Architectures for Hybrid Quantum/Classical Computing

Will Zeng

ORNL 2017 05.25.2017

rigetti

Outline

> Near-term quantum computers & What they can and can't do

> An architectural outline for hybrid classical/quantum computing

  > The Quantum Instruction Language (Quil) for hybrid computing

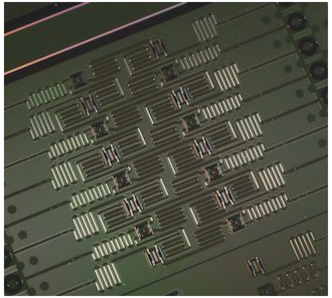
  > Intro to Higher-level programming with pyQuil

> A worked example: QAOA for MAXCUT compiling from pyQuil down to the metal

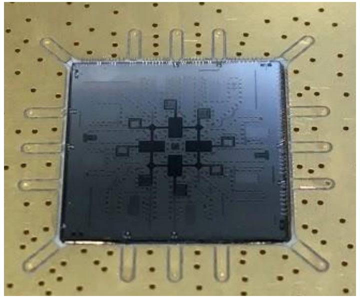
> Example open problems for collaboration:

  > Routing, generic unitary compilation, high-performance noisy simulation, and classical integration

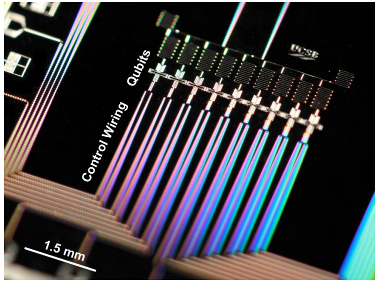
Near-term Quantum Computers



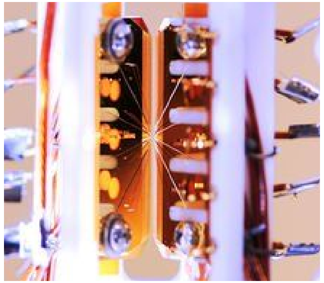
IBM



Rigetti



Google



IonQ / UMD

<u>Tens to low hundreds of physical qubits</u>

> Nearest-neighbor lattices: superconducting qubits
> Finite fidelities
> Measurable cross-talk
> Typically capable of approximately parameterized gates
> Fast-feedback limitations
> Limited Error-correction

What we can’t do near-term

> Shor’s algorithm (of order 10^8 qubits c.f. Fowler et al. 1208.0928)

> Anything with a qRAM

> Grover’s search

> Exact Hamiltonian Simulation

> Fault-tolerant quantum computation

What we can do: hybrid pre-threshold algorithms

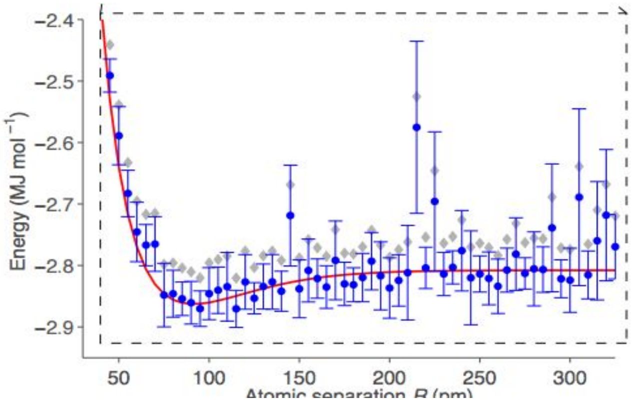
> Variational Quantum Eigensolver

> Quantum Approximate Optimization Algorithm

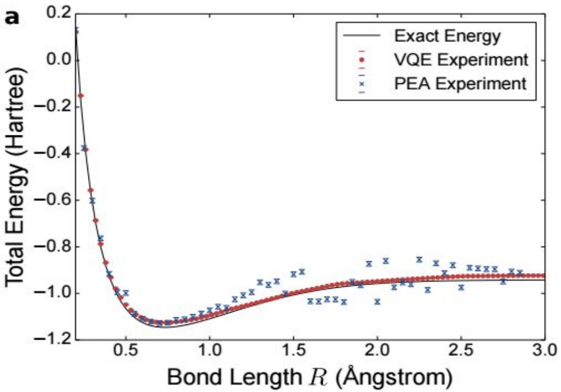
What we can do: hybrid pre-threshold algorithms

> Variational Quantum Eigensolver

> Quantum Approximate Optimization Algorithm

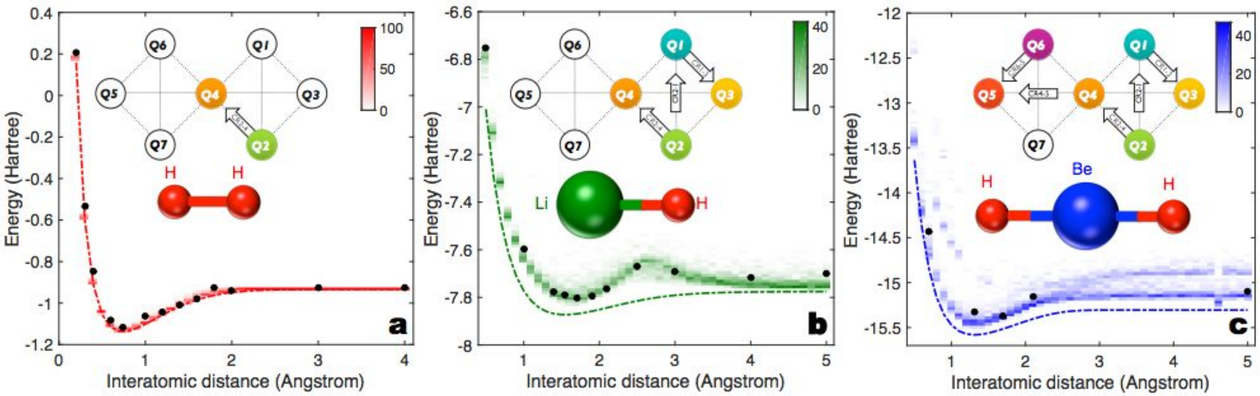


Peruzzo et al. 1304.3061



O'Malley et al. 1512.06860

Kandala et al.
1704.05018



Variational Quantum Eigensolver

1. MOLECULAR DESCRIPTION

e.g. Electronic Structure Hamiltonian

$$H = \sum_{i,j<i}^{N_n} \frac{Z_i Z_j}{|R_i - R_j|} + \sum_{i=1}^{N_e} \frac{-\nabla_{r_i}^2}{2} - \sum_{ij}^{N_n, N_e} \frac{Z_i}{|R_i - r_j|} + \sum_{i,j<i}^{N_e} \frac{1}{|r_i - r_j|}.$$

2. MAP TO QUBIT REPRESENTATION

e.g. Bravyi-Kitaev or Jordan-Wigner Transform

e.g. DI-HYDROGEN

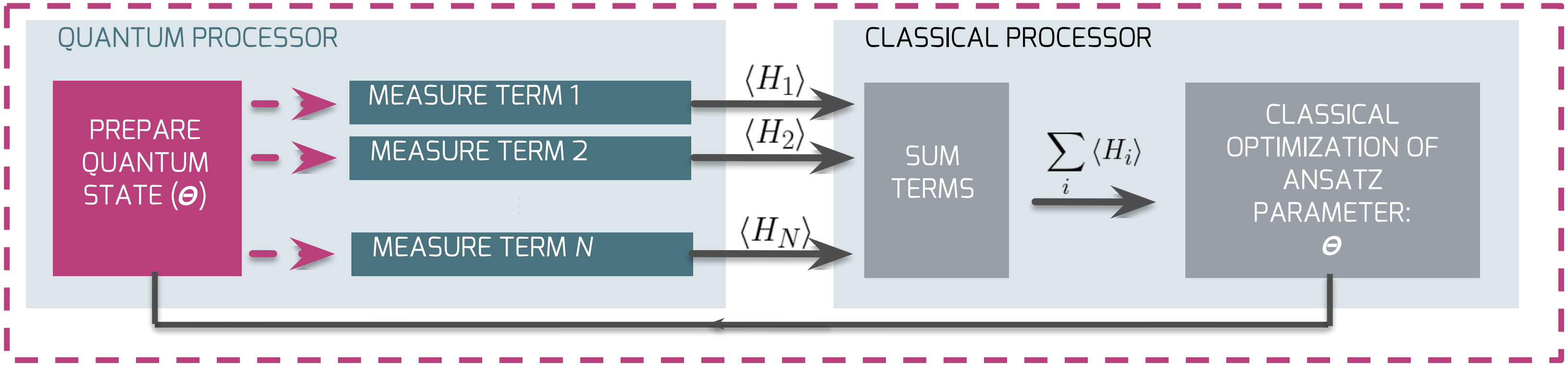
$$\begin{aligned} H = {} & f_0 \mathbb{1} + f_1 Z_0 + f_2 Z_1 + f_3 Z_2 + f_1 Z_0 Z_1 \\ & + f_4 Z_0 Z_2 + f_5 Z_1 Z_3 + f_6 X_0 Z_1 X_2 + f_6 Y_0 Z_1 Y_2 \\ & + f_7 Z_0 Z_1 Z_2 + f_4 Z_0 Z_2 Z_3 + f_3 Z_1 Z_2 Z_3 \\ & + f_6 X_0 Z_1 X_2 Z_3 + f_6 Y_0 Z_1 Y_2 Z_3 + f_7 Z_0 Z_1 Z_2 Z_3 \end{aligned}$$

