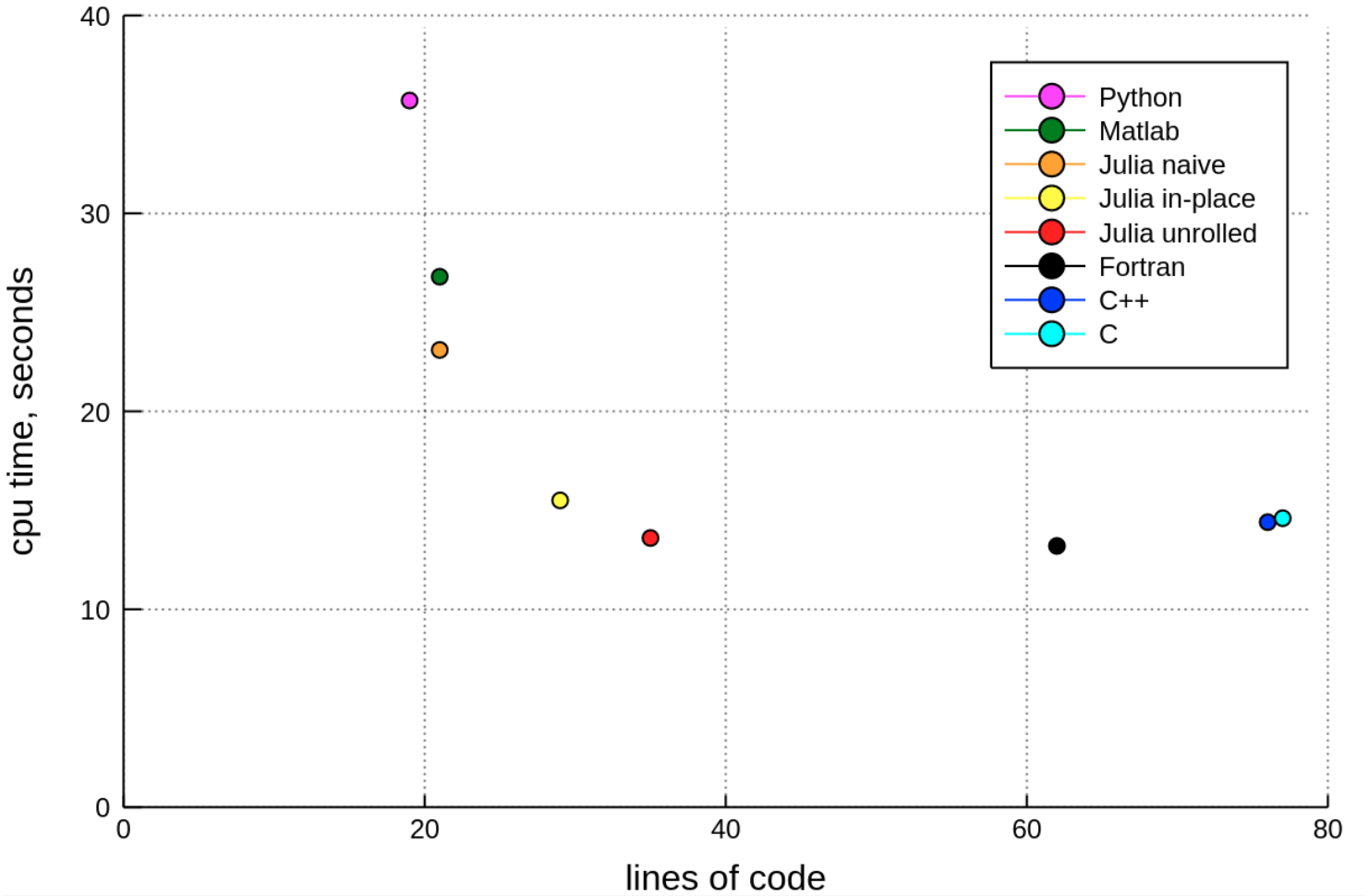


Figure: Compression of time and lines of code.

