The Julia programming language in the Quantum Community

An overview of the Julia programming language and its applications in quantum software

Xiuzhe (Roger) Luo

2023-11-17

Perimeter Institute, University of Waterloo.

Outline

1. From Punch Card to JIT Compiler
2. Introduction of Selected Packages
3. The Quantum "Gang" in Julia Community

From Punch Card to JIT Compiler

Fortran 1957 - The Punch Card Language

Fortran ***1957***

punch cards

- uppercase only
- goto statements
- multi-dimensional arrays
- …

I WOULD LIKE TO SAY THANKS TO ALL OUR VISITORS! PLEASE KEEP COMING BACK!
MASS:WERK DATA CENTER <masswerk.at>
CDL 0815

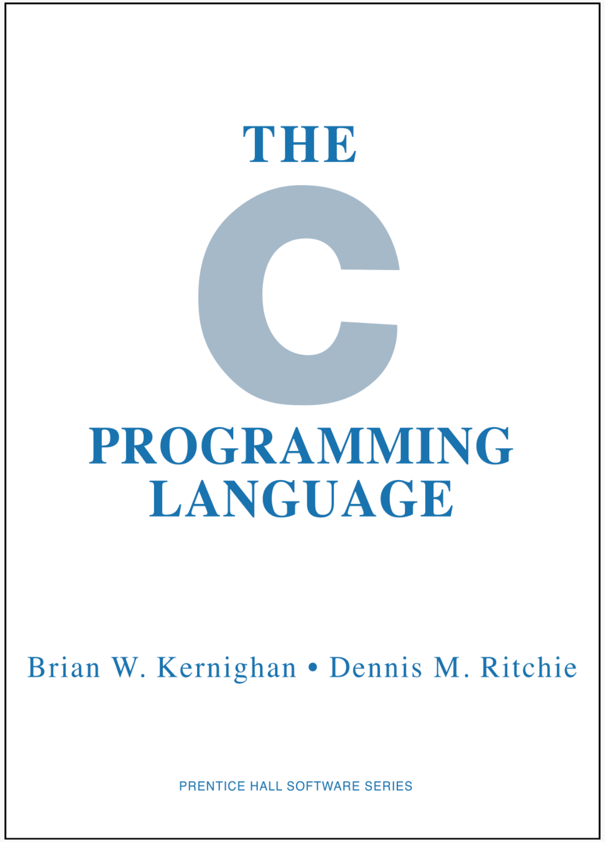
Punch Card generated by Fortran *by Nobert Landsteiner*

C - Compiler Language

C ***1972***

the compiled language

- type system
- pointers
- structs
- …



THE
C
PROGRAMMING LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES

The C Programming Language *by Brian Kernighan and Dennis Ritchie*

Python - Interpreter Language

Python *1991*

the interpreted language

- dynamic typing
- garbage collection
- package manager
- …

Well-known Python Libraries

- NumPy
- SciPy
- Pandas
- Matplotlib
- …

The Python Logo

Compiled vs Interpreted

Compiled

Pros

- no runtime or small runtime
- finer analysis thus more optimizations

Interpreted

Pros

- faster development cycle
- easier to learn
- interactive

Compiled vs Interpreted

Compiled

Pros

- no runtime or small runtime
- finer analysis thus more optimizations

Cons

- slower development cycle
- stiff learning curve
- not interactive

Interpreted

Pros

- faster development cycle
- easier to learn
- interactive

Cons

- slower
- bigger runtime

Why not X for Scientific Computing?

Scientific Computing

- extreme performance requirements
- a lot of deadlines (fast development cycle)
- non-professional programmers (scientists)

Frontier - The World's Fastest Supercomputer (2023).

Why not X for Scientific Computing?

Scientific Computing

- extreme performance requirements
- a lot of deadlines (fast development cycle)
- non-professional programmers (scientists)

More and more complicated HPC

- heterogeneous hardware (CPU, GPU, …)
- distributed systems (cluster, cloud, …)
- …

Frontier - The World's Fastest Supercomputer (2023).

Why not X for Scientific Computing?