3. PARAMETERIZED ANSATZ

e.g. Unitary Coupled Cluster
Variational Adiabatic Ansatz

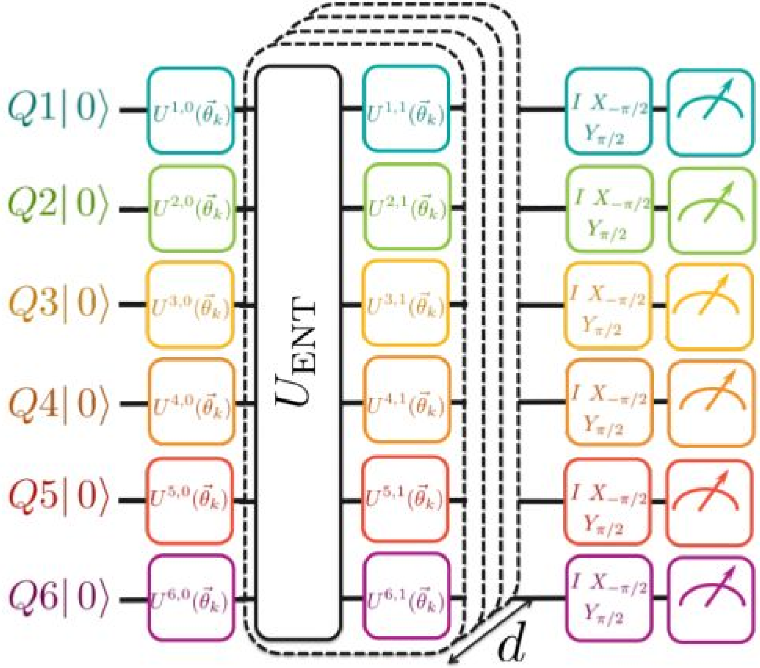
$$\frac{\langle \varphi(\vec{\theta})| \, H \, |\varphi(\vec{\theta})\rangle}{\langle \varphi(\vec{\theta}) | \varphi(\vec{\theta})\rangle} \geq E_0$$

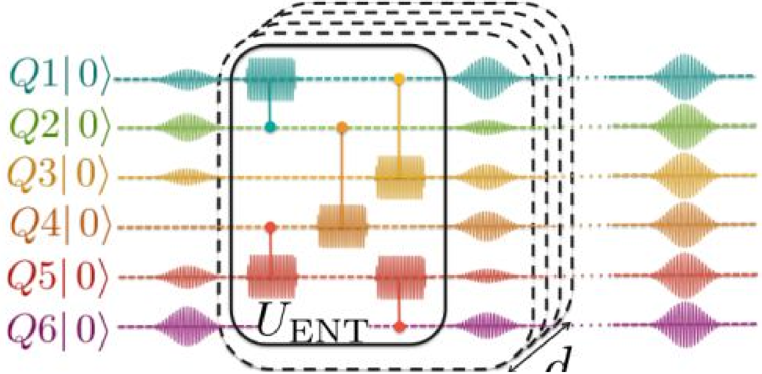
4. RUN Q.V.E. QUANTUM-CLASSICAL HYBRID ALGORITHM



O'Malley, P. J. J., et al. (2015). Scalable Quantum Simulation of Molecular Energies. *arXiv:1512.06860*.
Wecker, D., et al. (2015). Progress towards practical quantum variational algorithms. *Physical Review A*, *92*(4), 042303.
McClean, J. R. et al. (2015). The theory of variational hybrid quantum-classical algorithms. *arXiv:1509.04279*.
Peruzzo, A., et al. (2014). A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, *5*.

Variational Quantum Eigensolver





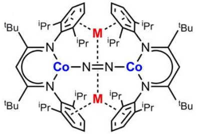
Kandala et al. 1704.05018

Challenge in near-term quantum algorithms

Up to low hundreds of noisy physical qubits

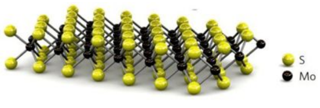
Quantum Simulation
Variational Quantum Eigensolvers [1,2,3,4]

Catalysts



N_2-activation, H_2O-O_2, etc

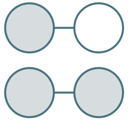
Complex Materials [5]



High-T_C, dichalcogenides, etc

Quantum Optimization
QAOA [6,7]

MAX-CUT

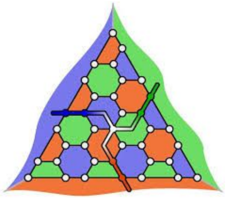


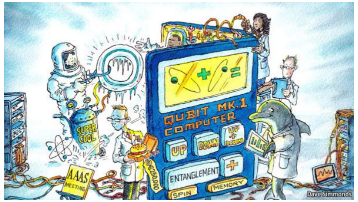
Score +1

Score 0

Max-Cut cost operator $\hat{C}_{ij} = \frac{1}{2}\left(\mathbf{I} - \sigma_i^Z \sigma_j^Z\right)$

QCVV & QEC
Topological Codes, GST, RB, CSS Codes, etc. [8, 9, 10, 11, 12]





What are the gate counts / resource reqs?

How do they behave under noise?

How do we optimize them?

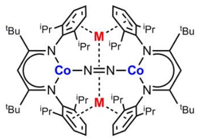
[1] Peruzzo et al. "A variational quantum eigensolver on a photonic quantum processor." *Nature Communications*, vol. 5, 2014
[2] O'Malley et al. "Scalable quantum simulation of molecular energies." *Phys. Rev. X*, vol. 6, p. 031007, July 2016
[3] Mc … rid quantum-cl … *ics*, vol. 18, p. 0
[4] N. … roach for large … density matrix … 16
[5] Ba … oach to correlat … 045, Sep 2016
[6] Fa … zation algorithm
[7] Fa … zation algorithm … int problem." …
[8] Ch … ar lattice for an … *Technology+Ap* … s and
Photo
[9] Ke … error detection … *ature*, vol. 519
[10] G … all experiments.
[11] Nickerson. "Error correcting power of small topological codes." *arXiv:1609.01753*
[12] Erik, LSaldyt, Jonathan Gross, Travis Scholten, kmrudin, tjproct, & David Nadlinger. (2017). pyGSTio/pyGSTi: Version 0.9.3 [Data set]. Zenodo. http://doi.org/10.5281/zenodo.269609
…and more!

