HPC.NRW

PADERBORN UNIVERSITY

Paderborn Center for Parallel Computing

Introduction to Julia for High-Performance Computing

Carsten Bauer

February 13, 2024

University College London, Advanced Research Computing Center (UCL ARC)

NATIONALES HOCHLEISTUNGS RECHNEN

Course Objectives

- Learn about **Julia** for HPC
- Learn about **HPC** with Julia
- Have some fun!

	Tuesday	Wednesday	Thursday	Friday
	Foundations	**Single Core**	**Node**	**Cluster**
09:00 - 10:45 (1.75h)	Julia for HPC Fundamentals	Optimising Performance I	Multithreading	Distributed Computing
10:45 - 11:00 (15m)	Break	Break	Break	Break
11:00 - 12:30 (1.5h)	Compilation	Exercises	Exercises	Exercises
12:30 - 13:30 (1h)	Lunch	Lunch	Lunch	Lunch
13:30 - 15:00 (1.5h)	Fast & Generic Code	Optimising Performance II	GPU Programming	Exercises
15:00 - 15:30 (30m)	Break	Break	Break	Q&A
15:30 - 17:00 (1.5h)	Exercises	Exercises	Exercises	

Quick Survey

https://etc.ch/Q7QB

The Julia Programming Language

Domain science

- Dynamic & **interactive**
- High-level syntax
- Strong math support
- Package manager

High Performance Computing

- **Compiled**
- Multithreading
- Distributed computing (e.g. MPI)
- Hardware accelerators

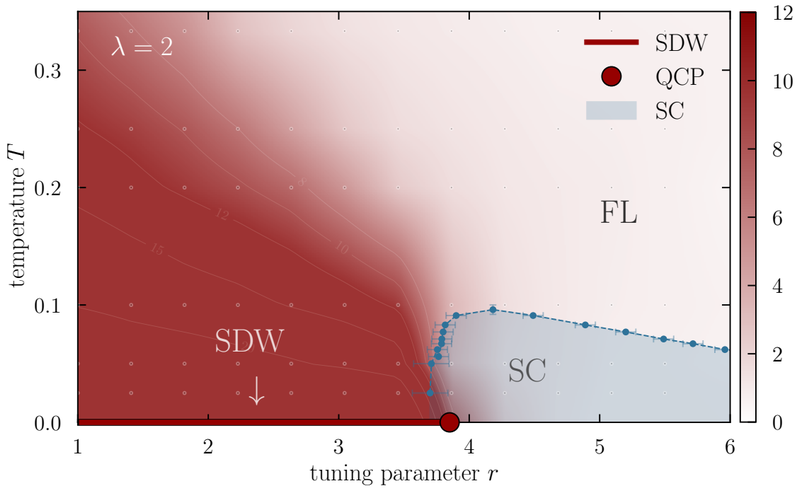
```
julia> function sumup()
           x = 0
           for i in 1:100
               x += i
           end
           return x
       end
sumup (generic function with 1 method)

julia> @code_llvm debuginfo=:none sumup()
define i64 @julia_sumup_177() #0 {
top:
  ret i64 5050
}
```

Where I Come From

Condensed matter physics / quantum field theory

- Large-scale Quantum Monte Carlo simulations
- O(15 Mio. CPU-core hours)
- O(100 TB)



Domain Science

julia

HPC

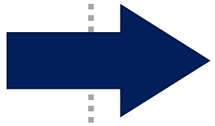
Exclusive HPC

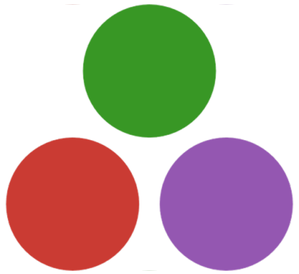
Domain Science

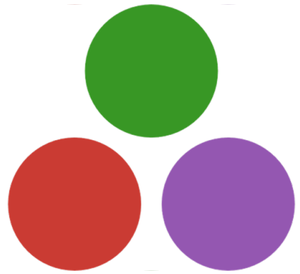
HPC

Inclusive HPC

Domain Science







HPC

LIKWID.jl + Pluto.jl

Counting Flops

```
x = rand(10_000);
```

```
function computation(x)
    x .+ x
end;
```

Counted Flops: 10000

How?

```
using LIKWID
```

```
metrics, events = @perfmon "FLOPS_DP" computation(x);
```

Compute from derived metrics

```
10000
```

```
begin
    flops_per_second = metrics["FLOPS_DP"][1]["DP [MFLOP/s]"] * 1e6
    runtime = metrics["FLOPS_DP"][1]["Runtime (RDTSC) [s]"]
    flops = round(Int, flops_per_second * runtime)
end
```

