Free software for quantum computation

Piotr Gawron

Institute of Theoretical and Applied Informatics
Polish Academy of Sciences

KQIS AGH
13 March 2019 Kraków

Outline

Outline

Introduction—Free software

Free software actions for quantum computation

- Quantum Open Source Foundation
- Fosdem 19 Quantum computing track
- Xanadu.ai

Julia and quantum computing

- Introduction to Julia
- QuantumInformation.jl

Programming D-Wave Annealer

- Quantum annealing
- D-Wave annealer
- D-Wave software stack

Future work—interesting challenges

- Interesting goals to pursue

Free software

IT Giants conference AGH 2009

Free software

A program is free software if the program's users have the four essential freedoms:

- The freedom to **run** the program as you wish, for any purpose (freedom 0).
- The freedom to **study** how the program works, and **change** it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to **redistribute** copies so you can help others (freedom 2).
- The freedom to distribute copies of your **modified** versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

`https://www.gnu.org/philosophy/free-sw.en.html`

Outline

Quantum Open Source Foundation I

Mission

The Quantum Open Source Foundation [...] is charged to expand the role of open source software in quantum computing and improve the standardization and quality thereof.

The objective of QOSF is to:

- Foster **collaboration** between the quantum hardware and software developer communities;
- Provide financial **funding** for selected projects and travel awards for selected QOSF members and maintainers of open source quantum projects;
- Incentivize and support the **distribution of free and open information** regarding advances in **quantum software engineering** and quantum computing in general;

Quantum Open Source Foundation II

- Provide a **forum** for physicists, software developers, quantum hardware providers and other parties to discuss common problems and obstacles related to open quantum software engineering;
- Organize free and open **conferences**, **workshops** and **informational sessions** on quantum software engineering;
- **Convey the fundamental concepts of quantum computing** and quantum software engineering to the general public.

`www.qosf.org`

In short

Do not (only) write proofs,
let's code!

Software resources

List of Open Quantum Projects

`https://www.qosf.org/project_list/`

List of Quantum Computation simulators

`https://www.quantiki.org/wiki/list-qc-simulators`

Short report from Fosdem I

Quantum computing devroom

- When open source meets quantum computing, Tomas Babej
  Fingerhuth M, Babej T, Wittek P (2018) **Open source software in quantum computing. PLoS ONE 13(12): e0208561.**
  `https://doi.org/10.1371/journal.pone.0208561`
- Forest: An Open Source Quantum Software Development Kit, Robert Smith
  Open-sourcing of quilc (compiler) and qvm (quantum virtual machine)
- Delivering Practical Quantum Computing, Murray Thom
  A review of D-Wave Annealer applications
- D-Wave's Software Development Kit, Alexander Condello
  `dwave-ocean-sdk` **review**
- D-Wave Hybrid Framework, Radomir Stevanovic
  How to build complex samplers using `dwave-ocean-sdk`

Short report from Fosdem II

- What is IBMQ, Mark Mattingley-Scott
- Qutip: Quantum simulations and collaborative code development, Shahnawaz Ahmed
  A widely used quantum mechanics and computation modelling framework written in python
- Strawberry Fields - software for photonic quantum computing, Nathan Killoran
  Quantum optics based gate model computation framework by Xanadu
- PennyLane - Automatic differentiation and ML of QC, Josh Izaac
  Neural networks with quantum optical components
- Quantum Computing at Google and in the Cloud, Kevin D. Kissell
- Promotion of open source and role of standardization in QC, Panel Discussion
- Exponential speedup in progress, Mark Fingerhuth

Short report from Fosdem III

Quantum computing workshop

- Towards Practical Quantum Machine Learning with NISQAI, Ryan LaRose
- Bayesforge: Elevating the QC Stack, Henning Dekant
  Quantum/classical Bayesian networks software distribution
- An Open-Source General Compiler for Quantum Computers, Kaitlin Smith
  A new yet unreleased quantum compiler
- Julia programming language for quantum software development, Piotr Gawron
- QCL - A Programming Language for Quantum Computers, Andrew Savchenko
  The first programming language for quantum computers
- Curry: A probabilistic quantum programming language, Lucas Saldyt