Challenge in near-term quantum algorithms

Up to low hundreds of noisy physical qubits

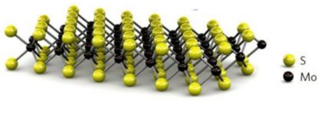
Quantum Simulation
Variational Quantum Eigensolvers [1,2,3,4]

Catalysts



N_2-activation, H_2O-O_2, etc

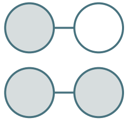
Complex Materials [5]



High-T_C, dichalcogenides, etc

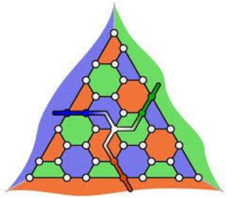
Quantum Optimization
QAOA [6,7]

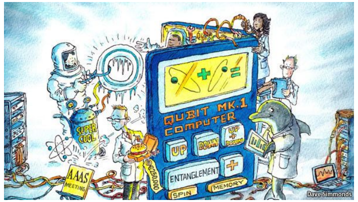
MAX-CUT



Max-Cut cost operator $\hat{C}_{ij} = \frac{1}{2}\left(\mathbf{I} - \sigma_i^Z \sigma_j^Z\right)$

QCVV & QEC
Topological Codes, GST, RB, CSS Codes, etc. [8, 9, 10, 11, 12]





[1] Peruzzo et al. "A variational quantum eigensolver on a photonic quantum processor." *Nature Communications*, vol. 5, 2014
[2] O'Malley et al. "Scalable quantum simulation of molecular energies." *Phys. Rev. X*, vol. 6, p. 031007, July 2016
[3] Mc … rid quantum-cl … ics, vol. 18, p. 0
[4] N. … proach for large … density matrix … 16
[5] Ba … oach to correlat … 045, Sep 2016
[6] Fa … zation algorithm
[7] Fa … zation algorithm … int problem." …
[8] Ch … ar lattice for ar … Technology+Ap … s and Photo
[9] Ke … error detection … *ature*, vol. 519
[10] G … ll experiments.
[11] Nickerson. "Error correcting power of small topological codes." *arXiv:1609.01753*
[12] Erik, LSaldyt, Jonathan Gross, Travis Scholten, kmrudin, tjproct, & David Nadlinger. (2017). pyGSTio/pyGSTi: Version 0.9.3 [Data set]. Zenodo. http://doi.org/10.5281/zenodo.269609
…and more!

What are the gate counts / resource reqs?

How do they behave under noise?

How do we optimize them?

How do we program near-term systems for near-term algorithms?

FOREST: A stack for classical/quantum hybrid programming

forest.rigetti.com

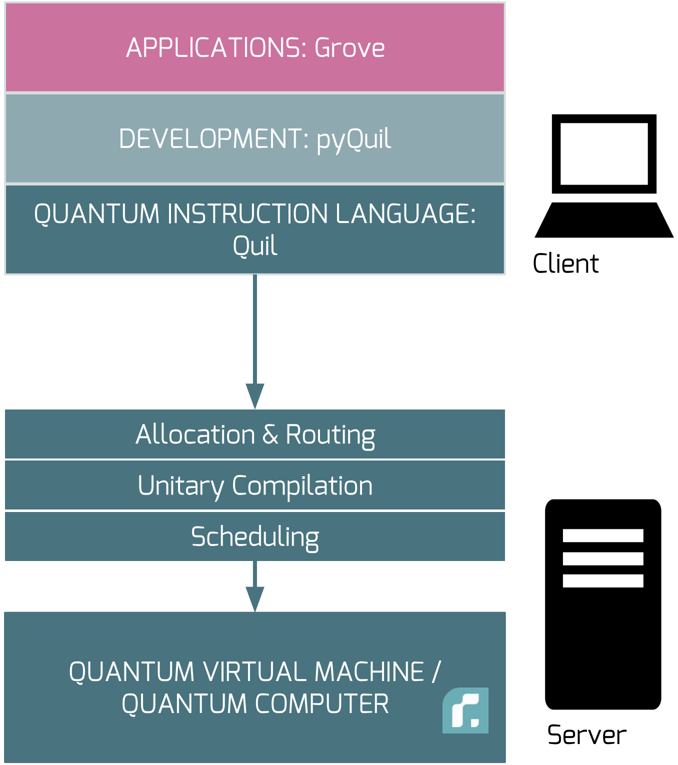
> Write applications...

> using tools...

> that build quantum programs...

> that compile onto quantum hardware...

> that execute on a real or virtual quantum processor.



Open-sourced on github under Apache v2.0 license

github.com/rigetticomputing/pyquil

github.com/rigetticomputing/grove

Open for private-beta signups

forest.rigetti.com

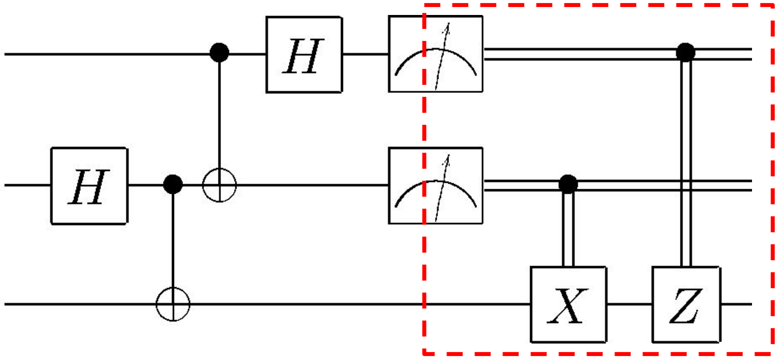
Quil and the Quantum Abstract Machine

A hybrid classical/quantum programming model.

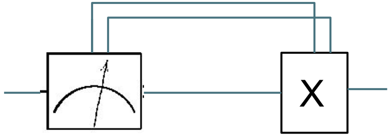
Quil is **portable**.

Quil is **portable, foundational**.

Quil is **portable, foundational, hybrid**.

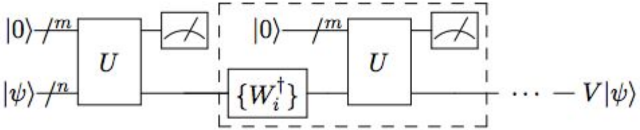


Fast Reset



Riste & DiCarlo. *Digital Feedback in Superconducting Quantum Circuits.* 1508.01385

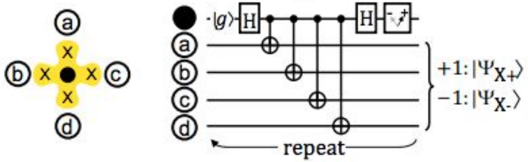
Repeat-until-success



Wiebe & Roettler. *Quantum arithmetic and numerical analysis using Repeat-Until-Success circuits.* 1406.2040

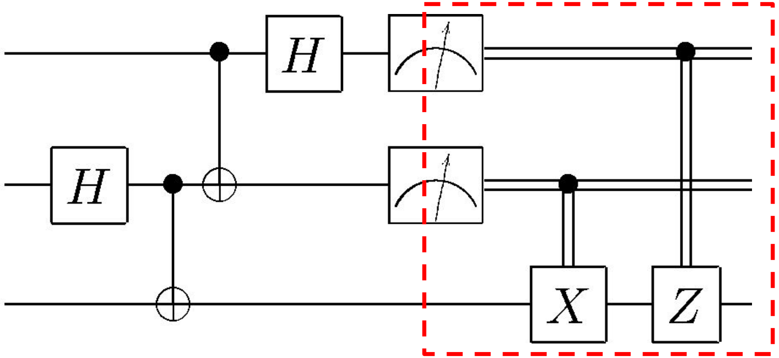
Bocharev et al. *Efficient Synthesis of Universal Repeat-Until-Success Circuits.* 1404.5320

Quantum Error Correction



Fowler et al. *Surface codes: Towards practical large-scale quantum computation.* 1208.0928

Quil is **portable, foundational, hybrid**.



The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine
> It is a syntax for representing state transitions.

The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine

> It is a syntax for representing state transitions.