Scientific Computing

- extreme performance requirements
- a lot of deadlines (fast development cycle)
- non-professional programmers (scientists)

More and more complicated HPC

- heterogeneous hardware (CPU, GPU, …)
- distributed systems (cluster, cloud, …)
- …

Requires

- highly optimized code
- fast development cycle
- easy to learn
- interactive
- heterogeneous programming

Why not X for Scientific Computing?

Scientific Computing

- extreme performance requirements
- a lot of deadlines (fast development cycle)
- non-professional programmers (scientists)

More and more complicated HPC

- heterogeneous hardware (CPU, GPU, …)
- distributed systems (cluster, cloud, …)
- …

Requires

- highly optimized code
- fast development cycle
- easy to learn
- interactive
- heterogeneous programming

Are we looking for a silver bullet?

Julia - the Language for JIT

Julia

the language designed for JIT

how to make python run faster
Julia
Look what they need just to
mimic a fraction of our power!

Julia - the Language for JIT

Julia

the language designed for JIT

What is JIT?

Just-In-Time compilation

- compile code at runtime
- ship the interpreter along with a compiler
- optimize only frequently executed code

how to make python run faster
Julia
Look what they need just to mimic a fraction of our power!

Julia - the Language for JIT

Julia

the language designed for JIT

What is JIT?

Just-In-Time compilation

- compile code at runtime
- ship the interpreter along with a compiler
- optimize only frequently executed code

bonus: compiler can know more about your code by running interpreter

how to make python run faster
Julia
Look what they need just to mimic a fraction of our power!

Julia - the Language for Science

Designed for Science

- nice syntax for scientific tasks

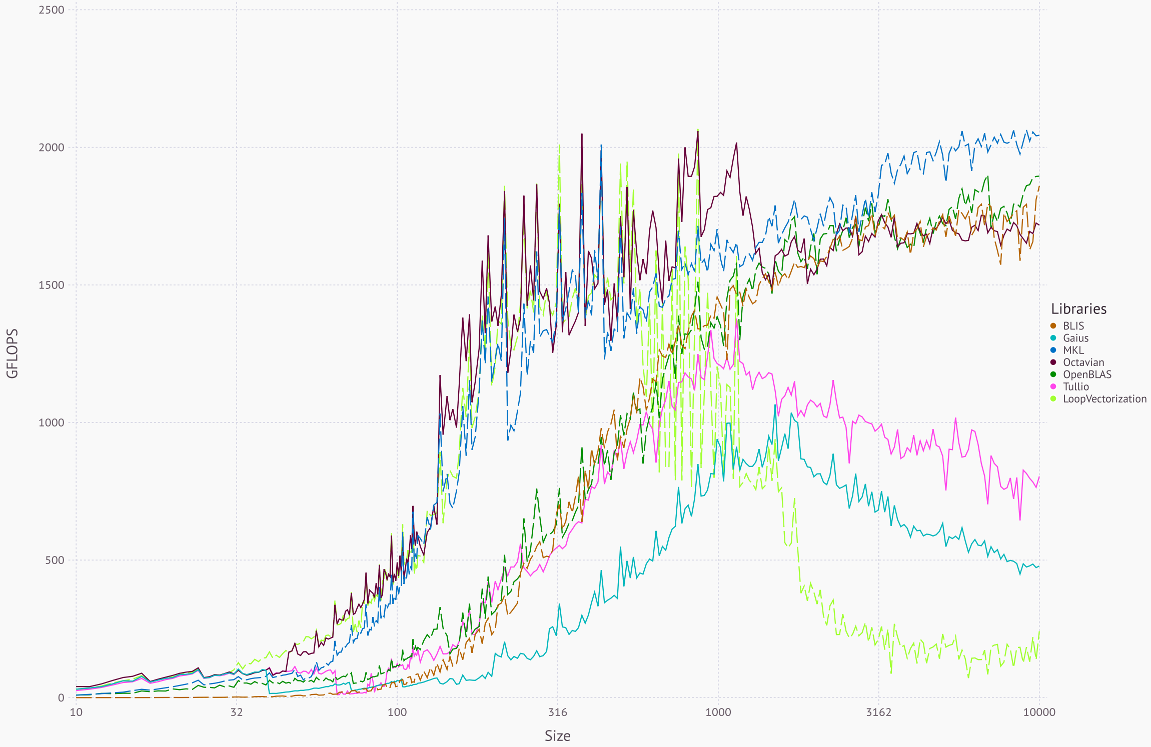
```
# broadcasting
sin.(A)
A .+ B .+ C

[1, 2, 3]        # Vector
[1 2 3;2 3 4]    # Matrix
[1 2 3;2 3 4;;; # Tensor
 1 1 1;2 2 2]
```