Short report from Fosdem IV

- PyZX: Graph-theoretic optimization of quantum circuits, John van de Wetering
  A category theory based quantum circuits optimization
- An implementation of a classifier on Qiskit, Carsten Blank
- Through the RevKit v3 implementation, Bruno Schmitt
  Reversible logic synthesis tool extension for quantum computing
- Q-bug: Visualizing Quantum Circuits, Felix Tripier
- SimulaQron — a simulator for developing quantum internet software, Axel Dahlberg
  Software stack for quantum internet developed in Netherlands

StrawberryFields I

	CV	**Qubit**
Basic element	Qumodes	Qubits
Relevant operators	Quadratures $\hat{x}, \hat{p}$ Mode operators $\hat{a}, \hat{a}^\dagger$	Pauli operators $\hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_z$
Common states	Coherent states $\|\alpha\rangle$ Squeezed states $\|z\rangle$ Number states $\|n\rangle$	Pauli eigenstates $\|0/1\rangle, \|\pm\rangle, \|\pm i\rangle$
Common gates	Rotation, Displacement, Squeezing, Beamsplitter, Cubic Phase	Phase shift, Hadamard, CNOT, T-Gate
Common measurements	Homodyne $\|x_\phi\rangle\langle x_\phi\|$, Heterodyne $\frac{1}{\pi}\|\alpha\rangle\langle\alpha\|$, Photon-counting $\|n\rangle\langle n\|$	Pauli eigenstates $\|0/1\rangle\langle 0/1\|, \|\pm\rangle\langle\pm\|$, $\|\pm i\rangle\langle\pm i\|$

Table I: *Basic comparison of the CV and qubit settings.*

StrawberryFields II

State family	Displacement	Squeezing
Vacuum state $\|0\rangle$	$\alpha = 0$	$z = 0$
Coherent states $\|\alpha\rangle$	$\alpha \in \mathbb{C}$	$z = 0$
Squeezed states $\|z\rangle$	$\alpha = 0$	$z \in \mathbb{C}$
Displaced squeezed states $\|\alpha, z\rangle$	$\alpha \in \mathbb{C}$	$z \in \mathbb{C}$
$\hat{x}$ eigenstates $\|x\rangle$	$\alpha \in \mathbb{C}$, $x = 2\sqrt{\frac{\hbar}{2}}\mathrm{Re}(\alpha)$	$\phi = 0$, $r \to \infty$
$\hat{p}$ eigenstates $\|p\rangle$	$\alpha \in \mathbb{C}$, $p = 2\sqrt{\frac{\hbar}{2}}\mathrm{Im}(\alpha)$	$\phi = \pi$, $r \to \infty$
Fock states $\|n\rangle$	N.A.	N.A.

Table II: *Common single-mode pure states and their relation to the displacement and squeezing parameters. All listed families are Gaussian, except for the Fock states. The* $n = 0$ *Fock state is also the vacuum state.*

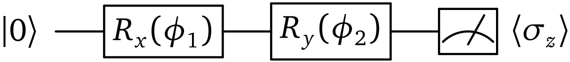
StrawberryFields III

Gate	Unitary	Symbol
Displacement	$D_i(\alpha) = \exp\left(\alpha \hat{a}_i^\dagger - \alpha^* \hat{a}_i\right)$	D
Rotation	$R_i(\phi) = \exp\left(i\phi \hat{n}_i\right)$	R
Squeezing	$S_i(z) = \exp\left(\frac{1}{2}(z^* \hat{a}_i^2 - z \hat{a}_i^{\dagger 2})\right)$	S
Beamsplitter	$BS_{ij}(\theta, \phi) =$ $\exp\left(\theta(e^{i\phi} \hat{a}_i \hat{a}_j^\dagger - e^{-i\phi} \hat{a}_i^\dagger \hat{a}_j)\right)$	BS
Cubic phase	$V_i(\gamma) = \exp\left(i \frac{\gamma}{3\hbar} \hat{x}_i^3\right)$	V