Ψ: Quantum state (qubits) → quantum instructions

C: Classical state (bits) → classical and measurement instructions

κ: Execution state (program) → control instructions (e.g., jumps)

```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```

The Quil Programming Model

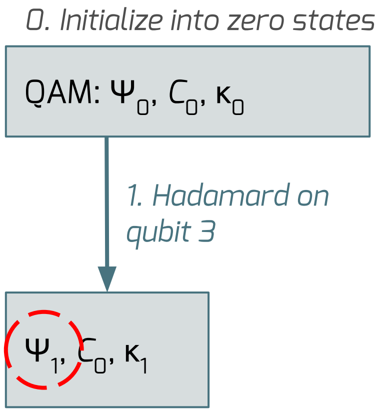
Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine
> It is a syntax for representing state transitions.

Ψ: Quantum state (qubits) → quantum instructions

C: Classical state (bits) → classical and measurement instructions

κ: Execution state (program) → control instructions (e.g., jumps)




```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```

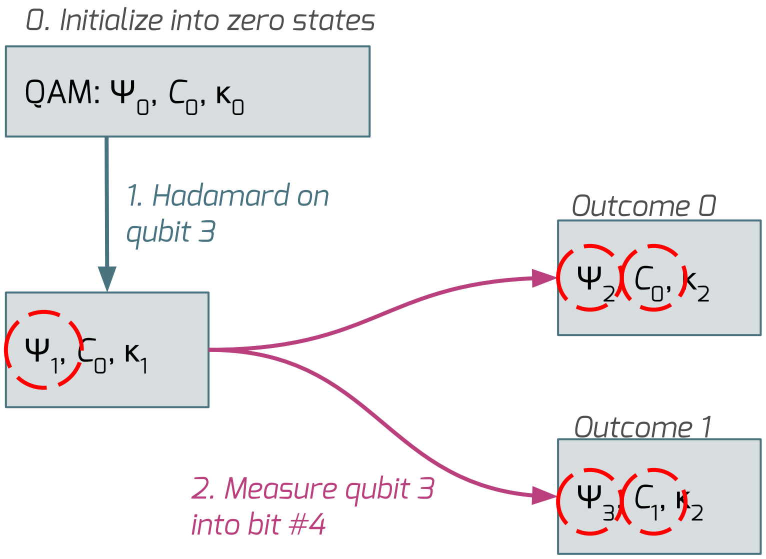
The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine
> It is a syntax for representing state transitions.

Ψ: Quantum state (qubits) → quantum instructions
C: Classical state (bits) → classical and measurement instructions
κ: Execution state (program) → control instructions (e.g., jumps)

```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```



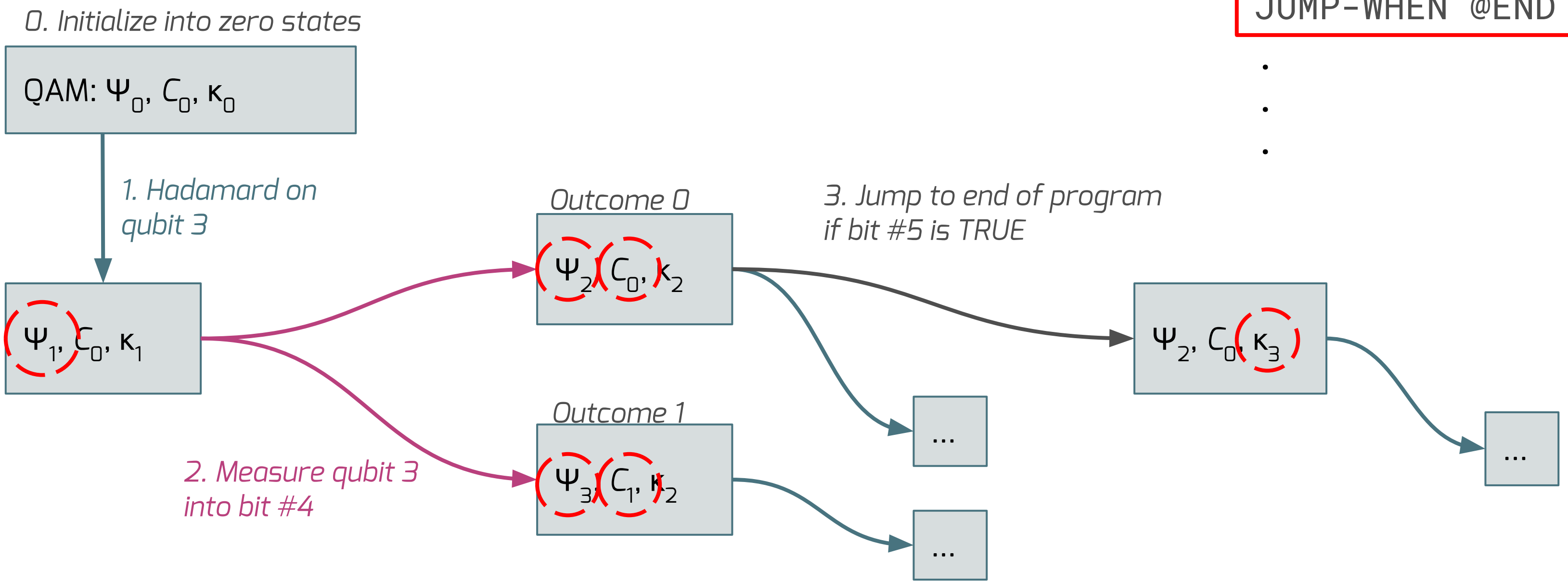
The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine
> It is a syntax for representing state transitions.

Ψ: Quantum state (qubits) → quantum instructions
C: Classical state (bits) → classical and measurement instructions
κ: Execution state (program) → control instructions (e.g., jumps)

```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```

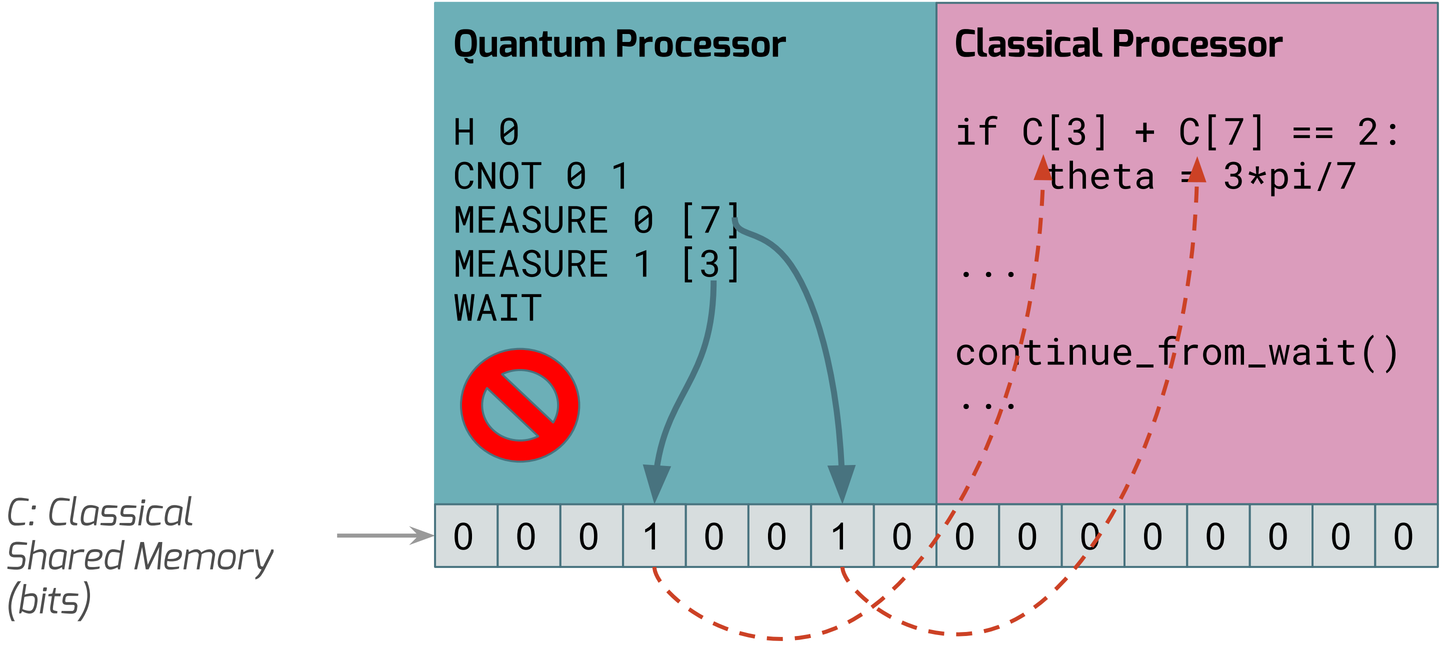


Interacting with a Classical Computer

> The Quantum Abstract Machine has a **shared classical state**.

> The QAM becomes a practical device with this shared state.

> Classical computers can take over with classical/quantum synchronization.