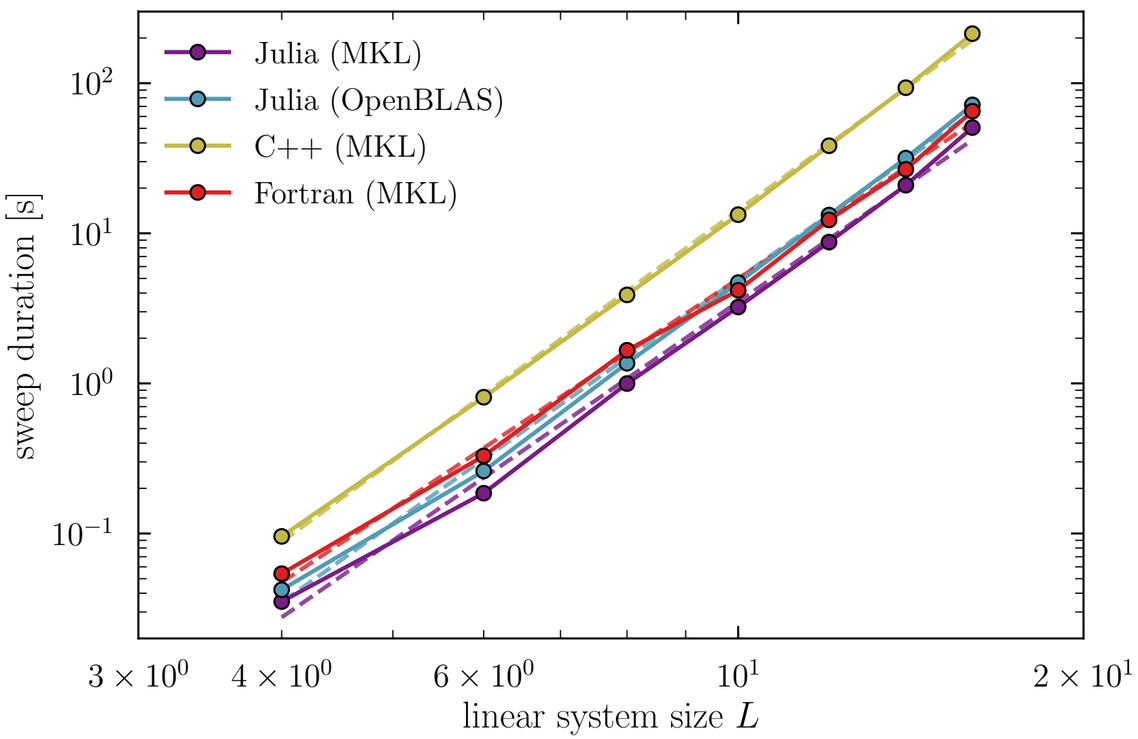
Paderborn Center for Parallel Computing

NHR FAU

NHR NATIONALES HOCHLEISTUNGS RECHNEN

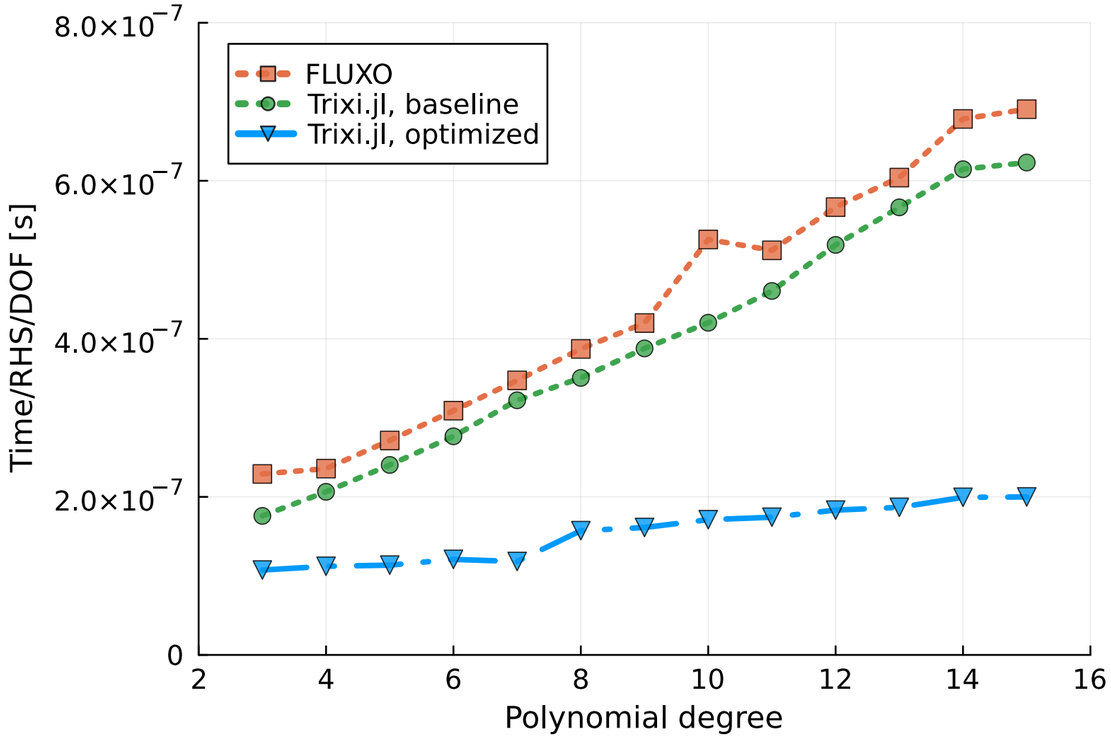
More: https://youtu.be/l2fTNfEDPC0

Julia Can Be Fast



QMC

(MonteCarlo.jl)