Table III: *Some important CV model gates. All listed gates except the cubic phase gate are Gaussian.*

Source: arXiv:1804.03159

PennyLane




```
import pennylane as qml
from pennylane.optimize import GradientDescentOptimizer
# Create device
dev = qml.device('default.qubit', wires=1)
# Quantum node
@qml.qnode(dev)
def circuit1(var):
    qml.RX(var[0], wires=0)
    qml.RY(var[1], wires=0)
    return qml.expval.PauliZ(0)
# Create optimizer
opt = GradientDescentOptimizer(0.25)
# Optimize circuit output
var = [0.1, 0.2]
for it in range(30):
    var = opt.step(circuit1, var)
    print("Step {}: cost: {}".format(it, circuit1(var)))
```

Source: arXiv:1811.04968

Outline

Julia

- Julia is a modern programming language focused on **numerical computing**.
- Julia is an **imperative**, **structural**, **dynamical**, **just-in-time compiled** programming language supporting **multiple dispatch**.
- Julia is equipped with a simple yet powerful type system consisting of **abstract** and **concrete**, potentially **parametrised**, types.
- Julia supports **meta programming** trough macros which allow for creation of domain specific languages.
- Julia natively supports **concurrent**, **parallel** and **distributed** computing models.

Julia and quantum computing

Julia vs. Python as a language for QC

- Julia solves **two languages problem** that exists with application of Python in numeric applications.
- Julia allows for **elegant notation** that closely resembles mathematical notation.
- Julia can be used as a **full-stack numerical computation language** able to handle and process petabytes of data therefore it is suitable to become the core element of **quantum computation infrastructure**.

Quantum landscape in Julia

Numerics

- QuantumOptics.jl — focused on quantum optics and open quantum systems.
- JuliaQuantum — ambitious, extensive but dead.

Quantum computation

- Yao.jl — A DSL for quantum computation.
- QuAlgorithmZoo.jl — implementation of a couple of quantum algorithms in Yao.jl

Raytheon BBN Technologies - Quantum Group

Has full stack for running their superconducting quantum system.

- QuantumInfo.jl, RandomQuantum.jl — strong overlap with our library.
- SchattenNorms.jl, Cliffords.jl, QSimulator.jl, ...

QuantumInformation.jl—goals

Goals

- Provide a simple numerical library for performing calculations in quantum information theory.
- Focus on mixed states and quantum channels.
- Provide fast and tested (!) generation methods of random quantum objects.
- Provide a wide selection of functionals (distances, entanglement measures, norms).

Gawron P, Kurzyk D, Pawela Ł (2018) QuantumInformation.jl—A Julia package for numerical computation in quantum information theory. PLoS ONE 13(12): e0209358.
`https://doi.org/10.1371/journal.pone.0209358`

QuantumInformation.jl—overview I

Listing 1: Quantum pure states are represented as 1d arrays. The inner product is expressed naturally.

```
julia> ψ=(1/sqrt(2)) * (ket(1,2) + ket(2,2))
2-element Array{Complex{Float64},1}:
 0.7071067811865475 + 0.0im
 0.7071067811865475 + 0.0im

julia>  ϕ=(1/2) * ket(1,2) + (sqrt(3)/2) * ket(2,2)
2-element Array{Complex{Float64},1}:
 0.5 + 0.0im
 0.8660254037844386 + 0.0im

julia> ϕ' * ψ
 0.9659258262890682 + 0.0im

julia> sqrt(ϕ' * ϕ)
 0.9999999999999999 + 0.0im
```