Formal Details: The Quil White Paper

For more Quil information see our updated white paper arXiv:1608.03355

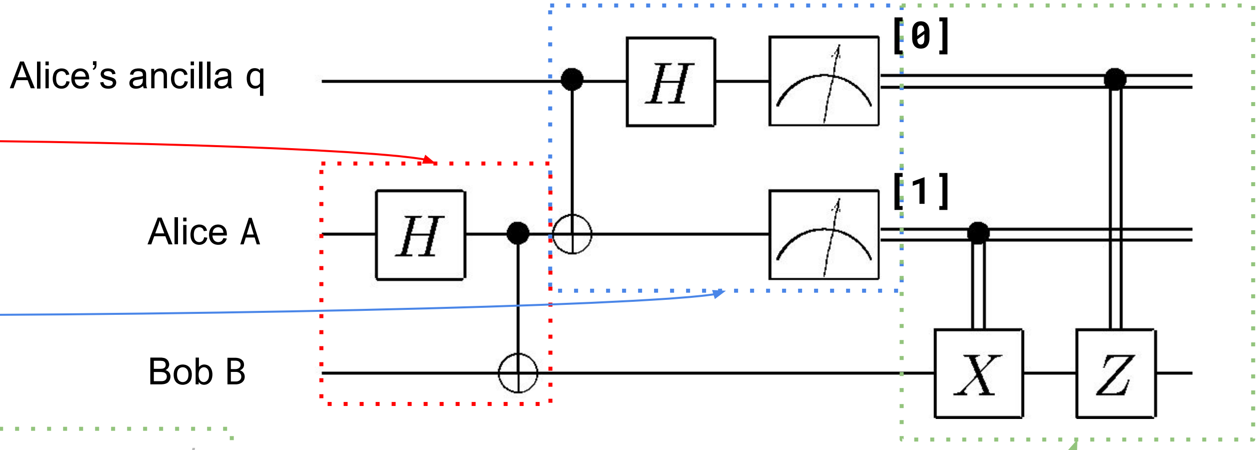
A Practical Quantum Instruction Set Architecture

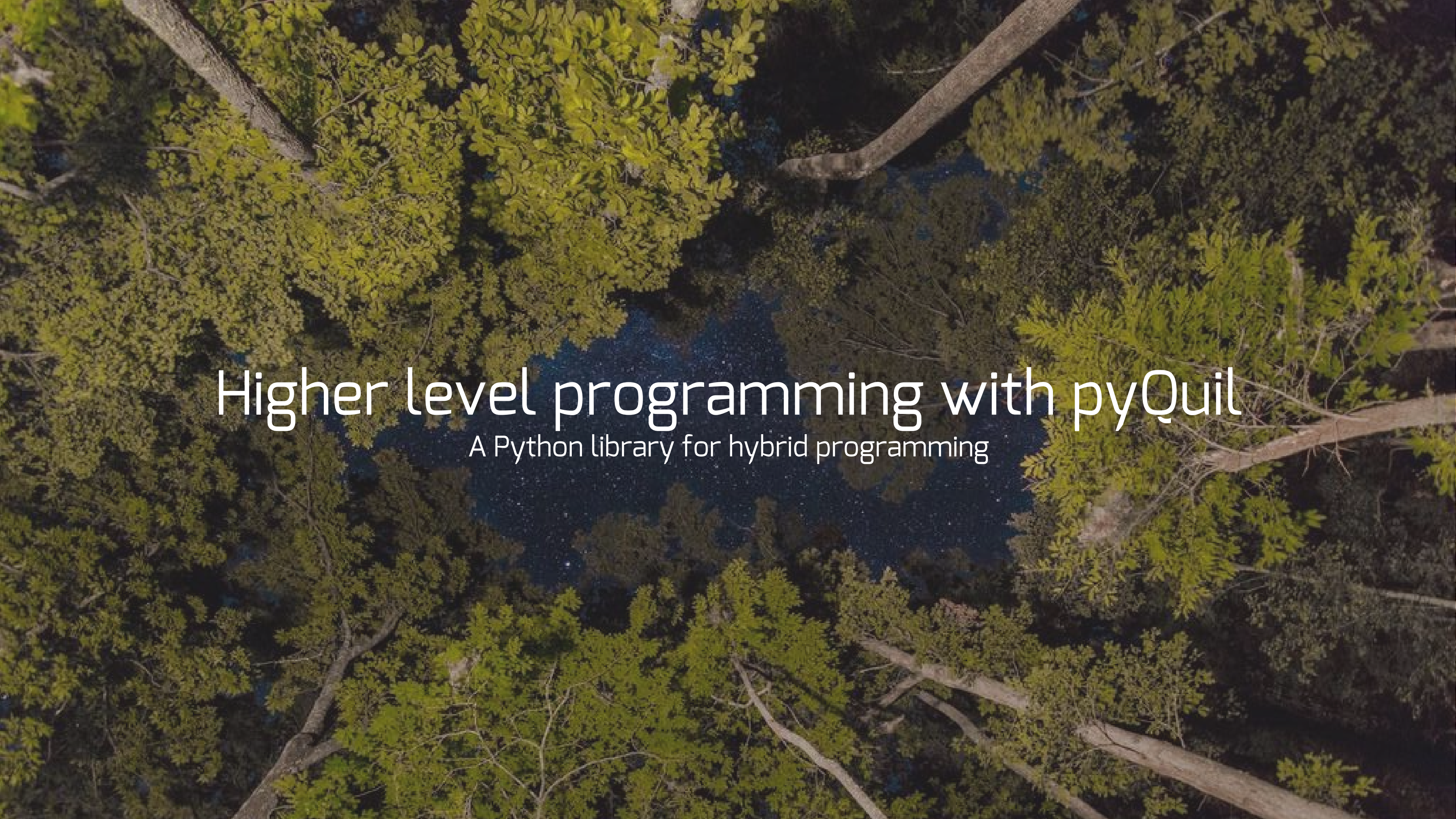
Robert S. Smith, Michael J. Curtis, William J. Zeng
Rigetti Computing
775 Heinz Ave.
Berkeley, California 94710
Email: {robert, spike, will}@rigetti.com

***Abstract*—Quantum computing technology has advanced rapidly in the last few years. Physical systems—superconducting qubits in particular—promise scalable gate-based hardware. Alongside these advances, new algorithms have been discovered that are adapted to the relatively smaller, noisier hardware that will become available in the next few years. These tend to be hybrid classical/quantum algorithms, where the quantum hardware is used in a co-processor model. Here, we introduce an abstract machine architecture for describing these algorithms, along with a language for representing computations on this machine, and discuss a classically simulable implementation architecture.** ***Keywords*—quantum computing, software architecture**

Quantum Teleportation in Quil

```
DEFCIRCUIT TELEPORT A q B:
    # Bell pair
    H           A
    CNOT        A B

    # Teleport
    CNOT        q A
    H           q
    MEASURE     q [0]
    MEASURE     A [1]

    # Classically communicate measurements
    JUMP-UNLESS @SKIP [1]
    X B
    LABEL @SKIP
    JUMP-UNLESS @END [0]
    Z B
    LABEL @END

# If Alice's qubits are 0 and 1
# and Bob's is 5
TELEPORT 0 1 5
```





Higher level programming with pyQuil

A Python library for hybrid programming

MAXCUT on a near-term quantum computer

Compiling the QAOA hybrid algorithm down to the metal

FOREST: Tools for experimental quantum programming

http://grove-docs.readthedocs.io/en/latest/qaoa.html

Grove

latest

Search docs

- Installation and Getting Started
- Quantum Teleportation
- Variational-Quantum-Eigensolver (VQE)
- ⊟ Quantum Approximate Optimization Algorithm (QAOA)
  - Overview
  - Cost Functions
  - Quickstart Examples
  - ⊞ Algorithm and Details
  - ⊞ Source Code Docs
- Quantum Fourier Transform (QFT)
- Phase Estimation Algorithm

Docs » Quantum Approximate Optimization Algorithm (QAOA)

Edit on GitHub

Quantum Approximate Optimization Algorithm (QAOA)

Overview

pyQAOA is a Python module for running the Quantum Approximate Optimization Algorithm on an instance of a quantum abstract machine.

The pyQAOA package contains separate modules for each type of problem instance: MAX-CUT, graph partitioning, etc. For each problem instance the user specifies the driver Hamiltonian, cost Hamiltonian, and the approximation order of the algorithm.