Julia - the Language for Science

Designed for Science

- nice syntax for scientific tasks
- comparable performance to highly-optimized C & Assembly



Native Julia GEMM vs OpenBLAS & MKL

Julia - the Language for Science

Designed for Science

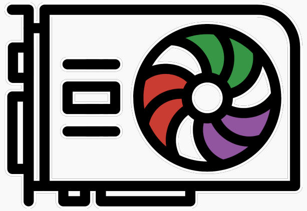
- nice syntax for scientific tasks
- comparable performance to highly-optimized C & Assembly
- a rich ecosystem of matrices & arrays with unified interface

- Dense Arrays (builtin)
- SparseMatrix (builtin)
- Heterogeneous Arrays (CUDA, OpenCL, AMD, Metal, ...)
- Special Matrices (Permutation, Tridiagonal, Gaussian, ...)
- ...

Julia - the Language for Science

Designed for Science

- nice syntax for scientific tasks
- comparable performance to highly-optimized C & Assembly
- a rich ecosystem of matrices & arrays with unified interface
- heterogeneous programming



The JuliaGPU organization

- KernelAbstractions.jl - unified interface for GPU programming
- CUDA.jl - high-level and kernel programming with CUDA
- AMDGPU.jl - high-level and kernel programming with AMD GPUs
- Metal.jl - high-level and kernel programming with Apple GPUs

Julia - the Cons

The Technical Price of JIT

- bigger runtime
- slower startup time (warm-up)
- not able to generate small binaries (yet/never)

The Non-Technical Cons

- relatively young and small community (not an ideal language for business)
- evolving ecosystem

Julia - the Cons

The Technical Price of JIT

- bigger runtime
- slower startup time (warm-up)
- not able to generate small binaries (yet/never)

Yes

- (performance) build from scratch & runs > 5s
- (expressiveness) my code has many different cases

The Non-Technical Cons

- relatively young and small community (not an ideal language for business)
- evolving ecosystem

Julia - the Cons

The Technical Price of JIT

- bigger runtime
- slower startup time (warm-up)
- not able to generate small binaries (yet/never)

Yes

- (performance) build from scratch & runs > 5s
- (expressiveness) my code has many different cases