CFD

(Trixi.jl)

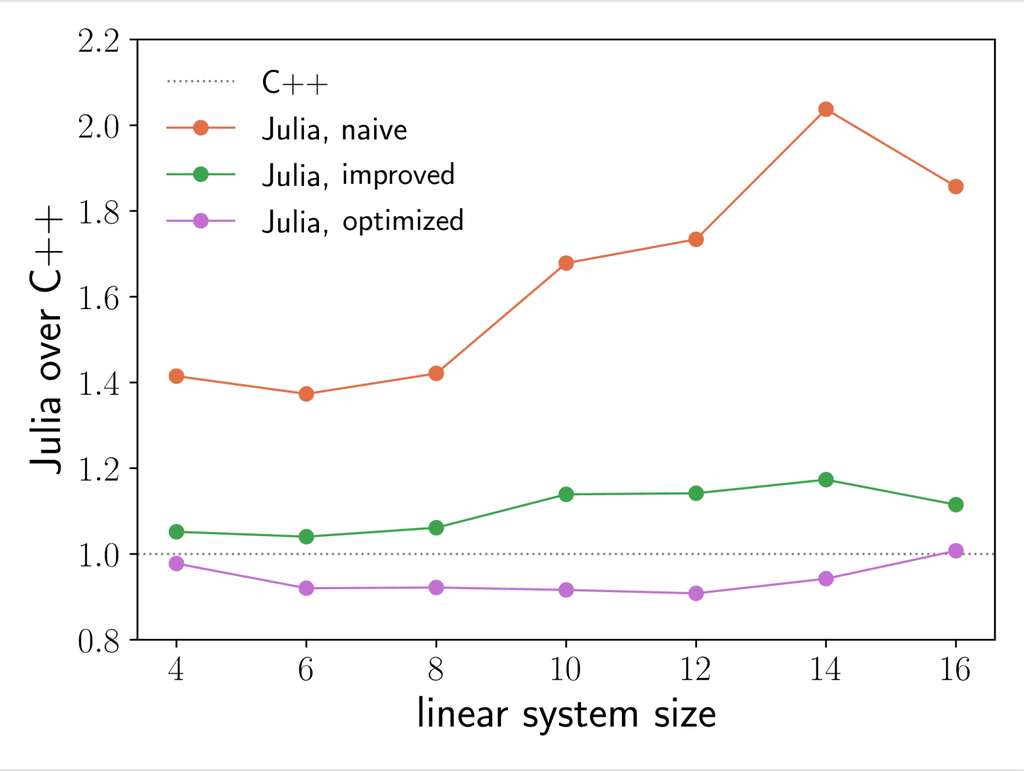
Julia can be fast

First Julia implementation

- ~5x faster than Python
- ~2x slower than C++

Gradual performance improvement, no disruptive language change!

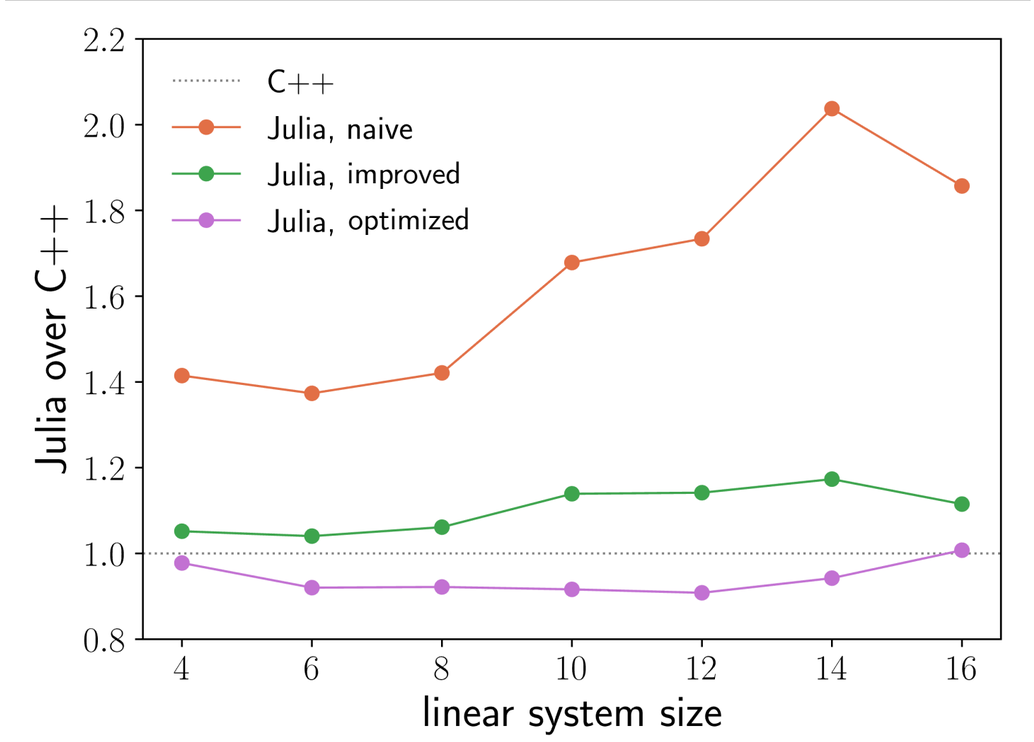
(A lot of fun!)