QuantumInformation.jl—overview II

Listing 2: Density matrices are 2d arrays.

```
julia> ρ = [0.25 0.25im; -0.25im 0.75]
2×2 Array{Complex{Float64},2}:
 0.25+0.0im   0.0+0.25im
-0.0-0.25im   0.75+0.0im

julia> σ = [0.4 0.1im; -0.1im 0.6]
2×2 Array{Complex{Float64},2}:
 0.4+0.0im   0.0+0.1im
-0.0-0.1im   0.6+0.0im

julia> ptrace(ρ ⊗ σ, [2, 2], [2])
2×2 Array{Complex{Float64},2}:
 0.25+0.0im    0.0+0.25im
 0.0-0.25im    0.75+0.0im
```

QuantumInformation.jl—overview III

Listing 3: Quantum Channels are in four representations, each having its own type.

```
julia> γ=0.4
 0.4

julia> K0 = Matrix([1 0; 0 sqrt(1-γ)])
2×2 Array{Float64,2}:
 1.0  0.0
 0.0  0.774597

julia> K1 = Matrix([0 sqrt(γ); 0 0])
2×2 Array{Float64,2}:
 0.0  0.632456
 0.0  0.0

julia> Φ = KrausOperators([K0,K1])
KrausOperators{Array{Float64,2}}
dimensions: (2, 2)
 [1.0 0.0; 0.0 0.774597]
 [0.0 0.632456; 0.0 0.0]

julia> iscptp(Φ)
 true
```

QuantumInformation.jl—overview IV

Listing 4: Convertions between chanels representations are implemented.

```
julia> Ψ1 = convert(SuperOperator{Matrix{ComplexF64}}, Φ)
SuperOperator{Array{Complex{Float64},2}}
dimensions: (2, 2)
Complex{Float64}
 [1.0+0.0im 0.0+0.0im 0.0+0.0im 0.4+0.0im;
 0.0+0.0im 0.774597+0.0im 0.0+0.0im 0.0+0.0im;
 0.0+0.0im 0.0+0.0im 0.774597+0.0im 0.0+0.0im;
 0.0+0.0im 0.0+0.0im 0.0+0.0im 0.6+0.0im]

julia> Ψ2 = convert(DynamicalMatrix{Matrix{Float64}}, Φ)
DynamicalMatrix{Array{Float64,2}}
dimensions: (2, 2)
 [1.0 0.0 0.0 0.774597;
 0.0 0.4 0.0 0.0;
 0.0 0.0 0.0 0.0;
 0.774597 0.00.0 0.6]

julia> Ψ3 = convert(Stinespring{Matrix{Float64}}, Φ)
Stinespring{Array{Float64,2}}
dimensions: (2, 2)
 [...]
```

QuantumInformation.jl—overview V

Listing 5: Channels can be composed in parallel and in series. Application of channels is done naturally.

```
julia> ρ2=ϕ * ϕ'
2×2 Array{Complex{Float64},2}:
 0.25+0.0im      0.433013+0.0im
 0.433013+0.0im  0.75+0.0im

julia> (Φ ⊗ Φ)(ρ1 ⊗ ρ2)
4×4 Array{Complex{Float64},2}:
 0.385+0.0im     0.234787+0.0im  0.213014+0.0im  0.129904+0.0im
 0.234787+0.0im  0.315+0.0im     0.129904+0.0im  0.174284+0.0im
 0.213014+0.0im  0.129904+0.0im  0.165+0.0im     0.100623+0.0im
 0.129904+0.0im  0.174284+0.0im  0.100623+0.0im  0.135+0.0im

julia> (Ψ1 ∘ Ψ2)(ρ1)
2×2 Transpose{Complex{Float64},Array{Complex{Float64},2}}:
0.82+0.0im   0.3+0.0im
0.3+0.0im  0.18+0.0im
```

QuantumInformation.jl—overview VI

Listing 6: A sub-package for random matrices is implmented. Random Hermitian matrices.

```
julia> g = GinibreEnsemble{2}(2,3)
 GinibreEnsemble{2}(m=2, n=3)

julia> rand(g)
2×3 Array{Complex{Float64},2}:
 0.835803+1.10758im   -0.622744-0.130165im  -0.677944+0.636562im
 1.32826+0.106582im   -0.460737-0.531975im  -0.656758+0.0244259im
```