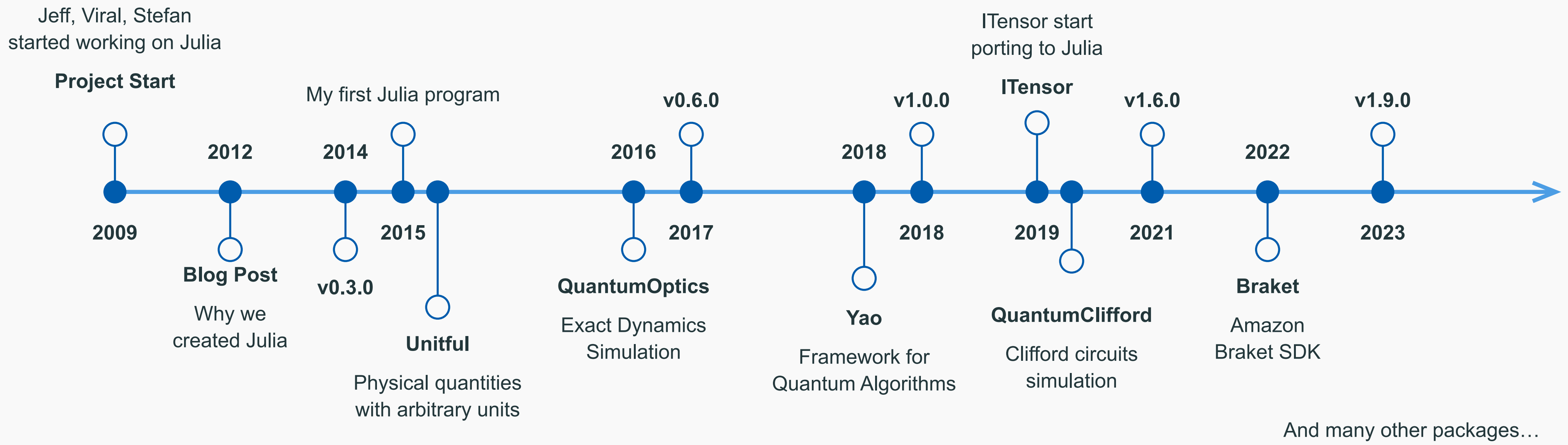
The Non-Technical Cons

- relatively young and small community (not an ideal language for business)
- evolving ecosystem

No

- Just wanna write a tiny CLI app (use rust)
- Mainly want to use a package (e.g PyTorch)
- My users are not technical
  - no new language (use python)
  - no code (use web app interface, e.g drag & drop circuit compilation)

A Long History with Quantum



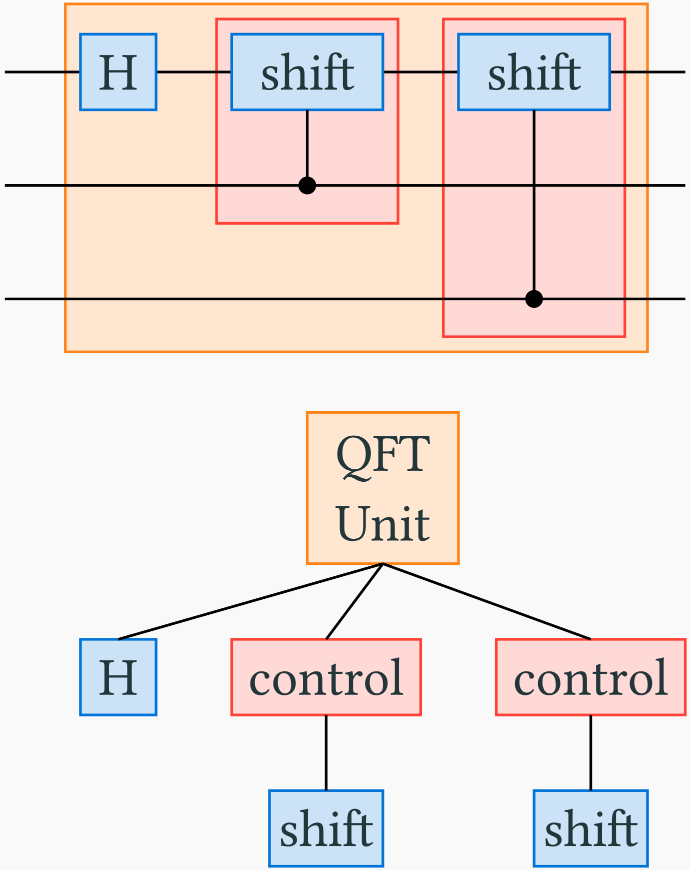
Introduction of Selected Packages

The Circuit Expression in Yao

A differentiable, efficient & extensible framework for quantum algorithm design. Top performance in exact simulation.