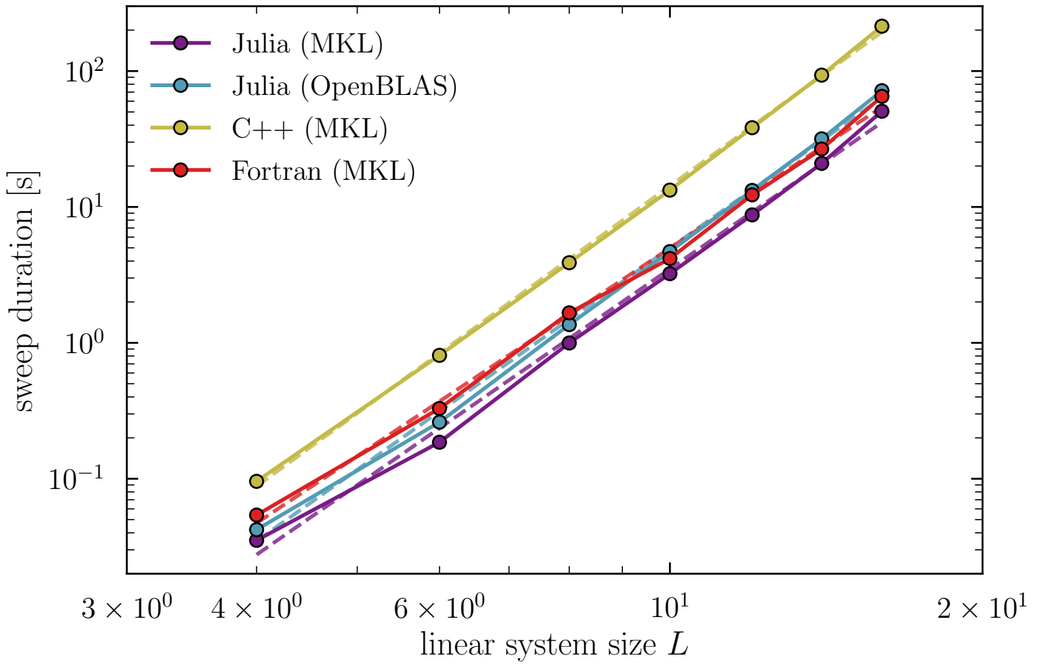


Performance

Gradual performance improvement, no disruptive language change!

(A lot of fun!)





