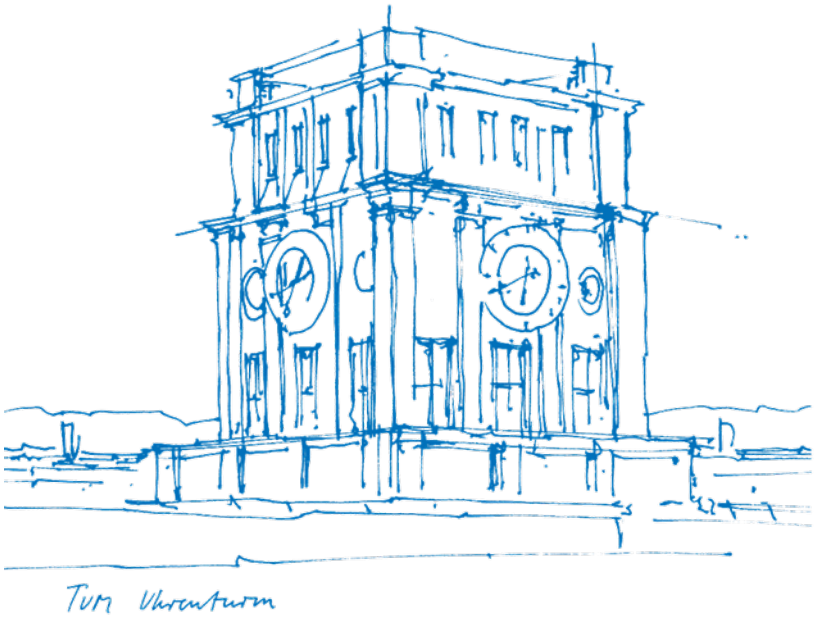
# Qaintum: A Julia-based Simulation Framework for Quantum Circuits

Qunsheng Huang  Ismael Medina  Esther Cruz  Shin Ho Cho  Christian Mendl

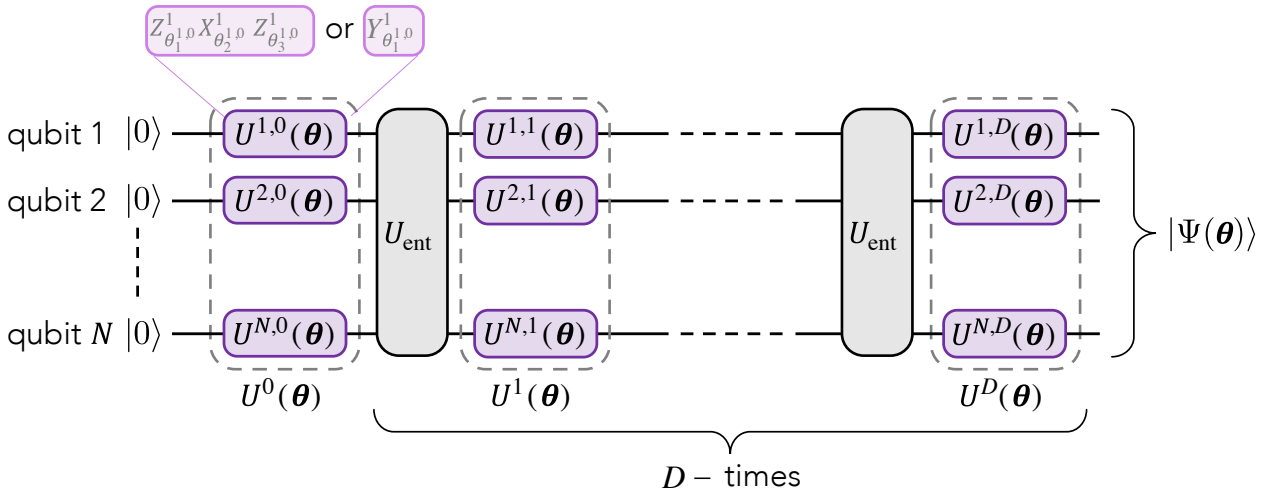
Technical University of Munich
Department of Informatics
Chair of Scientific Computing

München, 16. März 2021



# Why yet another (digital) quantum simulator?

- Learning tool
- Flexibility (e.g., add support for qudits, density matrices, . . . )
- Open source
- High performance of Julia