QuantumInformation.jl—overview VII

Listing 7: Random unitaries.

```
julia>  c = CircularEnsemble{2}(3)
CircularEnsemble{2}(
d: 3
g: GinibreEnsemble{2}(m=3, n=3)
)

julia> u = rand(c)
3×3 Array{Complex{Float64},2}:
 0.339685+0.550434im  -0.392266-0.3216im      -0.53172+0.203988im
 0.515118-0.422262im   0.392165-0.626859im    -0.0504431-0.084009im
 0.297203+0.222832im  -0.418737-0.143578im     0.607012-0.545525im


julia> u*u'
3×3 Array{Complex{Float64},2}:
 1.0+0.0im                    -5.55112e-17-5.55112e-17im   -2.77556e-17-4.16334e-17
-5.55112e-17+5.55112e-17im     1.0+0.0im                   -2.498e-16+0.0im
-2.77556e-17+4.16334e-17im    -2.498e-16+0.0im              1.0+0.0im
```

QuantumInformation.jl—overview VIII

Listing 8: Random quantum pure sates.

```
julia> h = HaarKet{2}(3)
HaarKet{2}(d=3)

julia> ψ = rand(h)
3-element Array{Complex{Float64},1}:
 0.1687649644765863 - 0.3201009507269653im
 0.7187423269572294 - 0.3940502277043476 7im
 0.1342475675218075 + 0.4232791563609603 6im

julia> norm(ψ)
 1.0
```

QuantumInformation.jl—overview IX

Listing 9: Random quantum channels are returned in appropriate channel type.

```
julia> c = ChoiJamiolkowskiMatrices(2, 3)
ChoiJamiolkowskiMatrices{2,1}(WishartEnsemble{2,1}(d=6), 2, 3)

julia> Φ = rand(c)
DynamicalMatrix{Array{Complex{Float64},2}}
dimensions: (2, 3)
Complex{Float64}
 [0.307971-4.98733e-18im -0.00411588+0.0368471im...
-0.0676732+0.024328im  0.0860858+0.00302876im;
-0.00411588-0.0368471im 0.167651+2.1684e-19im...
-0.0428561+0.0266119im    0.0191888+0.0101013im;
...;
-0.0676732-0.024328im -0.0428561-0.0266119im...
 0.210419+0.0im -0.103401-0.142753im;
 0.0860858-0.00302876im 0.0191888-0.0101013im...
-0.103401+0.142753im 0.411068+0.0im]

julia> ptrace(Φ.matrix, [3, 2],[1])
2×2 Array{Complex{Float64},2}:
 1.0-1.53957e-17im            -1.38778e-17-3.05311e-16im
 1.38778e-17+3.05311e-16im      1.0+2.1684e-19im
```

Outline

Adiabatic model of quantum computation – the basics I

Classical Ising model

Let a classical Hamiltonian (energy function) be given:

$$H(s) = -\sum_{i\in\mathcal{I}} h_i s_i - \sum_{(i,j)\in\mathcal{I}\times\mathcal{I}} J_{ij} s_i s_j,$$

Where

$$s = [s_i]_{i\in\mathcal{I}} \in \{-1, 1\}^{\mathcal{I}}, h_i \in \mathbb{R}, J_{ij} \in \mathbb{R}.$$

The goal is to find

$$s^\star = \arg\min_s H(s),$$

the minimal energy state.

Adiabatic model of quantum computation – the basics II

Quantum Ising Hamiltonian

$$H(t) = (1-\frac{t}{\tau})\underbrace{\left(-\sum_{i\in\mathcal{I}}\sigma_x^{(i)}\right)}_{H_0}+\frac{t}{\tau}\underbrace{\left(-\sum_{i\in\mathcal{I}}h_i\sigma_z^{(i)}-\sum_{(i,j)\in\mathcal{I}\times\mathcal{I}}J_{ij}\sigma_z^{(i)}\sigma_z^{(j)}\right)}_{H_p},$$

where

$$\sigma_{\{x,z\}}^{(i)} = \mathbb{1}_2^{\otimes(i-1)}\otimes\sigma_{\{x,z\}}\otimes\mathbb{1}_2^{\otimes(|\mathcal{I}|-i-1)}$$

$$\mathbb{1}_2 = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}, \sigma_z = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}, \sigma_x = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}.$$