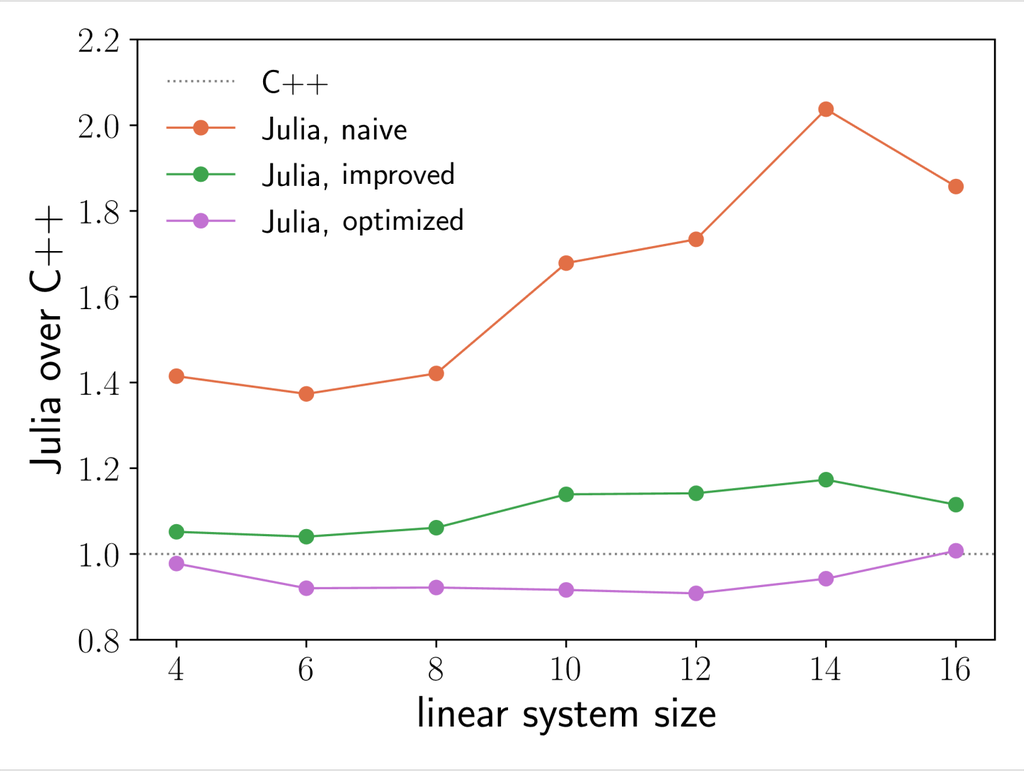
Julia can be fast

First Julia implementation

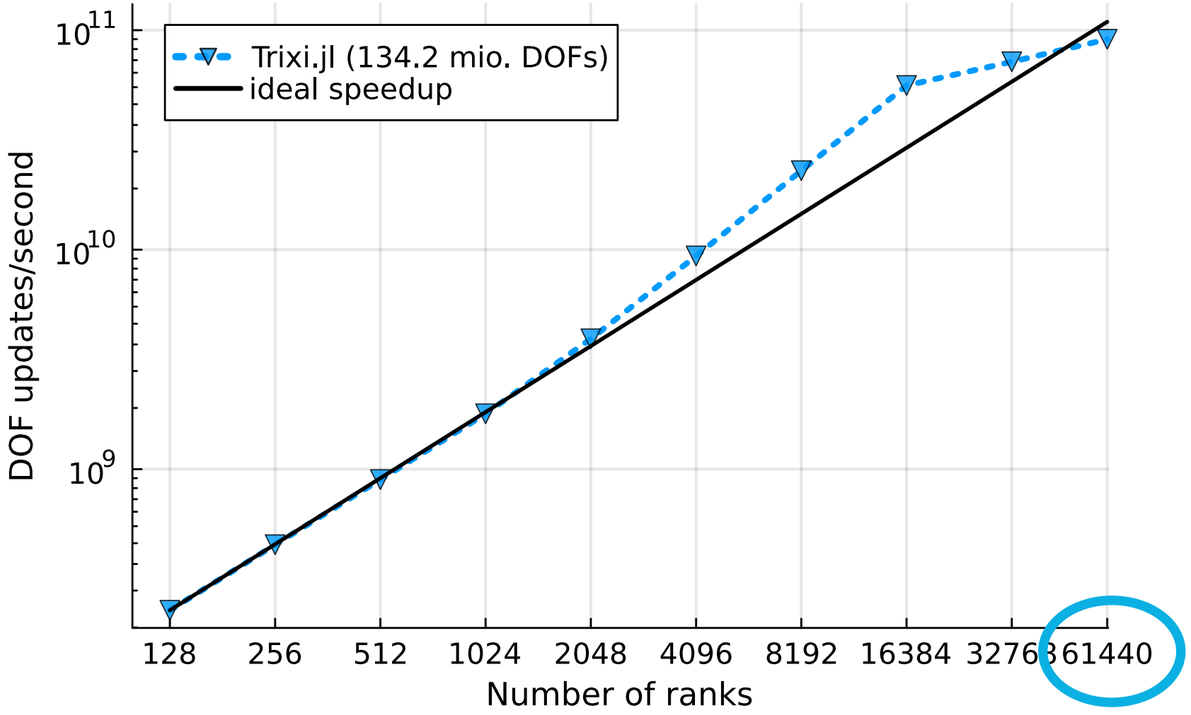
- ~5x faster than Python
- ~2x slower than C++

Gradual performance improvement, no disruptive language change!

(A lot of fun!)



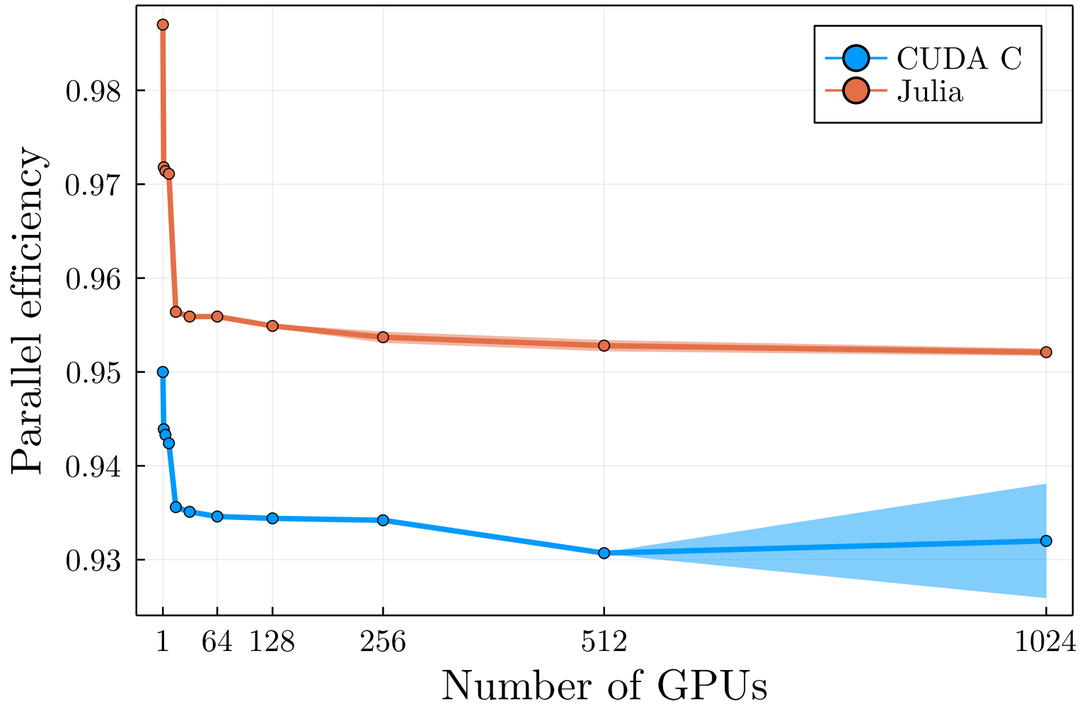
Julia Can Be Parallel



≈ 60k ranks

MPI-Parallel

(Trixi.jl)