Features

- Ability to compose circuits



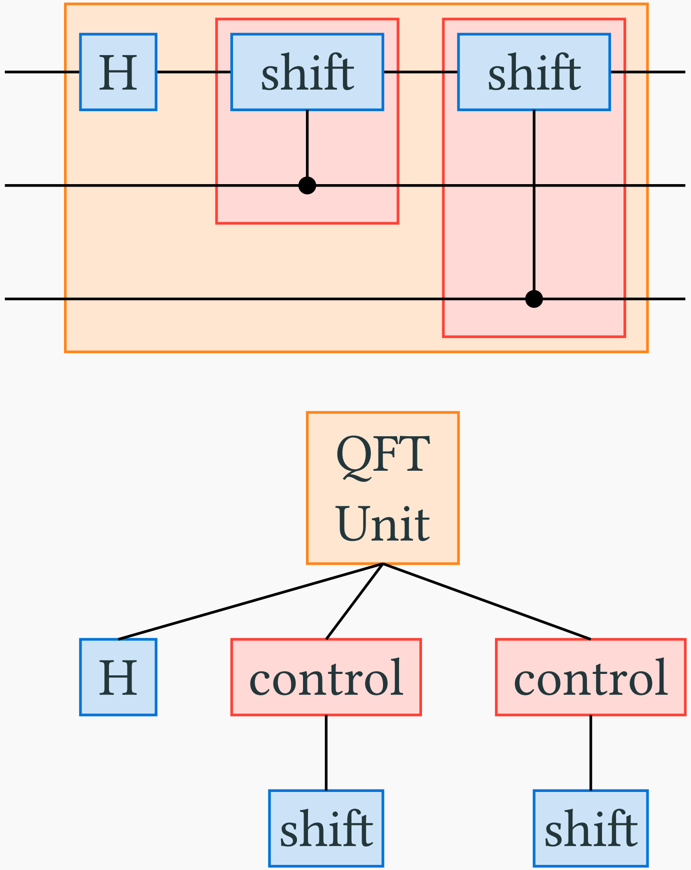
Xiuzhe Luo, Jinguo Liu, Pan Zhang and Lei Wang, Quantum, 2019

The Circuit Expression in Yao

A differentiable, efficient & extensible framework for quantum algorithm design. Top performance in exact simulation.

Features

- Ability to compose circuits
- Automatic differentiation (AD by reversibility, first in quantum)



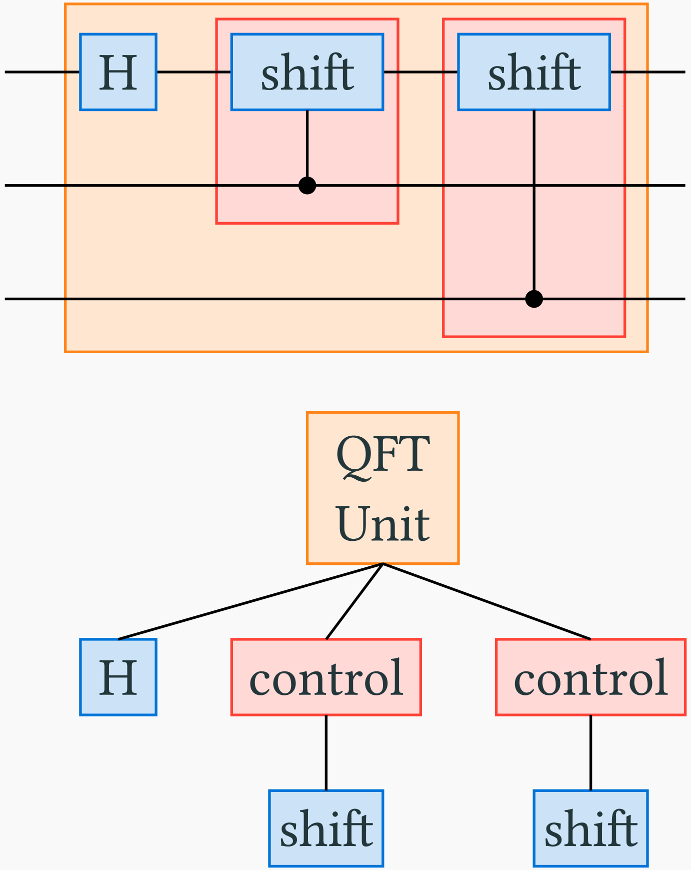
Xiuzhe Luo, Jinguo Liu, Pan Zhang and Lei Wang, Quantum, 2019

The Circuit Expression in Yao

A differentiable, efficient & extensible framework for quantum algorithm design. Top performance in exact simulation.