`qaoa.py` contains the base QAOA class and routines for finding optimal rotation angles via Grove's variational-quantum-eigensolver method.

The Quantum Approximate Optimization Algorithm

QAOA: kwaah-waah

> A Hybrid-Quantum Classical Algorithm: Farhi et al. 2014 (1411.4028)

- Quantum co-processor algorithm (like VQE)
- Noise tolerant

> Can demonstrate quantum supremacy: Farhi & Harrow 2016 (1602.07674)

> Similar to Digitized Quantum Annealing: Barends et al. 2015 (1511.03316)

The Quantum Approximate Optimization Algorithm

QAOA: kwaah-waah

> A Hybrid-Quantum Classical Algorithm: Farhi et al. 2014 (1411.4028)

- Quantum co-processor algorithm (like VQE)
- Noise tolerant

> Can demonstrate quantum supremacy: Farhi & Harrow 2016 (1602.07674)

> Similar to Digitized Quantum Annealing: Barends et al. 2015 (1511.03316)

THE PROBLEM

Constraint Satisfaction Problems: → MAXIMIZE

$z \in \{0,1\}^n$

$$C_a(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the constraint } a \\ 0 & \text{if } z \text{ does not .} \end{cases}$$

$$\boxed{C(z) = \sum_{a=1}^{m} C_a(z)}$$

The Quantum Approximate Optimization Algorithm

QAOA: kwaah-waah

> Constraint Satisfaction Problems: → MAXIMIZE

$z \in \{0,1\}^n$

$$C_a(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the constraint } a \\ 0 & \text{if } z \text{ does not .} \end{cases}$$

$$\boxed{C(z) = \sum_{a=1}^{m} C_a(z)}$$

The Quantum Approximate Optimization Algorithm

QAOA: kwaah-waah

> Constraint Satisfaction Problems: $\longrightarrow$ MAXIMIZE

$z \in \{0,1\}^n$

$$C_a(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the constraint } a \\ 0 & \text{if } z \text{ does not .} \end{cases}$$

$$\boxed{C(z) = \sum_{a=1}^{m} C_a(z)}$$

$$\hat{H}_s = (1-s)\hat{H}_{\text{Driver}} + s\hat{H}_{\text{Cost}}$$

$$e^{-i\hat{H}_s dt} = \underbrace{e^{-i(1-s)H_X dt}}_{U(\hat{H}_{\text{Driver}},\, \beta_s)} \underbrace{e^{-isH_Z dt}}_{U(\hat{H}_{\text{Cost}},\, \gamma_s)}$$

$$(1-s)dt = \beta_s$$

$$(s)dt = \gamma_s$$

The Quantum Approximate Optimization Algorithm

QAOA: kwaah-waah

> Constraint Satisfaction Problems: ⟶ MAXIMIZE

$z \in \{0,1\}^n$

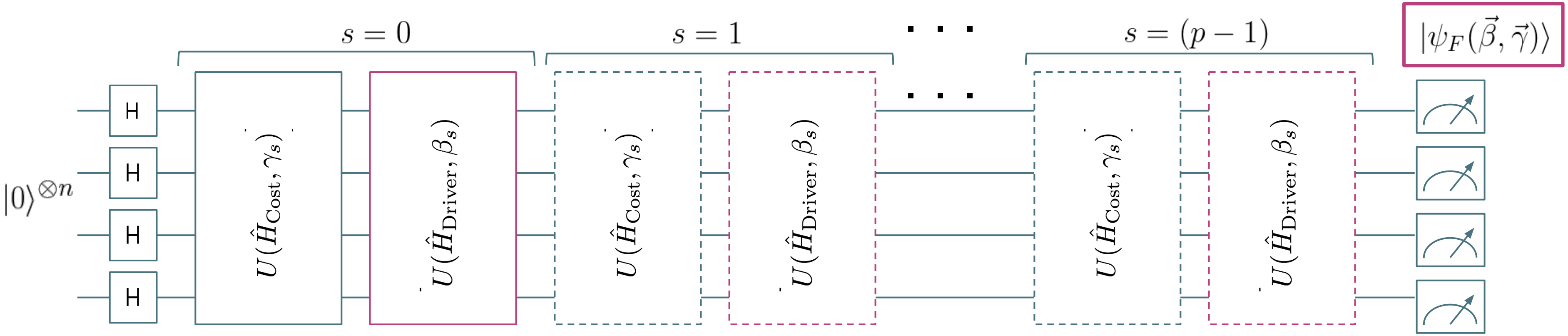
$$C_a(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the constraint } a \\ 0 & \text{if } z \text{ does not .} \end{cases}$$

$$\boxed{C(z) = \sum_{a=1}^{m} C_a(z)}$$

$$\hat{H}_s = (1-s)\hat{H}_{\text{Driver}} + s\hat{H}_{\text{Cost}}$$

$$e^{-i\hat{H}_s dt} = \underbrace{e^{-i(1-s)H_X dt}}_{U(\hat{H}_{\text{Driver}},\beta_s)} \underbrace{e^{-isH_Z dt}}_{U(\hat{H}_{\text{Cost}},\gamma_s)}$$

$$(1-s)dt = \beta_s$$
$$(s)dt = \gamma_s$$



The Quantum Approximate Optimization Algorithm

QAOA: kwaah-waah

> Constraint Satisfaction Problems: → MAXIMIZE

$z \in \{0,1\}^n$

$$C_a(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the constraint } a \\ 0 & \text{if } z \text{ does not .} \end{cases}$$

$$\boxed{C(z) = \sum_{a=1}^{m} C_a(z)}$$

MAXIMIZE

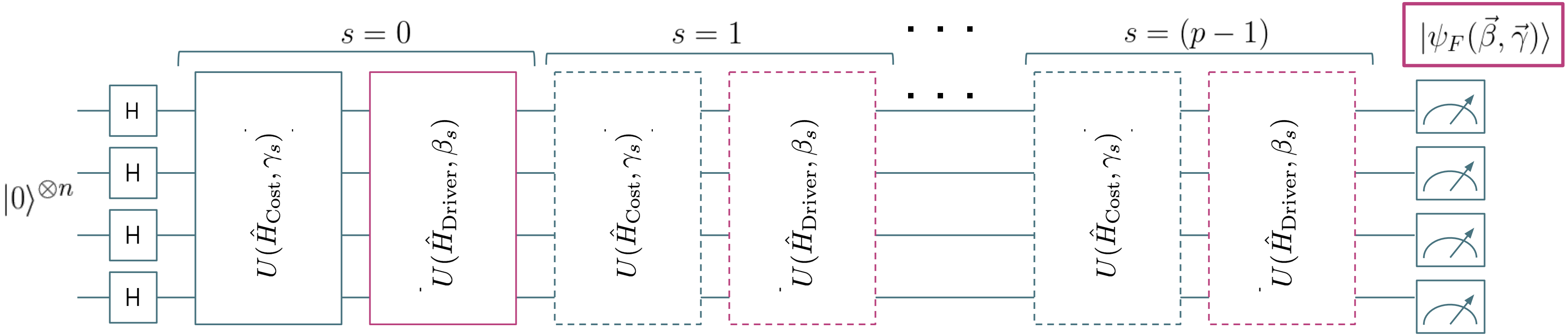
$$\boxed{\langle \psi_F(\vec{\beta}, \vec{\gamma}) | \, C \, | \psi_F(\vec{\beta}, \vec{\gamma}) \rangle}$$

$$\hat{H}_s = (1-s)\hat{H}_{\text{Driver}} + s\hat{H}_{\text{Cost}}$$

$$e^{-i\hat{H}_s dt} = \underbrace{e^{-i(1-s)H_X dt}}_{U(\hat{H}_{\text{Driver}}, \beta_s)} \underbrace{e^{-isH_Z dt}}_{U(\hat{H}_{\text{Cost}}, \gamma_s)}$$

$$(1-s)dt = \beta_s$$

$$(s)dt = \gamma_s$$



The Quantum Approximate Optimization Algorithm

QAOA: kwaah-waah

> Constraint Satisfaction Problems: → MAXIMIZE

$z \in \{0,1\}^n$

$$C_a(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the constraint } a \\ 0 & \text{if } z \text{ does not .} \end{cases}$$

