GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Anna Kahle

The Julia programming language

And its suitability for HPC

1 Introduction
- Introduction to Julia
- Simplicity and Efficiency

2 HPC in Julia
- Parallelization and GPU offload
- Benchmarks for CPU and GPU
- Examples for HPC applications

3 Popularity
- Statistics
- Downsides

4 Conclusion

Overview

1 Introduction
- Introduction to Julia
- Simplicity and Efficiency

2 HPC in Julia

3 Popularity

4 Conclusion

Introduction to Julia

- fairly new programming language
  - development started in 2009, released 2012
  - current version 1.7.2
- for HPC scientific computing and data analysis
- wants to solve the "two-language" problem
  - most programming languages either easy to write or fast
  - this leads to:
    - users program in high-level languages
    - rewrite the performance-critical parts in C/C++

Properties of Julia

- similar to read as Python and MATLAB
- free and open-source
- JIT compilation using LLVM
- multi-paradigm language:
  - object-oriented, functional, imperative
- easy to call C and Fortran code
- dynamically typed
- support for unicode

Sample Code: Sieve of Eratosthenes

```
function eratosthenes(max::Int64)
    # get an array of size max, every entry is a possible prime
    prim = fill(true, max)
    prim[1] = false
    for i in 2:isqrt(max)
        if prim[i]
            # mark every multiple of i that is not yet marked
            for j in i*i:i:max
                prim[j] = false
            end
        end
    end
    return prim
end
#take input from user
number = parse(Int64, Base.prompt("limit"))
prim = eratosthenes(number)
#print all indices of elements which are true
print(findall(prim))
```

