The N+1 language problem

Introducing Julia for HPC

•••

Valentin Churavy
JuliaLab, CSAIL, MIT
vchuravy@csail.mit.edu

Oh the irony...

> Python is *extensible*: if you know how to program in **C** it is easy to add a new built-in function or module to the interpreter, either to perform **critical operations at maximum speed**, or to link Python programs to libraries that may only be available in binary form (such as a vendor-specific graphics library).

(from: https://docs.python.org/3.5/tutorial/appetite.html)

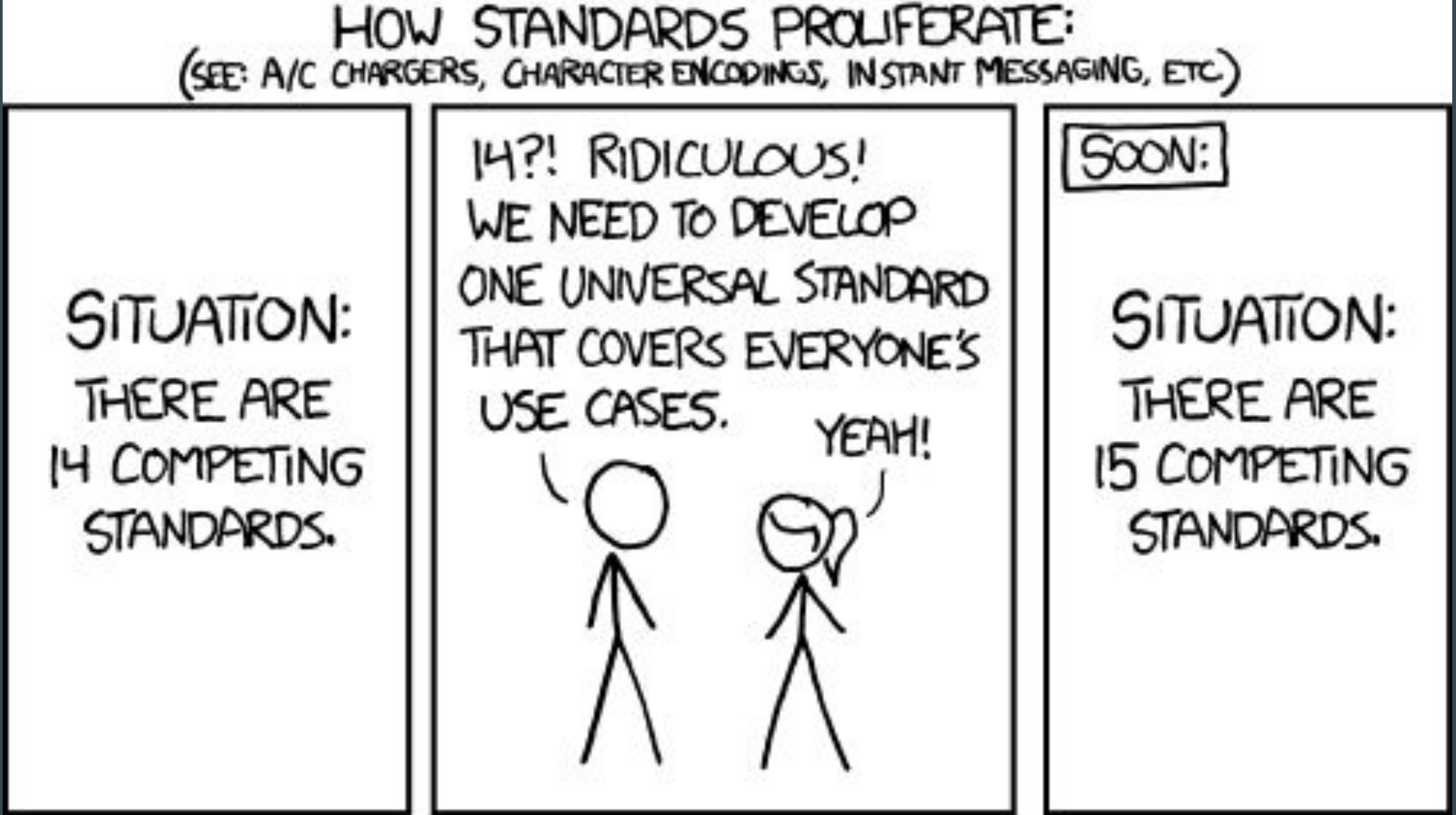
The two language problem

1. Frontend language (Python/R/Matlab/…) productivity above all else

2. Backend language (C/C++/Fortran/ASM/…) performance above all else

The N+1 language problem

1. At least one language for the domain experts
   a. Python
   b. Maybe R because plots and statics
2. One language for each hardware you are targeting
   a. Usually a C dialect


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Introducing Julia

Looks like ... Python

Feels like ... Lisp

Runs like ... Fortran

Get it at: https://julialang.org/

Or use https://juliabox.com/

One ring to rule them all,
One ring to find them,
One ring to bring them all and in the darkness bind them.