- Long-term goal: full quantum computing software stack

M. Schuld et al. “Evaluating analytic gradients on quantum hardware”.arXiv:1811.11184v1

# Qaintessent.jl

- Supports statevector or density matrix simulation with standard gate sets and arbitrary unitary gates
- Supports unitary matrix decomposition
- Rudimentary graph-based optimization

```
"""
Tailored apply for HadamardGate
"""
function _apply(cg::CircuitGate{1,HadamardGate}, ψ::Vector{<:Complex}, N::Int)
    i = cg.iwire[1]
    ψ = reshape(ψ, 2^(i-1), 2, 2^(N-i))
    A = similar(ψ)
    A[:, 1, :] .= (ψ[:, 1, :] .+ ψ[:, 2, :]) ./ sqrt(2)
    A[:, 2, :] .= (ψ[:, 1, :] .- ψ[:, 2, :]) ./ sqrt(2)
    return reshape(A, :)
end
```

M. Plesch et al. “Quantum-state preparation with universal gate decompositions”. arXiv:1003.5760
F. Vatan et al. “Optimal quantum circuits for general two-qubit gates”. 10.1103/PhysRevA.69.032315
Y. Nam et al. “Automated optimization of large quantum circuits with continuous parameters”. arXiv:1710.07345v2

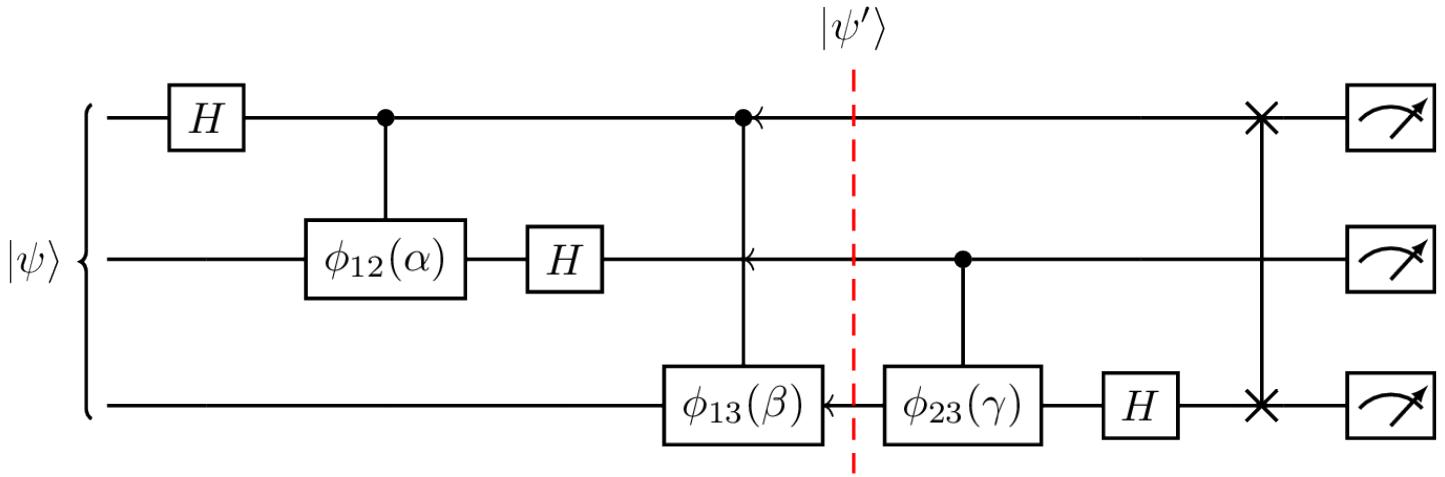
# Qaintessent.jl

- Supports statevector or density matrix simulation with standard gate sets and arbitrary unitary gates
- Supports unitary matrix decomposition
- Rudimentary graph-based optimization

```
julia> using Qaintessent; using Random; using RandomMatrices; using LinearAlgebra;

julia> r = Stewart(ComplexF64, 4); ψ = rand(ComplexF64, 4); ψ = ψ ./ norm(ψ); # create random unitary matrix and statevector

julia> meas = mop(Z, (1,)); # standard measurement on fastest running qubit

julia> c = Circuit{2}(unitary2circuit(r), [meas]) # create randomized 2-qubit operator

    2 -[Rz]—[Rx]—[Rz]—————————[X ]—[S†]———————•———————•———[Rz]—[S ]—[X ]—[Rz]—[Rx]—[Rz]—————————
                              |              |          |              |
    1 ————————[Rz]—[Rx]—[Rz]—•———[H ]—[S†]—[Rz]—[X ]—[Rz]—[X ]—[S ]—[H ]—•———————————[Rz]—[Rx]—[Rz]-


julia> apply(ψ, c) ≈ ψ'*r'*kron(I,Z)*r*ψ # check consistency
true
```

M. Plesch et al. “Quantum-state preparation with universal gate decompositions”. arXiv:1003.5760
F. Vatan et al. “Optimal quantum circuits for general two-qubit gates”. 10.1103/PhysRevA.69.032315
Y. Nam et al. “Automated optimization of large quantum circuits with continuous parameters”. arXiv:1710.07345v2

# Qaintellect.jl: Backward Pass




Customized adjoints calculate quantum states “on-the-fly” instead of storing them, idea from Yao.jl – extended to function with mixed states (density matrix representation).