Julia micro-benchmarks

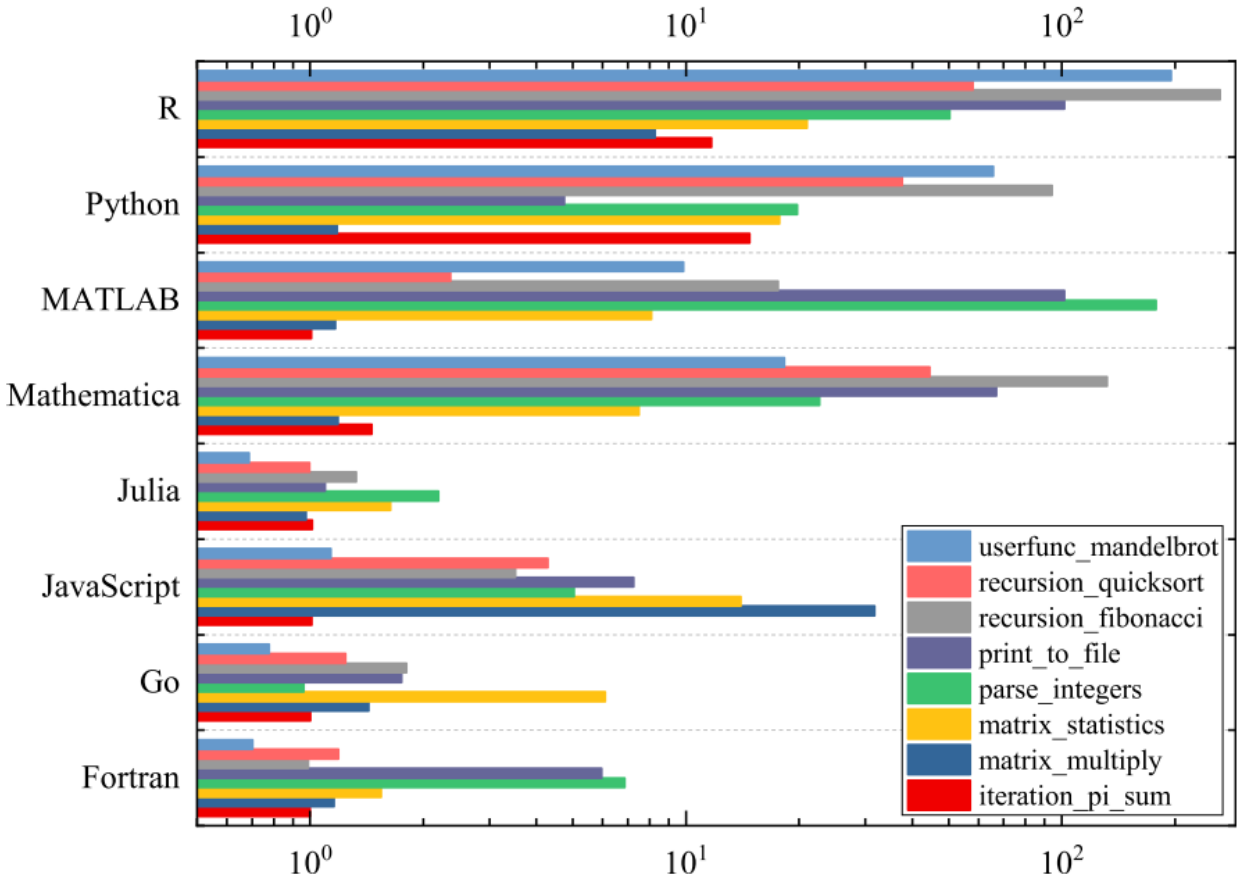

Figure: Julia language in machine learning[2]

Why is Julia so fast?

- an expressive parametric type system
  - optional type annotation
- multiple dispatch using type annotations
- dataflow type inference
- careful design of the language and standard library
- aggressive code specialization on run-time types
- just-in-time compilation (using LLVM)

Overview

1 Introduction

2 HPC in Julia
- Parallelization and GPU offload
- Benchmarks for CPU and GPU
- Examples for HPC applications

3 Popularity

4 Conclusion

Parallelization

three main types of parallel programming interesting for HPC:

- multi-threading:
  - contained in `Threads` standard library
- Julia wrapper for MPI
- Distributed computing:
  - main implementation of message passing for distributed-memory systems
  - contained in `Distributed` module as part of the standard library

GPU offload

- implements different packages for each vendor
  - CUDA (NVIDIA), oneAPI (Intel), ROCm (AMD)
  - wraps the native interfaces by respective offload solution
- going to another API: substitution of the API calls
- KernelAbstraction.jl (KA.jl)
  - can be executed on NVIDIA, AMD and CPU platforms
  - no source change needed
  - performance can be suboptimal

BabelStream

- implements the McCalpin STREAM benchmark
- uses five operations:
  - Copy: $A[i] = B[i]$
  - Mul: $A[i] = s \cdot B[i]$
  - Add: $A[i] = B[i] + C[i]$
  - Triad: $A[i] = B[i] + s \cdot C[i]$
  - Dot: $R = R + (A[i] \cdot B[i])$
- measures execution time for each kernel
- uses approximation [7] to calculate bandwidth:

$$\text{“bandwidth”} = \frac{\text{total amount of data moved}}{\text{execution time}}$$

Platform details

Vendor	Name	Architecture	Abbreviation	Device Type	Theoretical Peak Mem. Bandwidth (GB/s)	Theoretical Peak FP32 FLOP/s (TFLOP/s)
Intel	Xeon Gold 6230	Cascade Lake	Xeon	HPC CPU (20C*2, 2S)	281.6	4.096
AMD	EPYC 7742	Zen2 (Rome)	EPYC	HPC CPU (64C*2, 2S)	409.6	9.216
NVIDIA	Tesla A100 (SXM 80 GB)	Ampere	A100	HPC GPU	2039	19.490
NVIDIA	Tesla V100 (PCIe 16GB)	Volta	V100	HPC GPU	900	14.130