Speed/lines of code

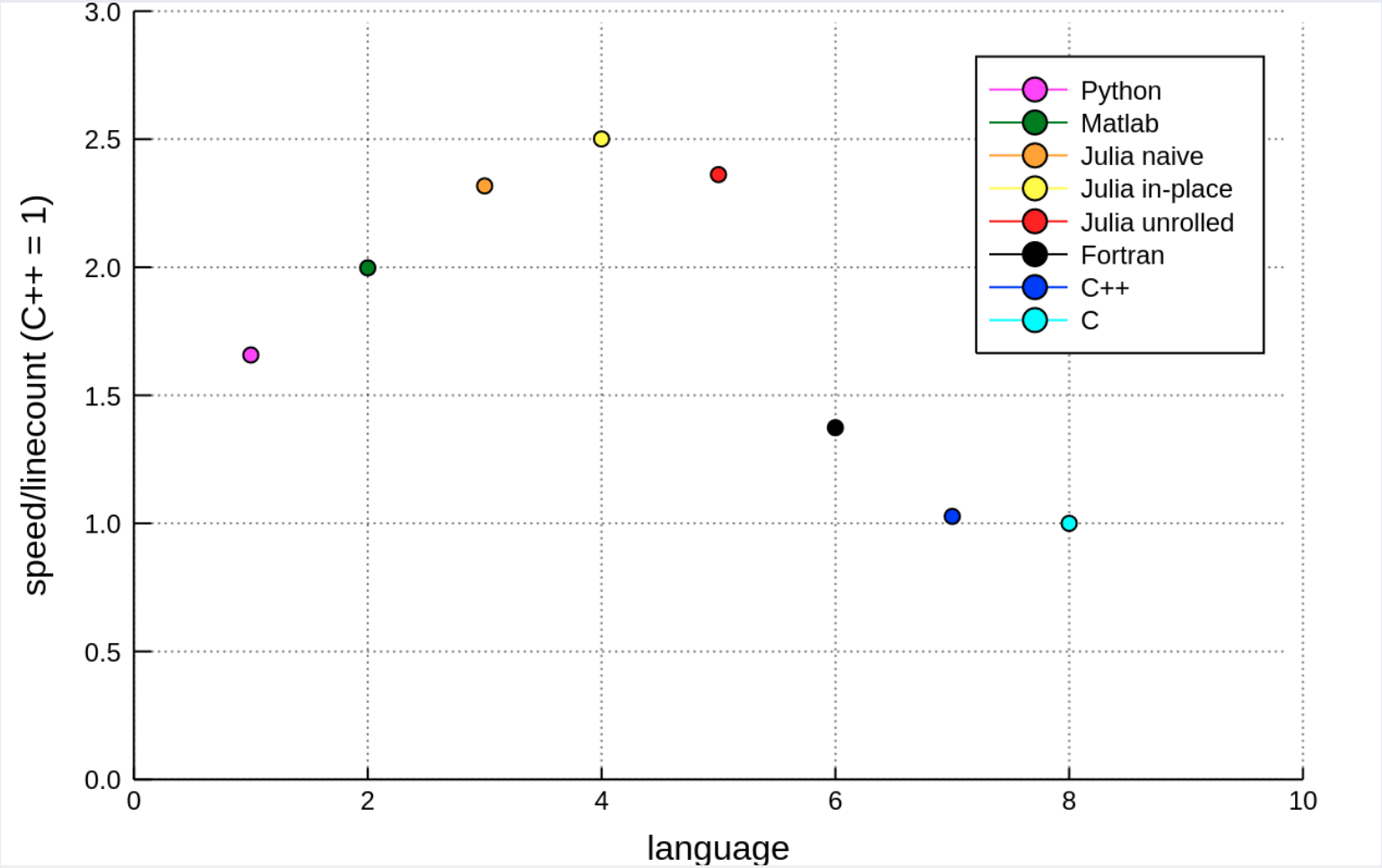


Figure: Compression of ratio of speed to the lines of code.

Few interesting things for which there is no time here

Packages and projects

- DynamicalSystems.jl, GitHub JuliaDynamics/DynamicalSystems.jl. Winner of one of three main prizes of The Dynamical System Web in 2018.
- QuantumOptics.jl. Numerical framework for numerical solving open quantum systems, more on this page https://qojulia.org/.
- Machine Learning on Google's Cloud TPUs (tensor processing units). *Targeting TPUs using our compiler, we are able to evaluate the VGG19 forward pass on a batch of 100 images in 0.23s which compares favorably to the 52.4s required for the original model on the CPU. Our implementation is less than 1000 lines of Julia, with no TPU specific changes made to the core Julia compiler or any other Julia packages.* Whole article by Keno Fischer and Elliot Saba here.

Few interesting things for which there is no time here

Things that are important or promising

- Type system, link.
- Metaprogramming and how use it to make code efficient, link.
- Multiple dispatch as rare way of Object Oriented Programming (I don't want holy war over this statement), link.
- Using GPU, link.
- Julia package manager, link.
- Symbolic computing like in Wolfram Mathematica, link.
- Monte Carlo, very immature, link.
- Writing fast code, link
- Parallel computing, link.
- Regulars expressions, link.
- Tensor compilers, link.
- How Julia work inside, link.