Adiabatic model of quantum computation – the basics III

Eigenstates of a Hamiltonian

Hamiltonians have eigenvalues E_n ($E_0 \leq E_1 \leq \ldots \leq E_n$) and corresponding eigenstates $|\psi\rangle_n$:

$$H\,|\psi\rangle_n = E_n\,|\psi\rangle_n\,.$$

If we will begin computation in the state

$$\left|\psi^{(0)}\right\rangle_0 : H_0 \left|\psi^{(0)}\right\rangle_0 = E_0^{(0)} \left|\psi^{(0)}\right\rangle_0 = \left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)^{\otimes|\mathcal{I}|},$$

then for large τ we will end up in the state:

$$\left|\psi^{(p)}\right\rangle_0 : H_p \left|\psi^{(p)}\right\rangle_0 = E_0^{(p)} \left|\psi^{(p)}\right\rangle_0\,.$$

Adiabatic model of quantum computation – the basics IV

Finally we perform a measurement:

$$\left\{ P_{\pm 1}^{\otimes|\mathcal{I}|} \right\},$$

where

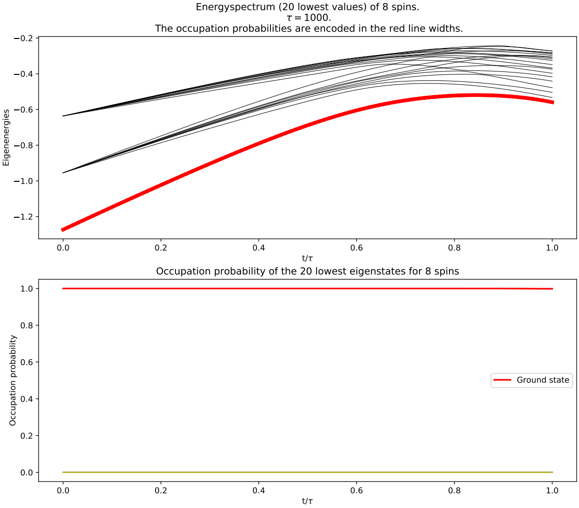
$$\{P_{-1} = |0\rangle\langle 0|\,, P_1 = |1\rangle\langle 1|\}.$$

As a result we obtain:

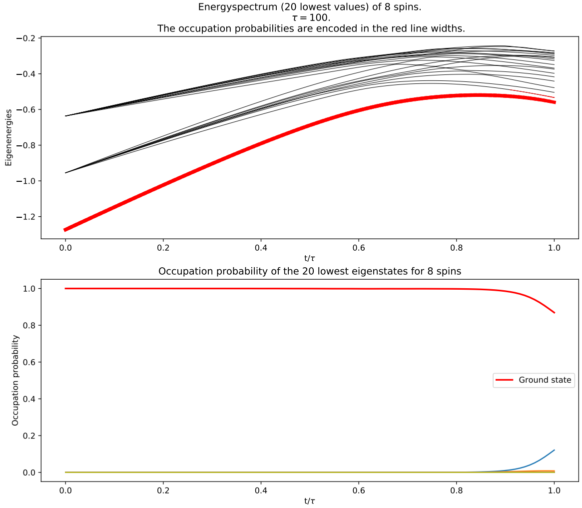
$$s = \{\pm 1\}^{|\mathcal{I}|},$$

what is the result of our minimization problem.