BabelStream results on CPU

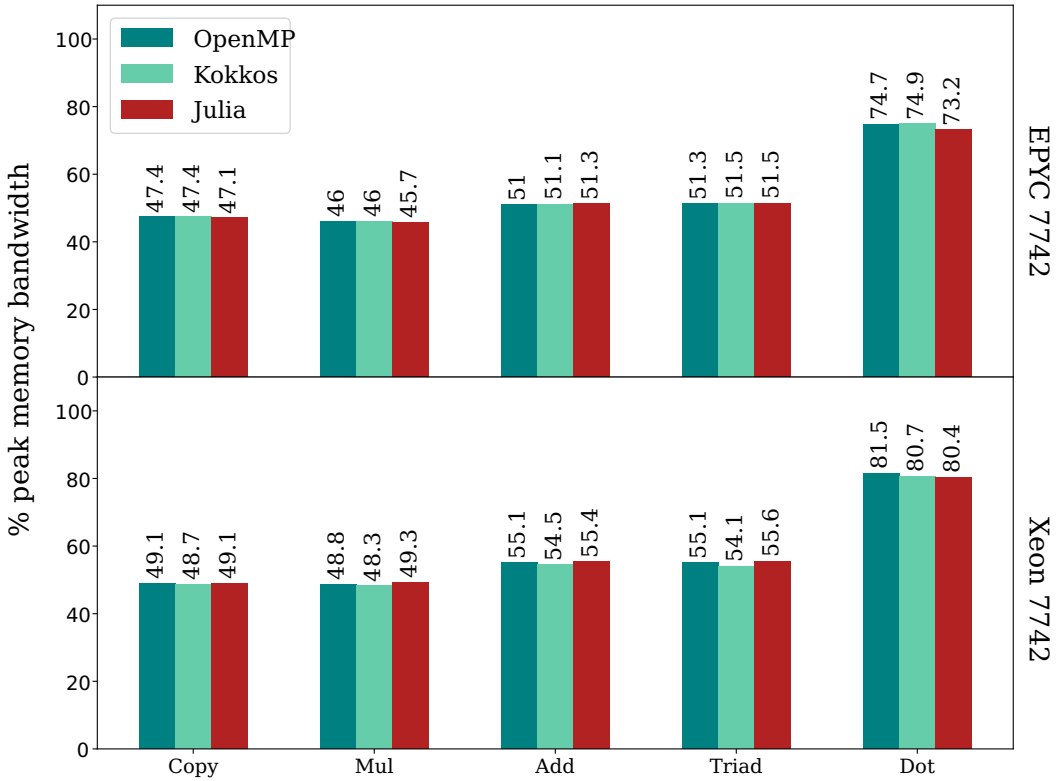

Figure: CPU results for BabelStream[3]