$$\boxed{C(z) = \sum_{a=1}^{m} C_a(z)}$$

$$\hat{H}_s = (1-s)\hat{H}_{\text{Driver}} + s\hat{H}_{\text{Cost}}$$

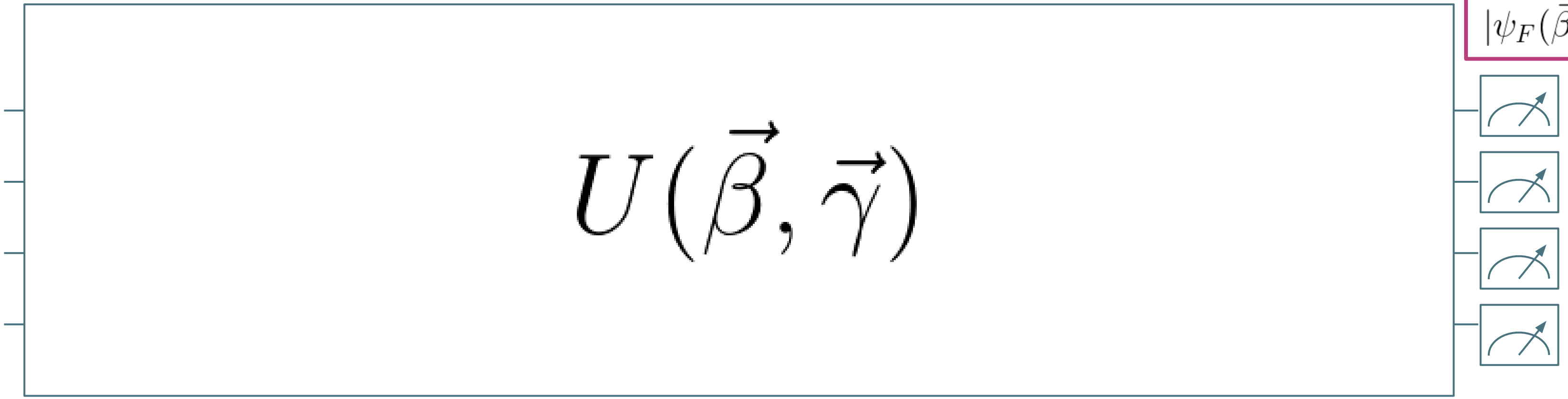
$$e^{-i\hat{H}_s dt} = \underbrace{e^{-i(1-s)H_X dt}}_{U(\hat{H}_{\text{Driver}},\beta_s)}\underbrace{e^{-isH_Z dt}}_{U(\hat{H}_{\text{Cost}},\gamma_s)}$$

$$(1-s)dt = \beta_s$$
$$(s)dt = \gamma_s$$

MAXIMIZE

$$\boxed{\langle\psi_F(\vec{\beta},\vec{\gamma})|\, C\, |\psi_F(\vec{\beta},\vec{\gamma})\rangle}$$

$|0\rangle^{\otimes n}$



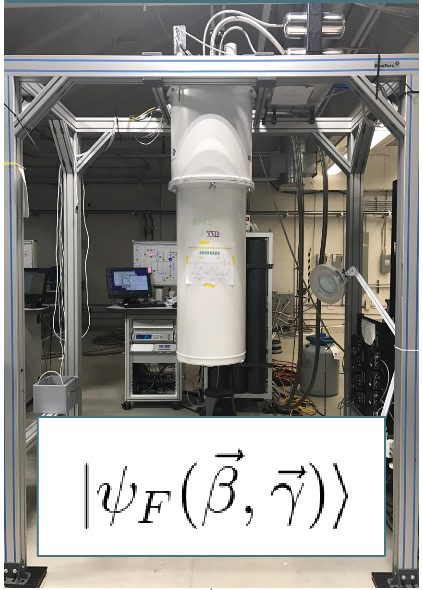
$$\boxed{|\psi_F(\vec{\beta},\vec{\gamma})\rangle}$$

Running QAOA

CPU

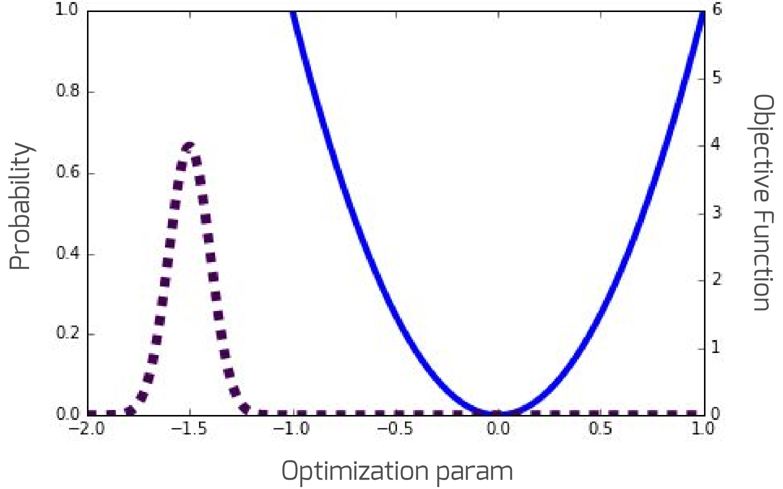
Update Beta, Gamma

Determine direction for minimization



$$|\psi_F(\vec{\beta}, \vec{\gamma})\rangle$$

Noisy Probability Density Minimization

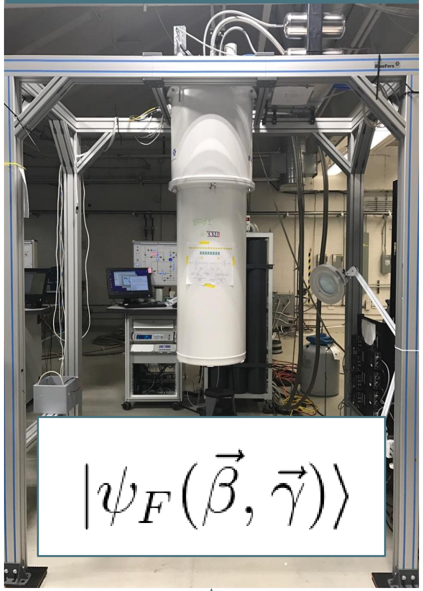


Running QAOA

CPU

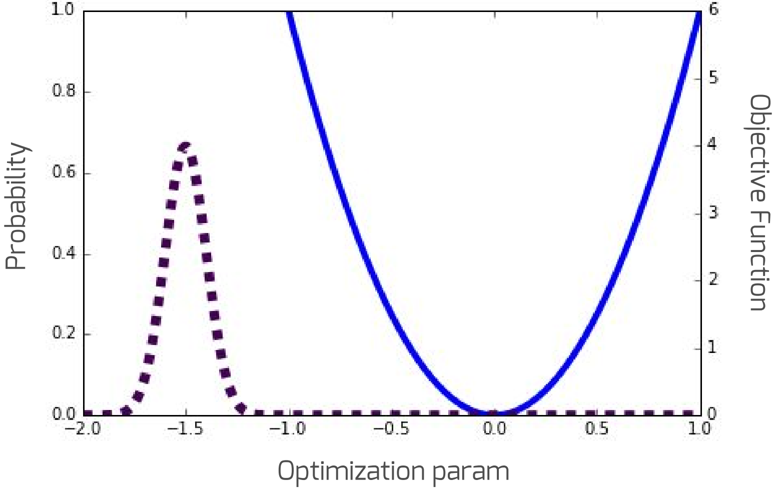
Update Beta, Gamma

Determine direction for minimization



$$|\psi_F(\vec{\beta}, \vec{\gamma})\rangle$$

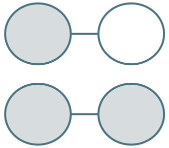
Noisy Probability Density Minimization



Alternative Approach:
analytically calculate optimal coefficients and run once.
WIP by Rieffel & NASA QuAIL

QAOA for MAX-CUT

> Define constraints with an arbitrary graph



Score+1

Score 0

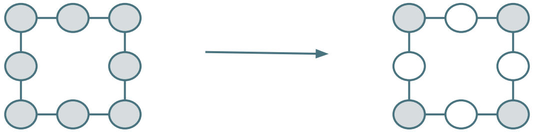
$$C_{ij} = \frac{1}{2}(1 - Z_i Z_j)$$

$$Z_i \in \{+1, -1\}$$

> Hamiltonian Cost Function:

$$\hat{C}_{ij} = \frac{1}{2}\left(\mathbf{I} - \sigma_i^Z \sigma_j^Z\right)$$

> Ring Example:

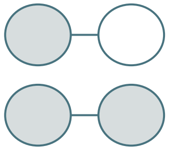


Score = 0

Score = +4

QAOA for MAX-CUT

> Define constraints with an arbitrary graph



Score+1

Score 0

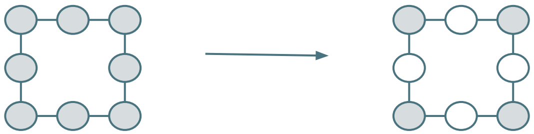
$$C_{ij} = \frac{1}{2}(1 - Z_i Z_j)$$

$$Z_i \in \{+1, -1\}$$

> Hamiltonian Cost Function:

$$\hat{C}_{ij} = \frac{1}{2}\left(\mathbf{I} - \sigma_i^Z \sigma_j^Z\right)$$

> Ring Example:



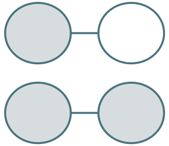
Score = 0

Score = +4

```
import itertools

from pyquil.quil import Program
from pyquil.paulis import sZ, sX, sI, exponential_map
from pyquil.compiler import rpqc

graph = [(0, 1), (1, 2), (2, 3), (3, 4)]
nodes = {node for edge in graph for node in edge}

cost_ham = sum(0.5 * sZ(i) * sZ(j) - 0.5*sI(0) for i, j in graph)
driver_ham = sum(-1. * sX(i) for i in nodes)
```

QAOA for MAX-CUT

> Define constraints with an arbitrary graph



Score+1

Score 0

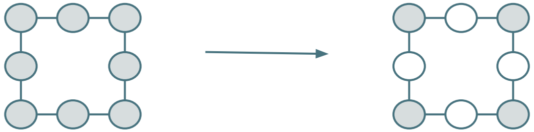
$$C_{ij} = \frac{1}{2}(1 - Z_i Z_j)$$

$$Z_i \in \{+1, -1\}$$

> Hamiltonian Cost Function:

$$\hat{C}_{ij} = \frac{1}{2}\left(\mathbf{I} - \sigma_i^Z \sigma_j^Z\right)$$

> Ring Example:



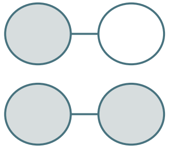
Score = 0

Score = +4

```python
import itertools

from pyquil.quil import Program
from pyquil.paulis import sZ, sX, sI, exponential_map
from pyquil.compiler import rpqc

graph = [(0, 1), (1, 2), (2, 3), (3, 4)]
nodes = {node for edge in graph for node in edge}

cost_ham = sum(0.5 * sZ(i) * sZ(j) - 0.5*sI(0) for i, j in graph)
driver_ham = sum(-1. * sX(i) for i in nodes)

def cost_step(gamma):
   return merge_program([exponential_map(term)(gamma) for term in cost_ham])

def driver_step(beta):
  return merge_program([exponential_map(term)(beta) for term in driver_ham])
```

QAOA for MAX-CUT

> Define constraints with an arbitrary graph



Score+1

Score 0

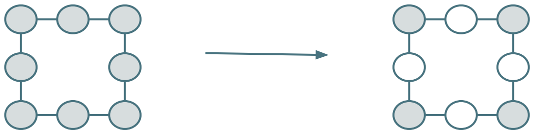
$$C_{ij} = \frac{1}{2}(1 - Z_i Z_j)$$

$$Z_i \in \{+1, -1\}$$

> Hamiltonian Cost Function:

$$\hat{C}_{ij} = \frac{1}{2}\left(\mathbf{I} - \sigma_i^Z \sigma_j^Z\right)$$

> Ring Example:



Score = 0 Score = +4

```
import itertools

from pyquil.quil import Program
from pyquil.paulis import sZ, sX, sI, exponential_map
from pyquil.compiler import rpqc

graph = [(0, 1), (1, 2), (2, 3), (3, 4)]
nodes = {node for edge in graph for node in edge}

cost_ham = sum(0.5 * sZ(i) * sZ(j) - 0.5*sI(0) for i, j in graph)
driver_ham = sum(-1. * sX(i) for i in nodes)

def cost_step(gamma):
   return merge_program([exponential_map(term)(gamma) for term in cost_ham])

def driver_step(beta):
  return merge_program([exponential_map(term)(beta) for term in driver_ham])

def qaoa_circuit_maker(gammas, betas):
    cost_steps = map(cost_step, gammas)
    driver_steps = map(driver_step, betas)

    interleaved_steps = zip(cost_steps, driver_steps)
    qaoa_circuit = merge_program([step for step_pair in interleaved_steps
                                  for step in step_pair])
    return rpqc(qaoa_circuit)

gammas = [2.0, 1.0]
betas = [0.0, 3.0]

qaoa_circuit_maker(gammas, betas)

# then execute the circuit and optimize over gammas and betas
```

Compilation

DEVELOPMENT: pyQuil

QUANTUM INSTRUCTION LANGUAGE: Quil

Allocation & Routing

Unitary Compilation

Scheduling

QUANTUM VIRTUAL MACHINE / QUANTUM COMPUTER

> Quil qubit labels to physical qubits
> Routing to deal with two-qubit gates
> Optimization over noise and errors

> Compilation into the natural gate set
> Rotation Decomposition

> Scheduling into microcode

Why do we need to schedule?

- Quil has **<u>no</u>** notion of time or synchronization.
- But time and synchronization are very important.
- What are our options?

Give up; Admit the physicists are better	**Include *ad hoc* synchronization instructions**	**Compile Quil into some temporal representation**
"Program" with buttons and wires.	Extend Quil to "know" about time.	Add machine-specific directives.

Pros: • Maximal control Cons: • Difficult to reason about • Nixes the idea of an abstraction • Difficult to automate • Have to think about hardware	Pros: • Directly addresses the issue • Still an abstract framework Cons: • Extremely complicated! • Difficult to reason about • Not easily extensible • Hard to implement • Loses the "essence"	Pros: • Remains abstract • Adds control as necessary • Extensible! • Keeps Quil "clean" Cons: • Compilation is more difficult • Performance characterization is machine-specific

Events

`(target, name, start_time, duration, param_dict)`

Schedules

A set of events (and some transformations on them)

```
RZ(1.25) 0
X-HALF 0
CPHASE 0 1
MEASURE 0 [0]
```

Compilation to Schedule →

```
{"events":[
     (0, "+X/2", 0.0, 40.e-9, {"z_shift":1.25}),
     (1, "+F", 50.0.e-9, 250.e-9, {}),
     (0, "READOUT", 310.0e.-9, 1.e-6, {})
]}
```

QPU Microcode is given by supported event types, e.g.

```
(target, "X-HALF", start_time, 40.e-9, {"z_shift":theta})
(target, "+F", start_time, 250.e-9, {})
(target, "-F", start_time, 250.e-9, {})
(target, "READOUT", start_time, 1.e-6, {})
```

Open Problems:

Allocation & Routing	Generic Unitary Decomposition	High performance simulation	Integration with Classical HPC
Optimal implementation includes optimization over: > Gate sets that vary across the chip > Noise in gates > Noise in qubits > Noise in measurements > Crosstalk ScaffCC (1507.01902)	Single-qubit case is well understood O(log(1/e)) (Kliuchnikov et al. 1510.03888) Martinez et al. Compiling quantum algorithms for architectures with multi-qubit gates. 1601.06819 Maslov. Basic circuit compilation techniques for an ion-trap quantum machine. 1603.07678	qHIPSTER. Smelyanski et al. 1601.07195. High Performance Emulation of Quantum Circuits. Haener et al. 1604.06460 0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit. Haener & Steiger. 1704.01127.	Post-processing to reduce impact of sampling error in VQE & QAOA Computationally intensive decoders in QEC Integrations of quantum co-processors in larger workflows, e.g. DMET w/ VQE. Rubin 1610.06910

Acknowledgements

Rigetti Quantum Programming Team

Robert Smith

Nick Rubin

+

Our hardware & control software colleagues, esp.
Spike Curtis, Matt Reagor and Rodney Sinclair

rigetti