Intro to Julia Language

Control, Optimization, and Networks Lab

The University of Texas at Dallas

Sleiman Safaoui

December 16, 2019

Topics

- What is Julia
- Why Julia?
  - Speed Test
- How to Use Julia
- Useful Libraries
- Resources

What is JuliaLang

- Fast
- General
- Dynamic
- Easy-to-use
- Optionally typed
- Open source

Why Julia?

- Dynamic Environment Problem
  - People love them
  - Usually compromise speed

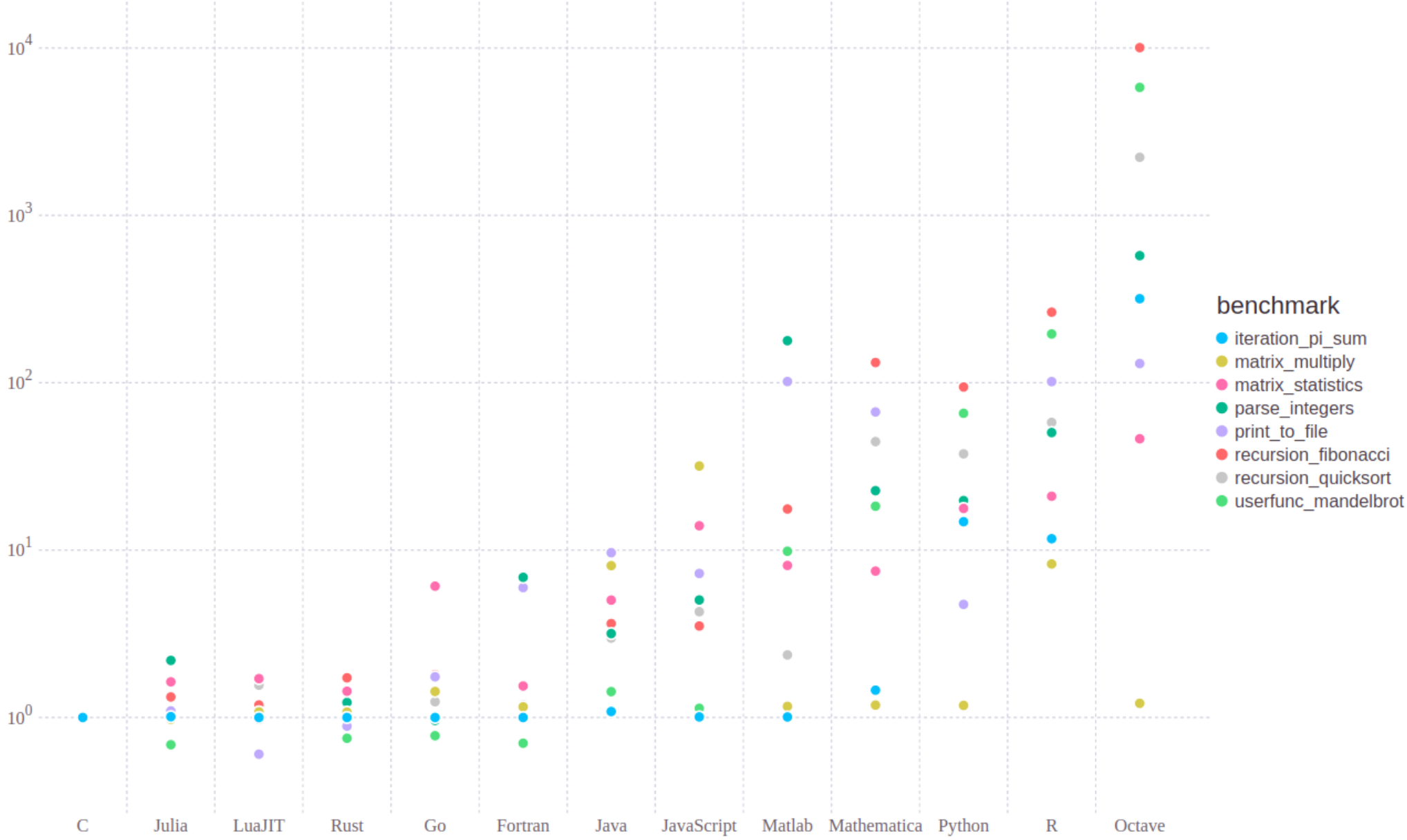
- **Programming Language Compromise:**
  - *High-level dynamic*
  - *Low-level fast*

Why Julia?