Closing remarks

Quite good summary

Julia is still not great when you want import some libraries and use them. It is great when you must write new library. Chris Rackauckas on Julia Discourse.

My reflections

- Julia proved that it can be used to make serious scientific work in various fields, including numerical computations relevant to field theory.
- Pedagogical perspective. Learning and using it is comparable to Python (in my opinion), which is starting language for computer scientists on MIT and at the same time is more deep as computer language and have more computational power.
- Have potential to make money outside academic realm.

Closing remarks

If you use it in your research

They ask you for citing paper **Julia: A Fresh Approach to Numerical Computing**, Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah (2017) SIAM Review, 59: 65–98. doi: 10.1137/141000671, url: http://julialang.org/publications/julia-fresh-approach-BEKS.pdf, link. And add you paper to the following list https://julialang.org/publications/, link.

I want to thank these people for help and discusions

- Tamas Papp,
- John F. Gibson,
- Yakir Luc Gagnon,
- Antoine Levitt,
- Krzysztof Musiał.

With most of them I converse on Julia Discourse, link.

Thank you.

Bibliography

Relevant articles

- Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah, *Julia: A Fresh Approach to Numerical Computing*, (2017) SIAM Review, 59: 65–98. doi: 10.1137/141000671, url: http://julialang.org/publications/julia-fresh-approach-BEKS.pdf, link.
- Chistopher Rackauckas and Qing Nie, *DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia*. Journal of Open Research Software (2017). 5(1), p.15. doi: http://doi.org/10.5334/jors.151, link.
- Keno Fischer, Elliot Saba, *Automatic Full Compilation of Julia Programs and ML Models to Cloud TPUs*, arXiv:1810.09868.
- Nick Heath, *Is Julia the next big programming language? MIT thinks so, as version 1.0 lands*. TechRepublic, August 29, 2018, link.

Sources used to make this presentation

- John F. Gibson, *julia-pde-benchmark*, GitHub johnfgibson/julia-pde-benchmark.
- John F. Gibson, *Why Julia*, GitHub johnfgibson/whyjulia.

Learning materials

Basics materials

Most of them are more or less outdated, since they were created before release of Julia 1.x.

- JuliaBoxTutorials, version 1.x, GitHub JuliaComputing/JuliaBoxTutorials. Good starting point.
- Julia 1.x Documentation. Always up to date, really good written in comparison to others manuals, https://docs.julialang.org/en/v1/.
- David P. Sanders, *Introduction to Julia for scientific Computing (Workshop)*, 2015. Outdated, but very good introduction to language, YouTube.
- The Julia Language channel on YouTube. Contains dozens videos from JuliaCons and hold regulars *Intro to Julia*, keeping it up to date. You can find next *Intro to Julia* and other introductions on Facebook.
- Ben Lauwens, Allen Downey, *Think Julia: How to Think Like a Computer Scientist*, link.

Learning materials

Practical introductions to many different topics

JuliaCon is annual conference on Julia language which start at 2014.

- Stefan Karpinski and Kristoffer Carlsson, *Pkg3: The new Julia package manager*, JuliaCon 2018, YouTube.
- Arch D. Robison, *Introduction to Writing High Performance Julia (Workshop)*, JuliaCon 2016, YouTube.
- Andy Ferris, *A practical introduction to metaprogramming in Julia*, JuliaCon 2018, YouTube.
- Chris Rackauckas, *Intro to solving differential equations in Julia*, YouTube.
- DiffEqTutorials.jl. Tutorials of JuliaDiffEq project, GitHub JuliaDiffEq/DiffEqTutorials.jl.

Learning materials

More theoretical, less practical materials

- Jeff Bezanson, *Why is Julia fast?*, 2015, YouTube.
- Jeff Bezanson, *The State of the Type System*, JuliaCon 2017, YouTube.
- Jiahao Chen, *Why language matters: Julia and multiple dispatch*, 2016, YouTube.
- Jameson Nash, *AoT or JIT: How Does Julia Work?*, (AoT – Ahead of Time compilation, JIT – Just In Time compilation), JuliaCon 2017, YouTube.
- John Lapyre, *Symbolic Mathematics in Julia*, JuliaCon 2018, YouTube.
- Julia Lab at MIT, *Parallel Computing (Workshop)*, JuliaCon 2016, YouTube.
- Tim Besard, Valentin Churavy and Simon Danisch, *GPU Programming with Julia*, JuliaCon 2017, YouTube.