BabelStream results on GPU

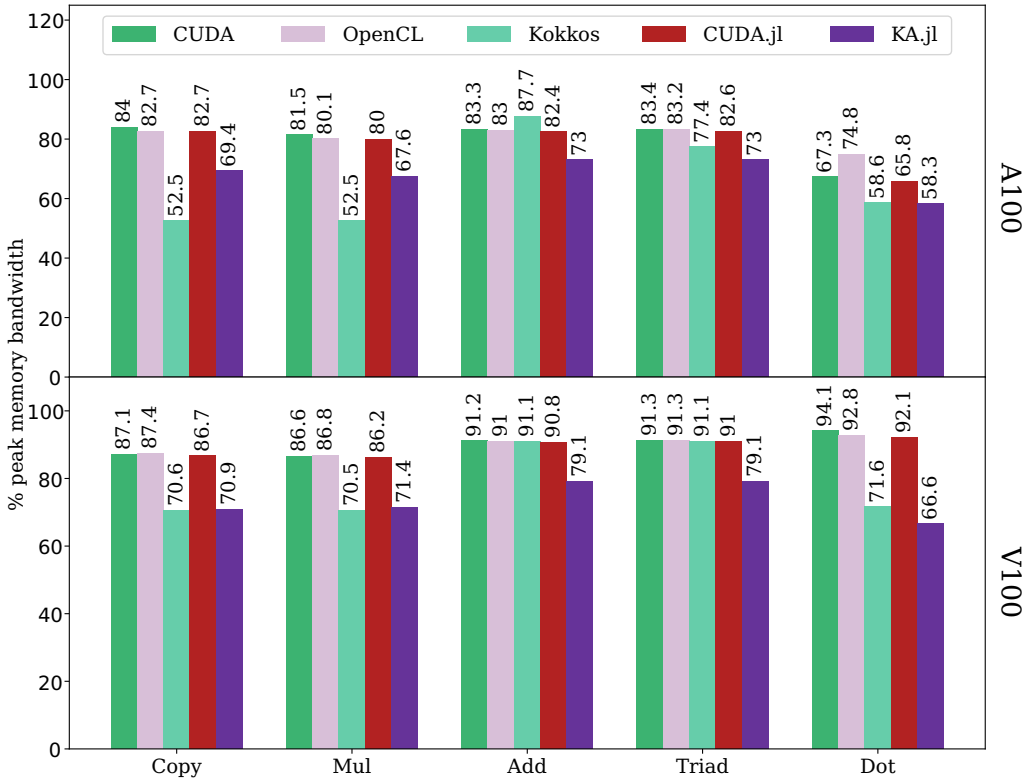

Figure: GPU results for BabelStream[3]

Oceananigans.jl

- Julia software for fluid dynamics
- on CPUs and GPUs
- suitable for
  - research
  - students
  - first-time programmers
- part of the Climate Modelling Alliance project
  - provide models to analyze climate change

Celeste.jl

- goal:
  - create list of all stars in galaxy
  - from all existing telescope data
- method:
  - used Julia to combine efficiency and performance
  - ran 1.3 million threads on 9,300 Knights Landing (KNL) nodes at NERSC
- result:
  - most accurate catalog of 188 million astronomical objects
  - only needed 14.6 minutes
  - peak performance of 1.54 petaflops

Overview

1 Introduction

2 HPC in Julia

3 Popularity
- Statistics
- Downsides

4 Conclusion

Stack Overflow Trends

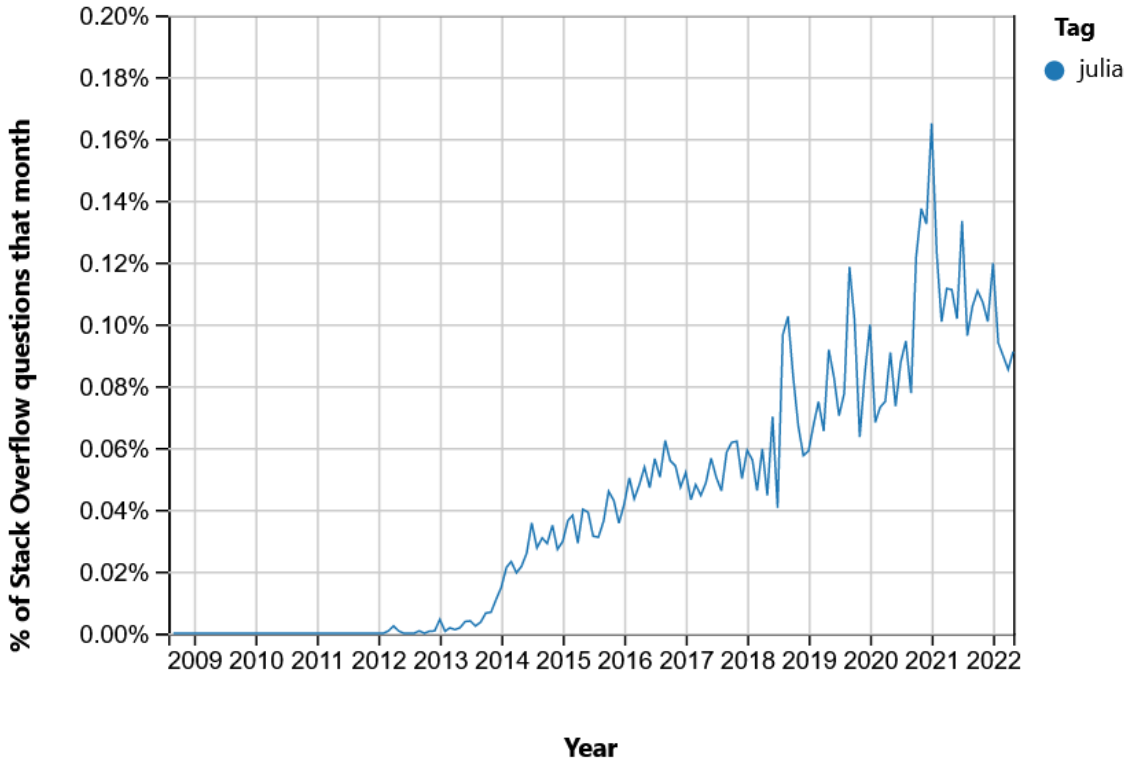

Figure: Stack Overflow trends [13]