Features

- Ability to compose circuits
- Automatic differentiation (AD by reversibility, first in quantum)
- Fast and generic simulation (Matrix × Symbol vs Matrix × Matrix)



Xiuzhe Luo, Jinguo Liu, Pan Zhang and Lei Wang, Quantum, 2019

The Circuit Expression in Yao

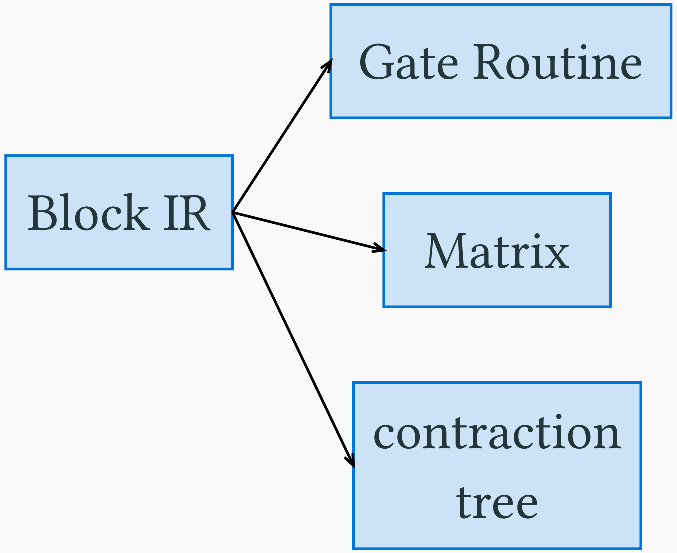
A differentiable, efficient & extensible framework for quantum algorithm design. Top performance in exact simulation.

Features

- Ability to compose circuits
- Automatic differentiation (AD by reversibility, first in quantum)
- Fast and generic simulation (Matrix × Symbol vs Matrix × Matrix)

Use cases

- Varitional quanutm algorithms (VQA)
- Tensor network circuits
- Matrix representation construction for general operators



Xiuzhe Luo, Jinguo Liu, Pan Zhang and Lei Wang, Quantum, 2019

Ecosystem

Addon Packages

- OpenQASM, YaoBlocksQASM: support for OpenQASM 2.0

Ecosystem

Addon Packages

- OpenQASM, YaoBlocksQASM: support for OpenQASM 2.0
- Bloqade: SDK for neutral atom quantum computing

Ecosystem

Addon Packages

- OpenQASM, YaoBlocksQASM: support for OpenQASM 2.0
- Bloqade: SDK for neutral atom quantum computing
- ZXCalculus: A port of `pyzx` to Julia

Ecosystem

Addon Packages

- OpenQASM, YaoBlocksQASM: support for OpenQASM 2.0
- Bloqade: SDK for neutral atom quantum computing
- ZXCalculus: A port of `pyzx` to Julia
- YaoToEinsum: convert Yao.jl circuits to tensor network contraction

Ecosystem

Addon Packages

- OpenQASM, YaoBlocksQASM: support for OpenQASM 2.0
- Bloqade: SDK for neutral atom quantum computing
- ZXCalculus: A port of `pyzx` to Julia
- YaoToEinsum: convert Yao.jl circuits to tensor network contraction
- QuAlgorithmZoo: a collection of quantum algorithms

Ecosystem

Addon Packages

- OpenQASM, YaoBlocksQASM: support for OpenQASM 2.0
- Bloqade: SDK for neutral atom quantum computing
- ZXCalculus: A port of `pyzx` to Julia
- YaoToEinsum: convert Yao.jl circuits to tensor network contraction
- QuAlgorithmZoo: a collection of quantum algorithms
- YaoPlots: visualize quantum circuits

Ecosystem

Addon Packages

- OpenQASM, YaoBlocksQASM: support for OpenQASM 2.0
- Bloqade: SDK for neutral atom quantum computing
- ZXCalculus: A port of `pyzx` to Julia
- YaoToEinsum: convert Yao.jl circuits to tensor network contraction
- QuAlgorithmZoo: a collection of quantum algorithms
- YaoPlots: visualize quantum circuits
- FLOYao: A fermionic linear optics simulator backend for Yao.jl

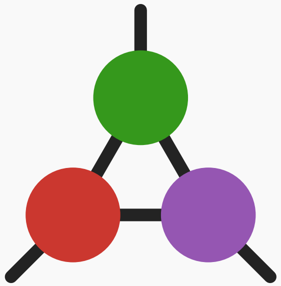
Tensor Networks

Tensor Networks

Tensor networks has a long history in Julia community.

TensorOperations

static & basic contraction order optimization, with good performance and syntax.