Gradients of parametrized gates calculated via by tracing out unaffected qubits.

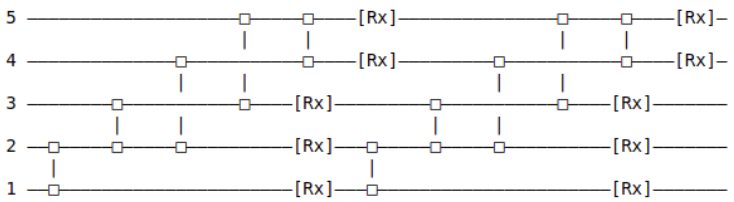
X. Luo et al. “Yao.jl: Extensible, efficient framework for quantum algorithm design”. arXiv:1912.10877.

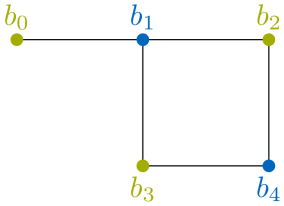
# Qaintellect.jl: QAOA Algorithm

In [2]:

```julia
# number of vertices
n = 5

# graph edges corresponding to the above graph
edges = [(0, 1), (1, 2), (1, 3), (2, 4), (3, 4)];

# assembles layers
function assemble_time_step_gates(p::Int, n::Int, edges::Vector{Tuple{Int,Int}})
    cgs = CircuitGate[]
    for _ in 1:p
        # C operator
        for e in edges
            # circuit gate uses 1-based indexing
            push!(cgs, circuit_gate(e[1] + 1, e[2] + 1, EntanglementZZGate(0.01*randn())))
        end
        # B operator
        for j in 1:n
           push!(cgs, circuit_gate(j, RxGate(0.01*randn())))
        end
    end
    return cgs
end

# create measurement operator representing C
Cmat = zeros(2^n, 2^n)
for edge in edges
    k1 = circuit_gate(edge[1] + 1, ZGate())
    k2 = circuit_gate(edge[2] + 1, ZGate())
    Cmat += 0.5*(I - sparse_matrix([k1, k2], n))
end

Cop = MeasurementOperator(Cmat, Tuple(1:n));

# example
circ = Circuit{n}(assemble_time_step_gates(2, n, edges), [Cop])
```

Out[2]:



# Qaintellect.jl: QAOA Algorithm

In [3]:

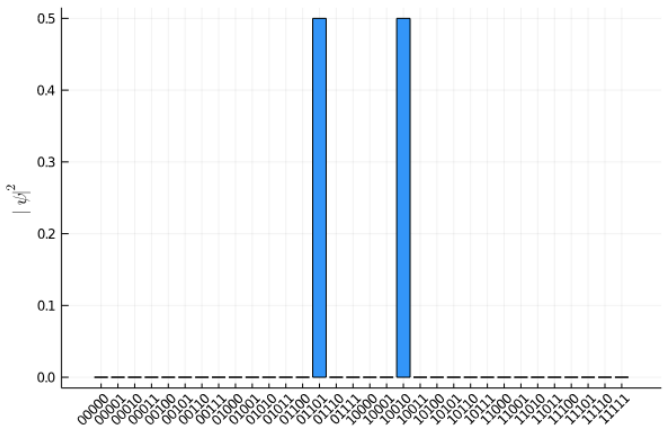
```
# gather parameters from circuit
paras = Flux.params(circ)

# there is not actually any input data for training
data = ncycle([()], 500)

# define optimizer
opt = Descent(0.5)

# define equal superposition state
s_uni = fill(1/√(2^n) + 0.0im, 2^n);

# define evaluation function
evalcb() = @show(apply(circ, s_uni))

# perform minimization with the negated target function to achieve maximization
Flux.train!(() -> -apply(circ, s_uni)[1], paras, data, opt, cb=Flux.throttle(evalcb, 0.5))
```

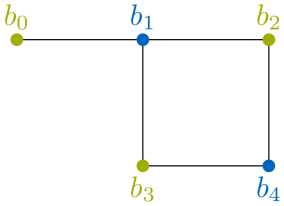
```
apply(circ, s_uni) = [2.500902440090582]
apply(circ, s_uni) = [4.999999759098869]
apply(circ, s_uni) = [4.999999999999966]
apply(circ, s_uni) = [5.000000000000001]
apply(circ, s_uni) = [4.9999999999999964]
```