- position 28 on the TIOBE index

Stack Overflow Trends

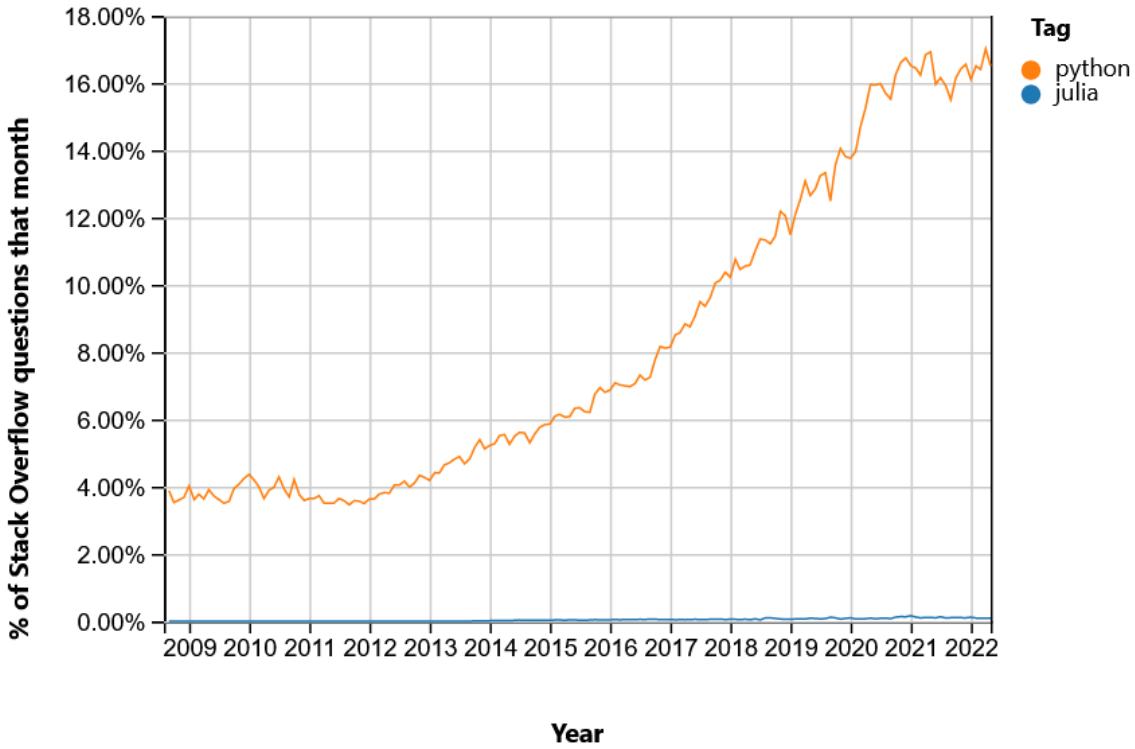

Figure: Stackoverflow Trends [13]