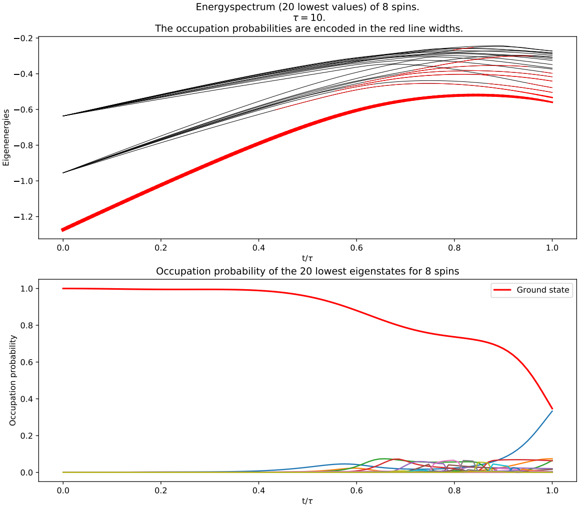
Adiabatic evolution – example



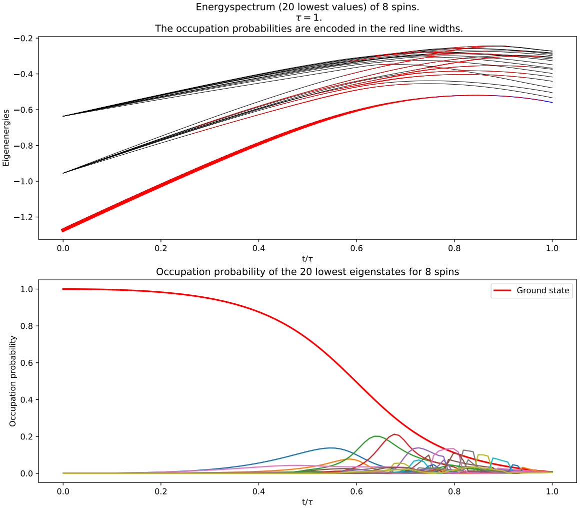
Adiabatic evolution – example



Adiabatic evolution – example



Adiabatic evolution – example



D-Wave annealer

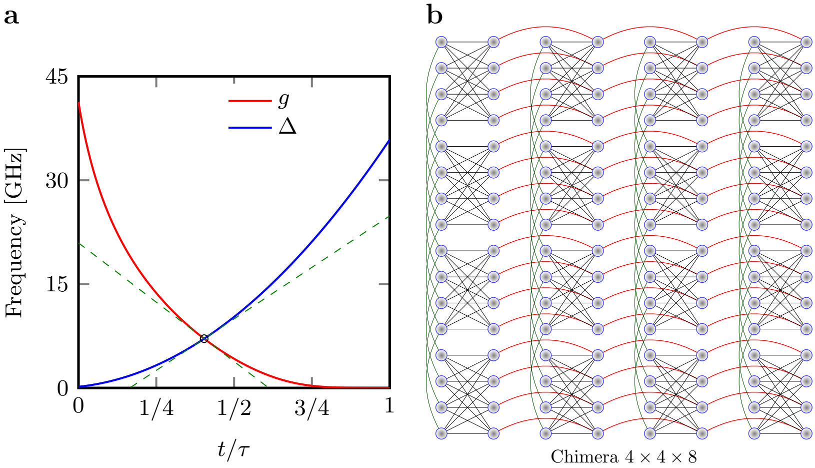
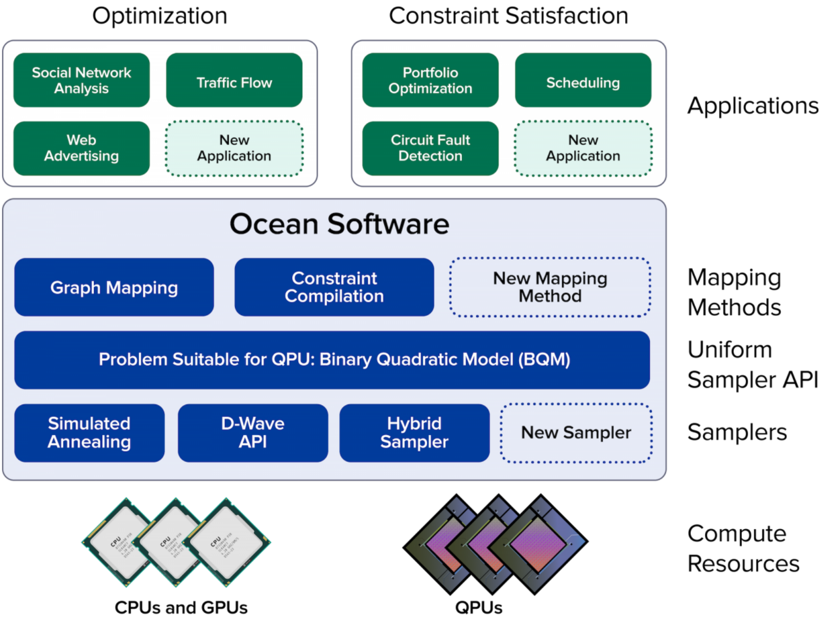