In [4]:

```
# corresponding optimized quantum wavefunction
ψ2 = apply(circ.moments, s_uni);
tags = [join(reverse(digits(i, pad=n, base=2))) for i in 0:2^n-1]
bar(tags, abs2.(ψ2), xticks=:all, xrotation=45, ylabel=L"|\psi|^2", legend=false)
```

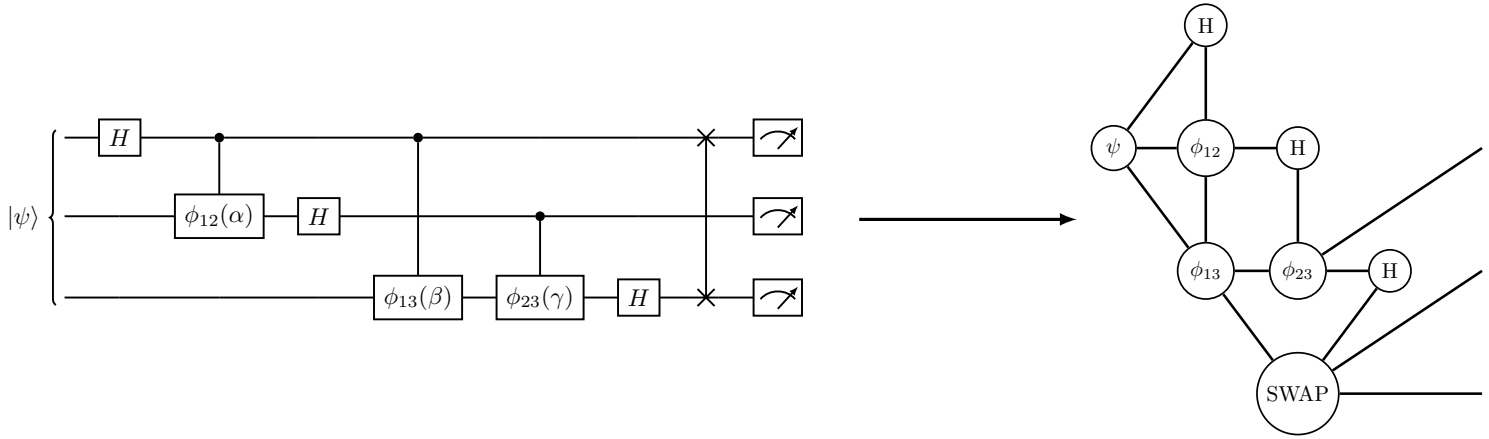
Out[4]:



# Qaintensor.jl

Supports conversion from arbitrary circuit/ unitary matrix to Tensor Network representation




```
julia> using Qaintessent; using Qaintensor; using LinearAlgebra; using Random; using RandomMatrices;

julia> gates = unitary2circuit(Stewart(ComplexF64, 4)); # create gates for randomized 2 qubit operator

julia> ψ = rand(ComplexF64, 4); ψ = ψ./norm(ψ); # create random 2 qubit statevector

julia> mps = MPS(ψ); tensor_circuit!(mps, gates); # convert to MPS format

julia> all(contract(mps)[:] .≈ apply(ψ, gates))
true
```

# Qaintum: planned features and future perspectives

- Improved circuit optimization
- Interface to actual quantum devices
- Long term goal: Full quantum computing software stack

# References


- Luo, Xiu-Zhe u. a. (2020). “Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design”. In: *Quantum* 4, S. 341. ISSN: 2521-327X. DOI: 10.22331/q-2020-10-11-341. URL: http://dx.doi.org/10.22331/q-2020-10-11-341.
- Nam, Yunseong u. a. (2018). “Automated optimization of large quantum circuits with continuous parameters”. In: *npj Quantum Information* 4.1. ISSN: 2056-6387. DOI: 10.1038/s41534-018-0072-4. URL: http://dx.doi.org/10.1038/s41534-018-0072-4.
- Plesch, Martin und Časlav Brukner (2011). “Quantum-state preparation with universal gate decompositions”. In: *Physical Review A* 83.3. ISSN: 1094-1622. DOI: 10.1103/physreva.83.032302. URL: http://dx.doi.org/10.1103/PhysRevA.83.032302.
- Schuld, Maria u. a. (2019). “Evaluating analytic gradients on quantum hardware”. In: *Physical Review A* 99.3. ISSN: 2469-9934. DOI: 10.1103/physreva.99.032331. URL: http://dx.doi.org/10.1103/PhysRevA.99.032331.
- Vatan, Farrokh und Colin Williams (2004). “Optimal quantum circuits for general two-qubit gates”. In: *Phys. Rev. A* 69 (3), S. 032315. DOI: 10.1103/PhysRevA.69.032315. URL: https://link.aps.org/doi/10.1103/PhysRevA.69.032315.