Feels like… Python

- High-level
- Quasi mathematical
- Combining Python/Matlab/R
- High productivity
- Modern software development

Feels like... Lisp

- Powerful metaprogramming
    - Macros
    - Generated functions
- Code as data
- Generic
- Multiple dispatch
- Compiler extensions
    - Contextual dispatch: Cassette.jl
    - CUDAnative.jl
    - I am working on compiler extensions for heterogeneous distributed computing

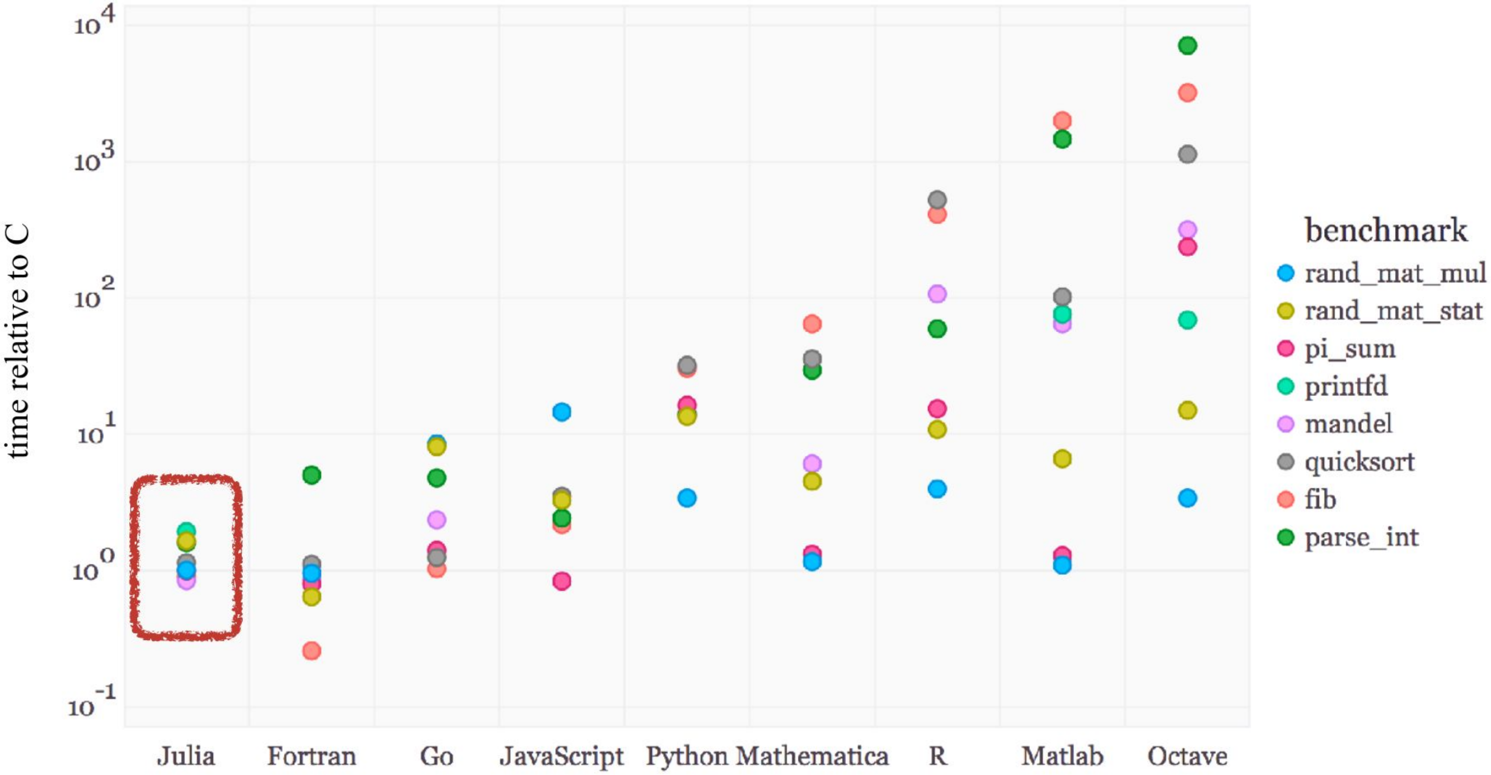
Runs like.. Fortran

- Fast
    - Often as fast as C and Fortran
    - If you write highly dynamic code: As slow as Python
    - I expect well written Julia code to be within 2x of C
- Numerics front and center
- Capable of using modern HW
    - GPUs
    - SIMD
- Build on-top of LLVM