Figure: $\mathcal{H}(t)/(2\pi\hbar) = -g(t)\sum_i \sigma_x^{(i)} - \Delta(t)\mathcal{H}_1, \quad t \in [0, \tau].$

Source: Bartłomiej Gardas



Source: Copyright © D-Wave Systems Inc.

Example of dimod application

Program:

```
import dimod
from dwave.system.samplers import DWaveSampler

J = {("a", "b"): -1.0, ("a", "c"): -0.5, ("c", "a"): 0.1}
h = {"a": 0, "b": -1, "c":0.5}

bqm = dimod.BinaryQuadraticModel.from_ising(h, J)
sampler = dimod.ExactSolver() # or DWaveSampler()
response = sampler.sample(bqm)
for datum in response.data(['sample', 'energy']):
    print(datum.sample, datum.energy)
```

Output:

```
{'a': 1, 'b': 1, 'c': -1} -2.1
{'a': 1, 'b': 1, 'c': 1} -1.9
{'a': -1, 'b': -1, 'c': -1} -0.9
{'a': -1, 'b': 1, 'c': -1} -0.9
{'a': -1, 'b': 1, 'c': 1} 0.9
{'a': -1, 'b': -1, 'c': 1} 0.9
{'a': 1, 'b': -1, 'c': -1} 1.9
{'a': 1, 'b': -1, 'c': 1} 2.1
```

Outline

Interesting goals to pursue I

QuantumInformation.jl

- Quantum channel composition using **tensor networks** (help needed!).
- Managing the **rank of quantum channels** in order to speed-up computation.
- Adding support for **sparse arrays** (moderate difficulty).
- Clean-up and enhancements.

Interesting goals to pursue II

AcausalNets.jl

- Finishing, polishing and publishing
  https://github.com/mikegpl/AcausalNets.jl
- Possible cooperation with http://artiste-qb.net/
  https://github.com/artiste-qb-net

Interesting goals to pursue III

Ising samplers Julia stack

- Existing project ThreeQ.jl: `https://github.com/omalled/ThreeQ.jl`.
- Maybe a new project with well developed type system is more suitable (moderate difficulty — MSc level).

Interesting goals to pursue IV

Assessment of existing gate model for quantum computation Julia stack

- QuantumBFS/Yao.jl `https://github.com/QuantumBFS/Yao.jl`.
- Quantum Bayesian networks to Yao.jl compiler (difficult — PhD level project).
- Quantum Gate Language compiler `https://github.com/BBN-Q/QGL.jl`.
- Quantum gate model compiler for dedicated architectures (difficult — PhD level project).

Interesting goals to pursue V

A general gate model compiler

- Based on quilc `https://github.com/rigetti/quilc` (very difficult).

Interesting goals to pursue VI

Quantum Computation Language to QASM / QUIL compiler

- Based on qcl2qml `https://github.com/ZKSI/qcl2qml` (easy).

Interesting goals to pursue VII

A general question

- How to **integrate** quantum computation with super-computing infrastructure for big data processing?

!!!

Thank you for your attention!

Questions?

Websites, e-mail

`www.quantiki.org`
`github.com/ZKSI/QuantumInformation.jl`
p.w.gawron@gmail.com