TensorOperations.jl

```julia
using TensorOperations
α = randn()
A = randn(5, 5, 5, 5, 5, 5)
B = randn(5, 5, 5)
C = randn(5, 5, 5)
D = zeros(5, 5, 5)
@tensor begin
    D[a, b, c] = A[a, e, f, c, f, g] * B[g, b, e] + α * C[c, a, b]
    E[a, b, c] := A[a, e, f, c, f, g] * B[g, b, e] + α * C[c, a, b]
end
```

Tensor Networks

Tensor networks has a long history in Julia community.

TensorOperations

static & basic contraction order optimization, with good performance and syntax.

ITensors

long established, with good performance and a strong DMRG related algorithm implementation.

ITensors.jl

```
using ITensors
N = 100
sites = siteinds("S=1",N)

os = OpSum()
for j=1:N-1
  os += "Sz",j,"Sz",j+1
  os += 1/2,"S+",j,"S-",j+1
  os += 1/2,"S-",j,"S+",j+1
end
H = MPO(os,sites)
```

```
psi0 = randomMPS(sites,10)

nsweeps = 5
maxdim = [10,20,100,100,200]
cutoff = [1E-10]

energy, psi = dmrg(H,psi0; nsweeps, maxdim, cutoff)
```

Tensor Networks

Tensor Networks

Tensor networks has a long history in Julia community.

TensorOperations

static & basic contraction order optimization, with good performance and syntax.

ITensors

long established, with good performance and a strong DMRG related algorithm implementation.

OMEinsum

a new package, with a focus on contraction optimization and automatic differentiation.

OMEinsum.jl

```
julia> optcode = optimize_code(code, uniformsize(code, 3), TreeSA())
SlicedEinsum{Char, DynamicNestedEinsum{Char}}(Char[], ago, goa ->
├─ ago
└─ gcojl, cjal -> goa
   ├─ bgck, bojlk -> gcojl
   ⋮  ⋮  ⋮
   └─ cjf, afl -> cjal
       ├─ cjf
       └─ afl
)
julia> contraction_complexity(optcode, uniformsize(optcode, 3))
Time complexity: 2^12.737881076857779
Space complexity: 2^7.92481250360578
Read-write complexity: 2^11.247334178028728

julia> optcode(fill(s, 10)...)[]
0
```

And More …

QuantumOptics: exact simulation of general quantum system

QuantumInformation

A Julia package for numerical computation in quantum information theory

QuantumSavory

Clifford circuits, graph states, and other quantum Stabilizer formalism tools.