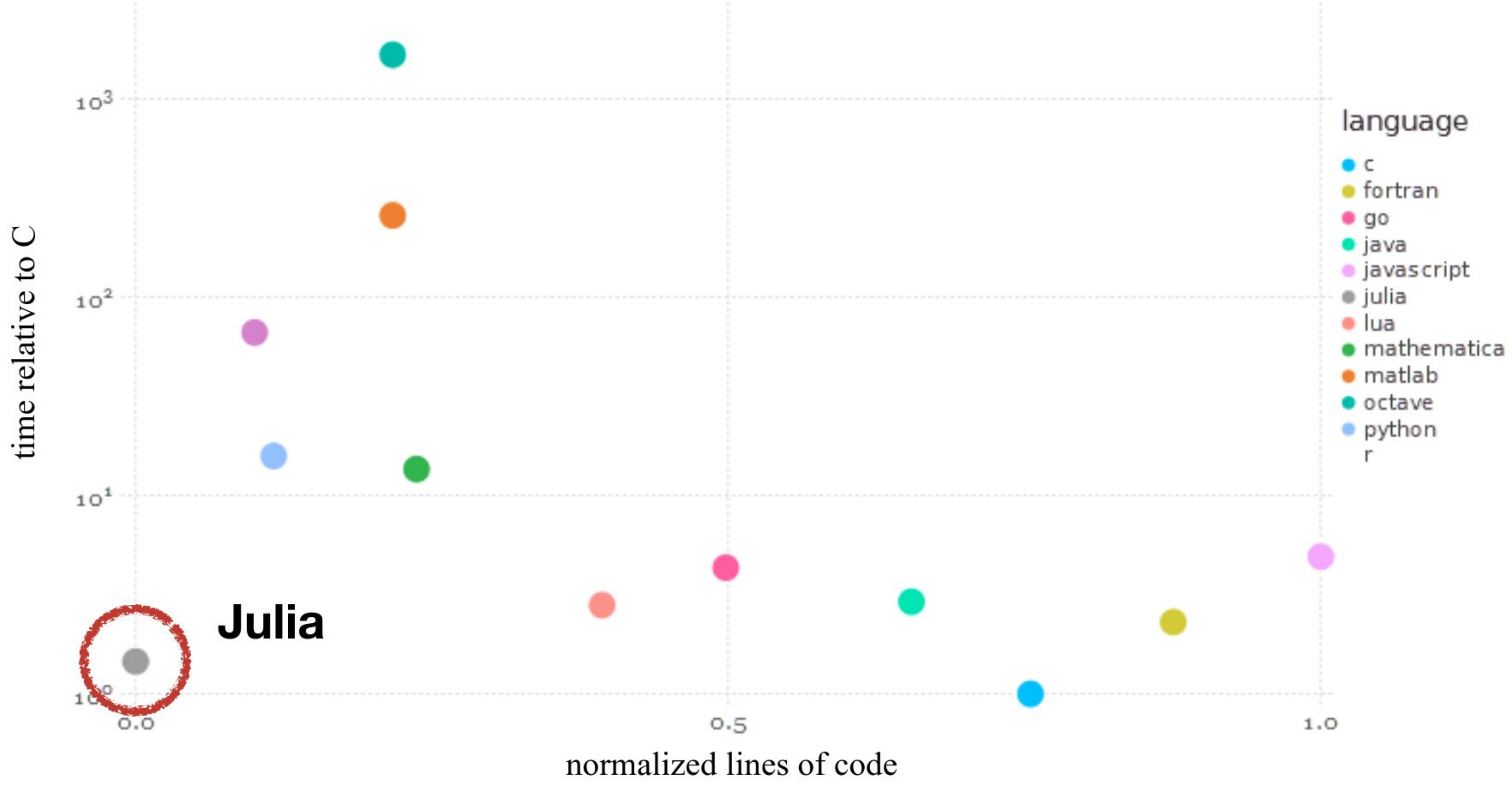
Squaring the Circle

- Greedy programmers
    - https://julialang.org/blog/2012/02/why-we-created-julia
- Efficient to write, efficient to run
- Support for accelerators
- Distributed programming

Speed


time relative to C
10^4
10^3
10^2
10^1
10^0
10^-1
Julia
Fortran
Go
JavaScript
Python
Mathematica
R
Matlab
Octave
benchmark
rand_mat_mul
rand_mat_stat
pi_sum
printfd
mandel
quicksort
fib
parse_int

Speed vs. Productivity



Celeste

1. Story: https://arxiv.org/pdf/1801.10277.pdf
2. Bayesian inference model running on Cori II, written in Julia
3. Sustained peaked at 1.56 DP Pflops
4. "As succinct as Python, as fast as C"
5. Uses StructOfArrays.jl and StaticArrays.jl
6. 67% runtime in Julia, 18% native dependencies and Julia runtime, 10% the system math library, 3% the Intel Math Kernel Library, and 2% in libc
7. Finding enough parallelism for KNL (in the end 82.3% of FLOPS operated on 8-wide AVX512 vector registers)
8. Compiler tuning — most changes are upstreamed

Demos

Resources

- Parallelism in Julia (as it is today)
  - https://github.com/stevengj/18S096/blob/master/lectures/lecture5/Parallelism.ipynb
  - https://slides.com/valentinchuravy/julia-parallelism
- https://docs.julialang.org
- https://github.com/JuliaLang/julia
- https://github.com/JuliaGPU
- https://github.com/JuliaParallel
- http://www.nersc.gov/users/data-analytics/data-analytics-2/julia/
- https://devblogs.nvidia.com/gpu-computing-julia-programming-language/