- **Easy** to learn and use, **Fast**, and **Dynamic**
- Can define equally fast and compact data types as native ones
- Can use unicode character (\gamma, \delta, \Eta, …) with tab completion
- Designed for numerical and scientific computing
- FREE
- Bonus I: Good documentation
- Bonus II: Growing community
- Bonus III: Many tutorials

Speed Test

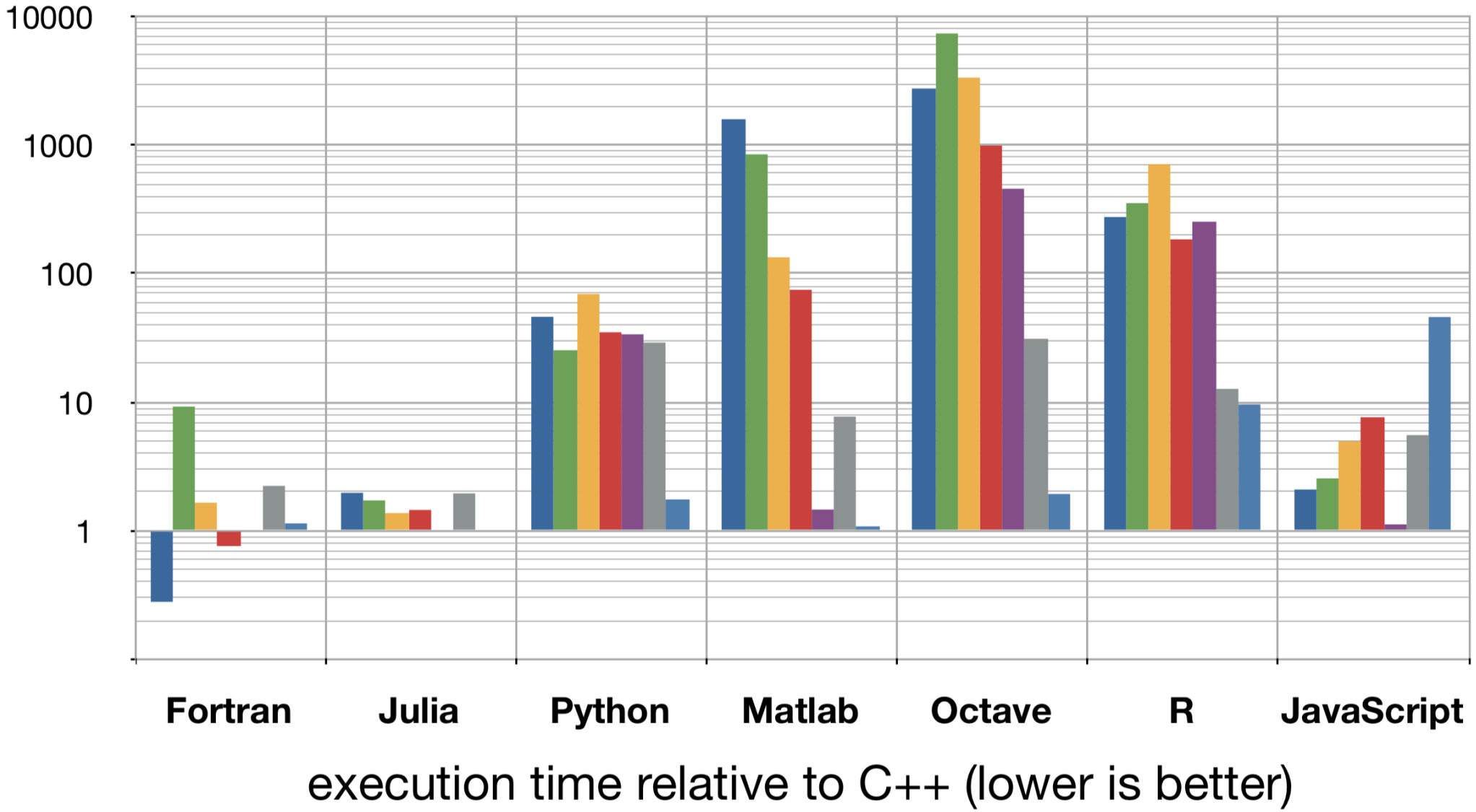
- Julia benchmark tests: https://julialang.org/benchmarks/