GPU-Parallel

(ParallelStencil.jl)

Julia Can Be Portable

Laptop

```
➜  ~/myproject  tree
.
├── Manifest.toml
├── Project.toml
└── code.jl

0 directories, 3 files

➜  ~/myproject  cat Project.toml
[deps]
CUDA = "052768ef-5323-5732-b1bb-66c8b64840ba"
DifferentialEquations = "0c46a032-eb83-5123-abaf-570d42b7fbaa"
MKL = "33e6dc65-8f57-5167-99aa-e5a354878fb2"
MPI = "da04e1cc-30fd-572f-bb4f-1f8673147195"

➜  ~/myproject
```

HPC Cluster

```
➜  bauerc@n2login3 myproject  julia --project
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.7.2 (2022-02-06)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

(myproject) pkg> st
      Status `~/myproject/Project.toml`
➜ [052768ef] CUDA v3.11.0
➜ [0c46a032] DifferentialEquations v7.1.0
➜ [33e6dc65] MKL v0.5.0
➜ [da04e1cc] MPI v0.19.2
        Info packages marked with ➜ not downloaded, use `instantiat
e` to download

(myproject) pkg> instantiate
```

(As long as you stay within the Julia-verse. Using **system software** can be a bit more tricky.)

New Challenges (at Scale)

Dynamism comes at a price

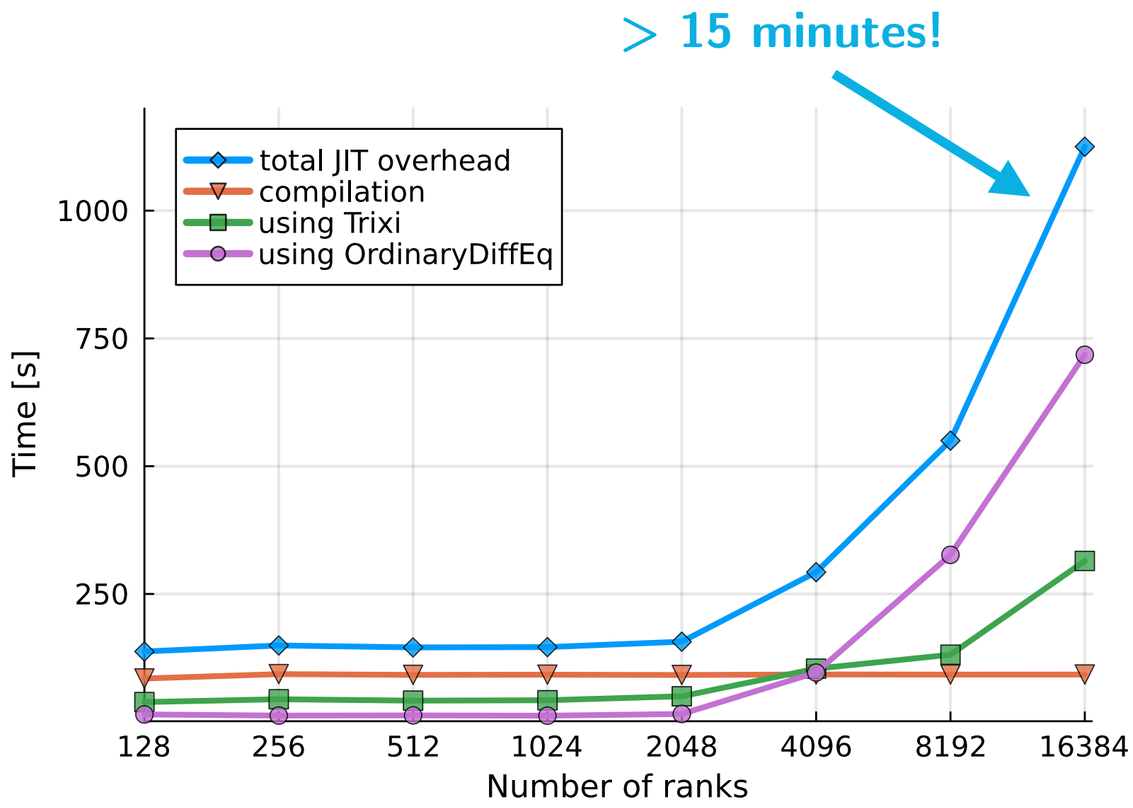
- **Type instabilities**
- No (small) **binaries**

Julia depot can get under (a lot) of pressure!

- Location matters (NFS vs GPFS vs RAM)

Many more challenges:

- "Interactivity at scale"
- Tooling
- ...