Learning materials

More theoretical, less practical materials

- Peter Ahrens, *For Loops 2.0: Index Notation And The Future Of Tensor Compilers*, JuliaCon 2018, YouTube.
- Carsten Bauer, *Julia for Physics: Quantum Monte Carlo*, JuliaCon 2018, YouTube.
- George Datseris, *Why Julia is the most suitable language for science*, case study of project JuliaDynamics, link.
- Nick Higham, *Tricks and Tips in Numerical Computing*, JuliaCon 2018, YouTube.

Mentioned projects and articles

Written Julia if not mentioned otherwise

Still many of them don't work in Julia 1.x.

- Channelflow (written in C++), http://channelflow.org/.
- DynamicalSystems.jl, GitHub JuliaDynamics/DynamicalSystems.jl.
- QuantumOptics.jl. Numerical framework for numerical solving open quantum systems, more on this page https://qojulia.org/.
- REPL (shell) for C++ (broken in Julia 1.x), GitHub Keno/Cxx.jl.
- Game and educational tool *Paddle Battle*, GitHub NHDaly/PaddleBattleJL.

Additional information and topics

Current state of Language

Some statistics

- Google Scholar: 607 papers about Julia or using it in research (state at 20 November 2018). See also topic *Research* at Julia Language page https://julialang.org/publications/, link.
- Used in some way in over 1,000 universities.
- Ecosystem of over 1,900 packages for wide are of topics.
- Over 2 millions downloads.
- Over 41,000 GitHub stars for language and packages.
- 101% annual growth (based on downloads).

Outside first positions rest all others are taken from Julia Computing and dated from August 2018.

Current state of language

Julia source code

Language	Percent of code
Julia	68.2%
C	16.7%
C++	10.1%
Scheme	3.4%
Makefile	0.6%
LLVM	0.3%
Other	0.7%

Table: Numbers from GitHub JuliaLang/julia, 20 November 2018.

Scheme – Lisp dialect from MIT, LLVM – Low Level Virtual Machine.

Cxx.jl (broken in 1.x)

RELP for C++

The Julia C++ Foreign Function Interface (FFI) and REPL.

```
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: http://docs.julialang.org
   _ _   _| |_  __ _   |  Type "?help" for help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.5.0-dev+3487 (2016-04-10 22:55 UTC)
 _/ |\__'_|_|_|\__'_|  |  Commit 9ee9aef* (1 day old master)
|__/                   |  x86_64-pc-linux-gnu

julia> using Cxx

C++ > // Press '<' to activate C++ mode

C++ > #include <iostream>
true

C++ > std::cout << "Welcome to Cxx.jl" << std::endl;
Welcome to Cxx.jl

C++ > std::string cxx = "C++";

julia> println("Combine Julia and ", bytestring(icxx"cxx;"))
Combine Julia and C++

julia>
```

Figure: Compression of ratio of speed to the lines of code.

Application in Julia

Mac App Store Preview

Open the Mac App Store to buy and download apps.

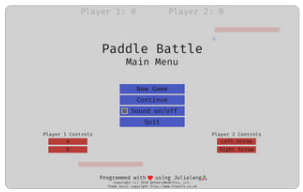
Paddle Battle 4+

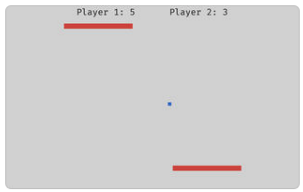
nhdalyMadeThis, LLC

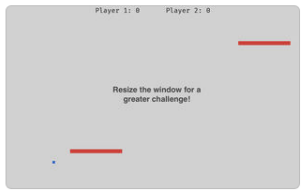
★★★★★ 5.0, 6 Ratings

Free

Screenshots







Go head-to-head with a friend in this modern take on a classic.

Play Paddle Battle as two friends sitting at one keyboard, just like the old flash games.

In addition to being a fun, easy arcade game, Paddle Battle is also an educational tool! All the code for the game is available online, open source, here:
http://github.com/nhdaly/PaddleBattleJL
You can read that code to learn how it was made, and make a game like it yourself!

Corollary

One man write game in pure Julia (with some shell scripts, to automatize building up binaries) and put it on Mac App Store, to show if you want, you can do it quite easily. Code of game is available on here.

Mentioned projects and articles

Written Julia if not mentioned otherwise

Still many of them don't work in Julia 1.x.

- Channelflow (written in C++), http://channelflow.org/.
- DynamicalSystems.jl, GitHub JuliaDynamics/DynamicalSystems.jl.
- QuantumOptics.jl. Numerical framework for numerical solving open quantum systems, more on this page https://qojulia.org/.
- REPL (shell) for C++ (broken in Julia 1.x), GitHub Keno/Cxx.jl.
- Game and educational tool *Paddle Battle*, GitHub NHDaly/PaddleBattleJL.