Speed Test

- MIT:

https://archive.is/o/ja8BQ/https://github.com/JuliaLang/julia-



Speed Test

- Hackernoon merge sort: https://hackernoon.com/performance-analysis-julia-python-c-dd09f03282a3

Speed Test

- Our speed test ..
  - **Not comprehensive**
  - **Very specific**
  - But *interesting results*
- Two main test:
  - Matrix Inverse
    - Load a native “optimized” function (except in c++)
  - Nested For Loops
    - For loops are basic
    - Simple problem
    - Single threaded

Speed Test

- Matrix Inverse:
  - Random 5000x5000
  - C++: using eigen
  - Python (numpy), Julia, Matlab: native inv function
  - Time only the inverse function
- Nested For Loops
  - Each loop counts up to 10000
  - Add index to a variable at every iteration
  - For loop and addition “+” are native, data types defined when needed

Speed Test

- Results
  - Matrix Inverse

Program	Threads	Avrg Time (sec)	Time for 10 experiments (sec)	Avrg*threads
C++ w/ -O3	1	24.27	242.7	24.27
C++ w/ Ofast	1	25.14	251.4	25.14
Julia (script)	**8**	**2.52**	25.2	20.16
Julia (function)	**8**	**2.48**	24.8	19.84
Matlab (script)	4	1.8	17.99	7.2
Matlab (function)	4	1.79	17.95	7.16
Python (script)	4	2.09	20.99	8.36
Python (function in script)	4	2.15	21.5	8.6
Python (fuction out of script	4	2.09	20.99	8.36

Speed Test

- Results
  - Nested For Loops

Program	Threads	Avrg Time (sec)	Time for 10 experiments (sec)
C++	1	0.22	2.2
C++ w/ -O3	1	2e-6	2e-5
C++ w/ Ofast	1	2e-6	2e-5
Julia (script)	1	**5.66**	56.6
Julia (function)	1	**3.38e-8**	3.38e-7
Matlab (script)	1	0.11	1.1
Matlab (function)	1	0.11	1.1
Python (script)	1	6.16	61.6
Python (function in script)	1	3.25	32.5
Python (function out of script)	1	3.47	34.7

Speed Test

- Results Takeaway:
  - Julia demonstrates higher level of native parallelization (8 threads vs 4) in native functions
  - Julia demonstrates speed in native functions (for, +)
  - Julia and Python are faster when function are used (as opposed to scripts)

How to Use

- Install Julia
- Write/Run code:
  - Julia REPL
  - Atom Julia Client + Juno, Install
  - Jupyter Notebook

How to Use

- JuliaBox
  - Run Julia in browser
  - Uses Jupyter

Useful Libraries/Packages

- Visualization:
  - Plots.jl (plotting API)
  - PyPlot.jl (MatPlotLib.PyPlot based plotter)
- Machine Learning:
  - JuliaML
  - Automatic Differentiation
  - CUDA GPU Acceleration cuArrays
  - JuliaDB (to work with persistent data set – terabytes of data)

Useful Libraries/Packages

- Fourier transforms: AbstractFFTs.jl
- Image processing: JuliaImages
- NonLinear Dynamics: JuliaDynamics
- Biology, quantum physics, quantitative economics, ….
- Control: ControlSytems.jl:
  - LQR
  - PID
  - Advanced pole-zero placement
  - Stability boundary for PID controllers
  - PID plots
  - Transfer functions, state space system, analysis, time and frequency response

Useful Libraries/Packages

- Parallel Computing
  - Coroutines (Tasks which can start, be interrupted, and resumed without using space)
  - Multi-threading
  - Multi-core/Distributed Processing
    - Divided over different CPU cores or different machines

Installing Packages

- In the Julia REPL run:
  - using Pkg
  - Pkg.add("Package Name")
- Replace "Package Name” with the desired package’s name
  - E.g. for Plots package: Pkg.add(”Plots")

Resources

- https://julialang.org/learning/
- https://archive.is/ja8BQ