Selected Julia HPC Projects

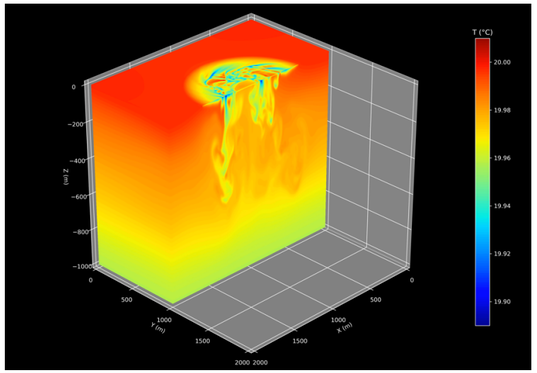
CliMA @ Caltech
- Climate Modeling Alliance

CESMIX @ MIT
- Exascale simulation of materials in extreme environments

Trixi @ RWTH Aachen / HLRS
- Computational fluid dynamics

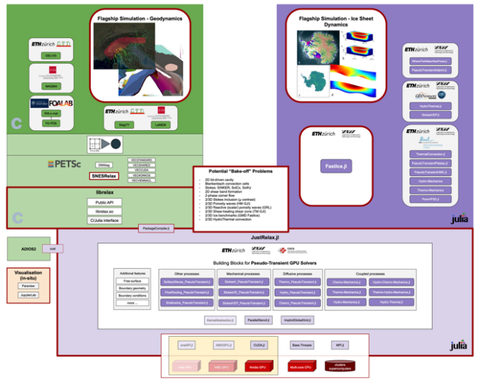
GPU4GEO @ ETH / CSCS
- Computational earth science

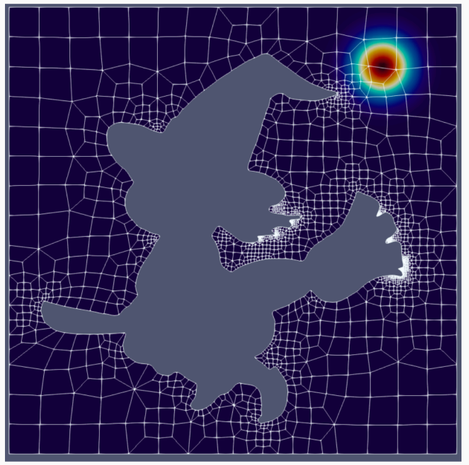
...



Oceananigans.jl

EXASCALE COMPUTING PROJECT





Credit: Andrew R. Winters

Julia HPC Community

Julia Slack

- **#hpc**
- #distributed
- #performance-helpdesk

Monthly **Julia HPC call**

- https://julialang.org/community/#events

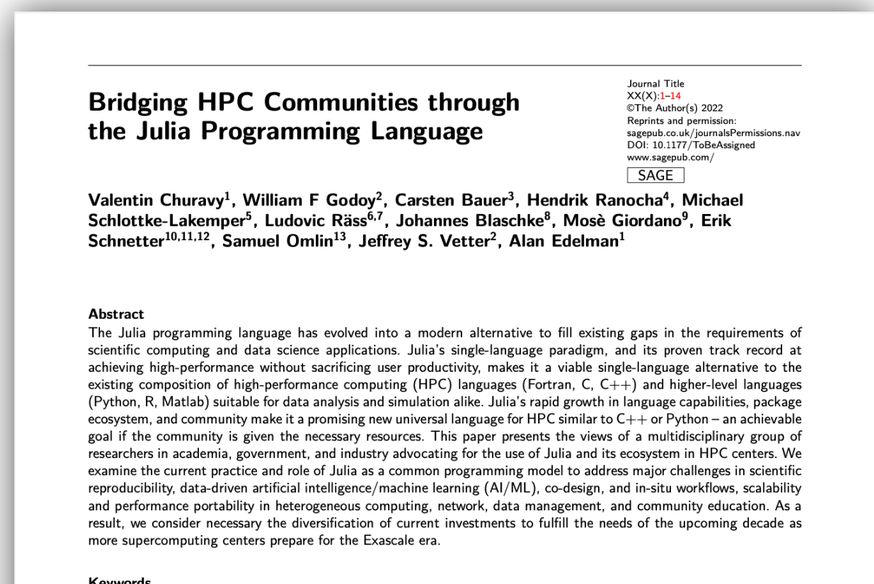
At conferences

- JuliaCon23, SC23, PASC23, SIAM CSE 23, ...


JULIA FOR HPC

JuliaCon 2023 Minisymposium

arxiv.org/abs/2211.02740



Bridging HPC Communities through the Julia Programming Language

Journal Title
XX(X):1–14

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Valentin Churavy[1], William F Godoy[2], Carsten Bauer[3], Hendrik Ranocha[4], Michael Schlottke-Lakemper[5], Ludovic Räss[6,7], Johannes Blaschke[8], Mosè Giordano[9], Erik Schnetter[10,11,12], Samuel Omlin[13], Jeffrey S. Vetter[2], Alan Edelman[1]

Abstract

The Julia programming language has evolved into a modern alternative to fill existing gaps in the requirements of scientific computing and data science applications. Julia's single-language paradigm, and its proven track record at achieving high-performance without sacrificing user productivity, makes it a viable single-language alternative to the existing composition of high-performance computing (HPC) languages (Fortran, C, C++) and higher-level languages (Python, R, Matlab) suitable for data analysis and simulation alike. Julia's rapid growth in language capabilities, package ecosystem, and community make it a promising new universal language for HPC similar to C++ or Python – an achievable goal if the community is given the necessary resources. This paper presents the views of a multidisciplinary group of researchers in academia, government, and industry advocating for the use of Julia and its ecosystem in HPC centers. We examine the current practice and role of Julia as a common programming model to address major challenges in scientific reproducibility, data-driven artificial intelligence/machine learning (AI/ML), co-design, and in-situ workflows, scalability and performance portability in heterogeneous computing, network, data management, and community education. As a result, we consider necessary the diversification of current investments to fulfill the needs of the upcoming decade as more supercomputing centers prepare for the Exascale era.