QuantumCircuitOpt

A Julia/JuMP Package for Optimal Quantum Circuit Design

DFTK

DFTK: Density-functional toolkit

…

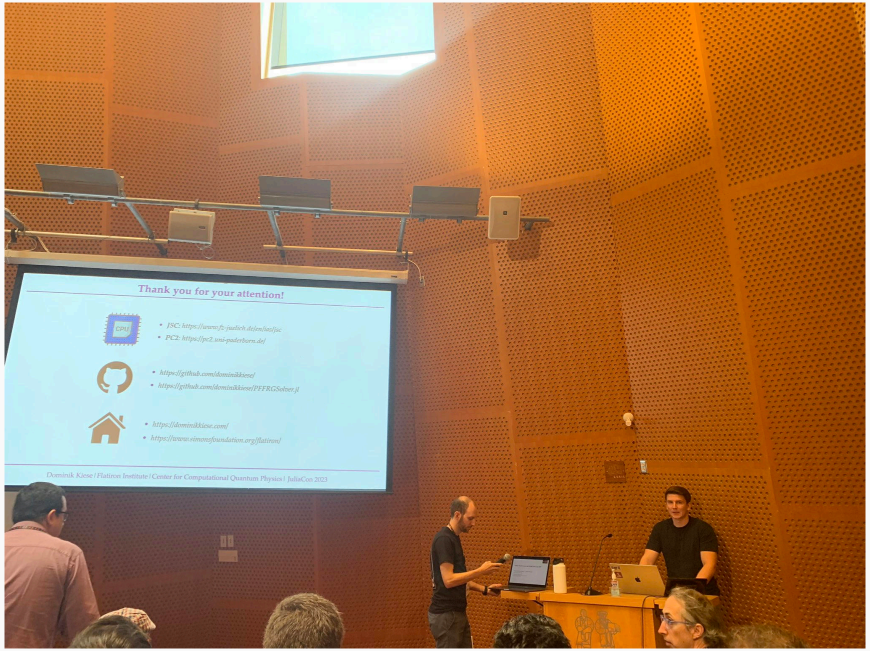
The Quantum "Gang" in Julia Community

The Quantum "Gang" in Julia

The Quantum Track at JuliaCon 2023

Organizers

- Xiuzhe (Roger) Luo, Perimeter Institute
- Katharine Hyatt, AWS Braket
- Ashley Milsted, AWS Braket
- Matthew Fishman, Flatiron Institute
- Miles Stoudenmire, Flatiron Institute
- Michael F. Herbst, EPFL



Thank you for your attention!

- JSC: https://www.fz-juelich.de/en/ias/jsc
- PC2: https://pc2.uni-paderborn.de/

- https://github.com/dominikkiese/
- https://github.com/dominikkiese/PFFRGSolver.jl

- https://dominikkiese.com/
- https://www.simonsfoundation.org/flatiron/

Dominik Kiese | Flatiron Institute | Center for Computational Quantum Physics | JuliaCon 2023

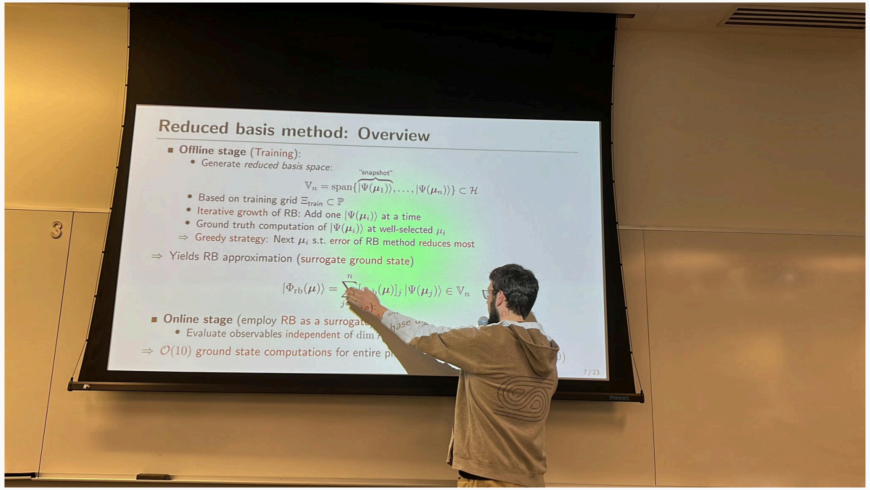
ITensor presentation

The Quantum "Gang" in Julia

The Quantum Track at JuliaCon 2023

over 14 talks including (online not included):

- Quantum Information
- Tensor Networks
- Quantum Chemistry
- Quanutm Control
- ...


Reduced basis method: Overview
Offline stage (Training):
Generate reduced basis space:
"snapshot"
$\mathbb{V}_n = \mathrm{span}\{|\Psi(\mu_1)\rangle, \ldots, |\Psi(\mu_n)\rangle\} \subset \mathcal{H}$
Based on training grid $\Xi_{\text{train}} \subset \mathbb{P}$
Iterative growth of RB: Add one $|\Psi(\mu_i)\rangle$ at a time
Ground truth computation of $|\Psi(\mu_i)\rangle$ at well-selected μ_i
⇒ Greedy strategy: Next μ_i s.t. error of RB method reduces most
⇒ Yields RB approximation (surrogate ground state)
$|\Phi_{\text{rb}}(\mu)\rangle = \sum^{n}$ $(\mu)]_j |\Psi(\mu_j)\rangle \in \mathbb{V}_n$
Online stage (employ RB as a surrogat
Evaluate observables independent of dim
⇒ $\mathcal{O}(10)$ ground state computations for entire p
7 / 23
3

DFTK talk by Michael Herbst

The Quantum "Gang" in Julia

The Quantum Track at JuliaCon 2023

1-day mini-symposium within JuliaCon

- chat with package developers and users
- learn about the latest developments
- discuss the future of quantum software in Julia

JuliaCon 2024 is coming to Eindhoven, Netherlands!

QuantumSymbolics by Stefan Krastanov