- position 28 on the TIOBE index

Number of packages

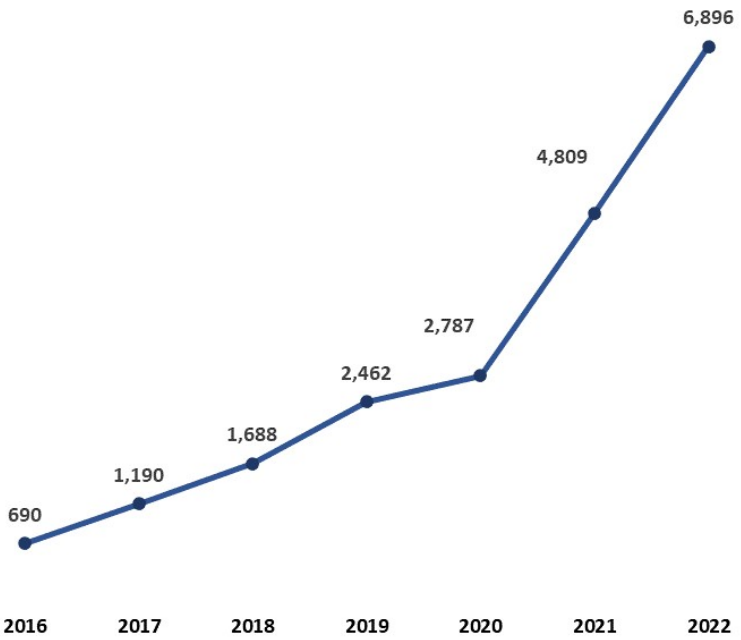

Figure: Registered Julia packages As Of Jan 1[10]

- Python has more than 350,000 packages as of Jan 17
- more than 50 times more packages for Python

What people do not like about Julia

- slow compile time
- slow time to first plot
- dynamically-typed vs. statically-typed
- not that widespread
- bugs, redesigns, or performance problems with libraries

Overview

1 Introduction

2 HPC in Julia

3 Popularity

4 Conclusion

Conclusion

- Julia not that popular yet
- libraries often not as sophisticated as in other languages
- succeeds in what it tries to achieve:
  - suitable to solve the two-language problem:
  - great productivity and performance
- could become strong competitor in HPC

References I

Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman.
Why we created julia.
https://julialang.org/blog/2012/02/why-we-created-julia/, 2012.
Accessed: 2022-05-30.

Kaifeng Gao, Gang Mei, Francesco Piccialli, Salvatore Cuomo, Jingzhi Tu, and Zenan Huo.
Julia language in machine learning: Algorithms, applications, and open issues.
Computer Science Review, Volume 37, 2020.

References II


Wei-Chen Lin and Simon McIntosh-Smith.
Comparing julia to performance portable parallel programming models for hpc.
pages 94–105, 2021.


Sascha Hunold and Sebastian Steiner.
Benchmarking julia's communication performance: Is julia hpc ready or full hpc?
In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 20–25, 2020.


Julia 1.7 documentation.
`https://docs.julialang.org/en/v1/`.
Accessed: 2022-05-30.

References III

- Why is julia so fast?
  https://groups.google.com/g/julia-users/c/Uu_UcYp49Qo?pli=1.
  Accessed: 2022-06-10.
- The stream benchmark.
  https://stackoverflow.com/questions/56086993/what-does-stream-memory-bandwidth-\benchmark-really-measure.
  Accessed: 2010-06-11.

References IV

Ali Ramadhan, Gregory LeClaire Wagner, Chris Hill, Jean-Michel Campin, Valentin Churavy, Tim Besard, Andre Souza, Alan Edelman, Raffaele Ferrari, and John Marshall. Oceananigans.jl: Fast and friendly geophysical fluid dynamics on gpus.
Journal of Open Source Software, 5(53):2018, 2020.

Parallel supercomputing for astronomy.
`https://juliacomputing.com/case-studies/celeste/`.
Accessed: 2022-06-14.

References V


Newsletter january 2022 - julia growth statistics.
https://juliacomputing.com/blog/2022/01/newsletter-january/.
Accessed: 2022-06-15.


Parallelization.
https://enccs.github.io/Julia-for-HPC/parallelization/.
Accessed: 2022-06-18.


Juliagpu.
https://juliagpu.org/.
Accessed: 2022-06-19.

References VI

Stack overflow trends.
https://insights.stackoverflow.com/trends.
Accessed: 2022-06-19.

What don't you like about julia for "serious work"?
https://discourse.julialang.org/t/what-dont-you-like-about-julia-\for-serious-work/54591.
Accessed: 2022-06-19.