Summary

Julia can be a great option for HPC!

- serial and parallel performance on-par with Fortran/C/C++
- portability and high-productivity
- new opportunities, e.g. interactive HPC

New challenges

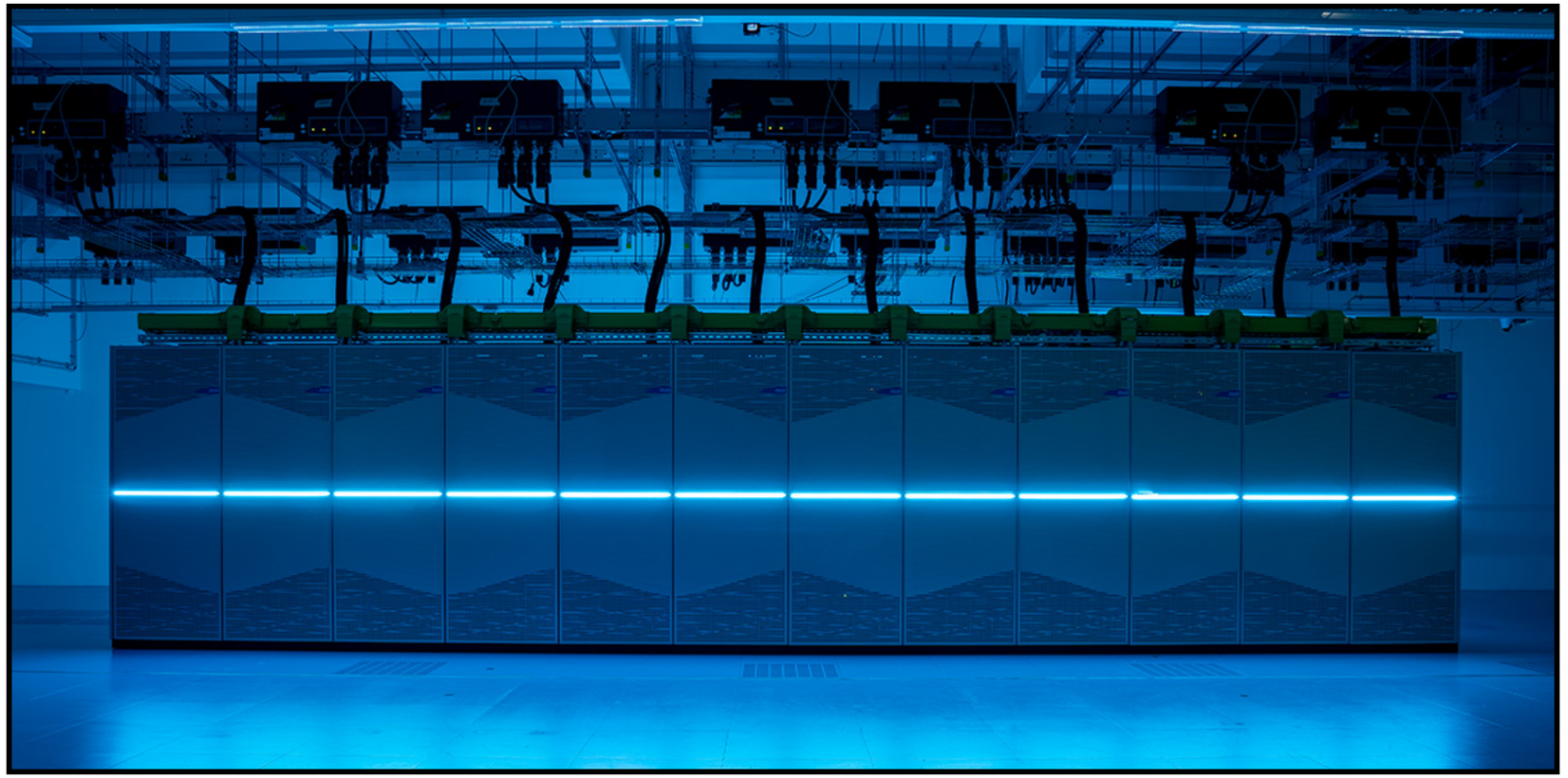
- workflow / interactivity at scale, Julia depot, system binaries, ...
- type instabilities

"Early adoption" cost

- Julia for HPC is a niche
- lack of documentation, tooling, ...

Let's get started!

- **PC2 JupyterHub**
  - https://pc2.de/go/jupyterhub



Noctua 2