Blake Johnson, VP of Quantum Engineering
Rigetti 19-Qubit Quantum Processor

8Q launched June 2017
19Q launched Dec 2017
Founded in 2013 by Chad Rigetti

Fab-1: Fremont, CA

~100 Employees

Forest: Quantum computing over the cloud

Rigetti 19-Qubit Quantum Processor

Venture backed startup

R&D lab: Berkeley, CA

Full-stack platform with superconducting qubits

30-qubit Quantum Virtual Machine™
Full Stack Quantum Computing

Quantum Processor

Hardware

Cloud based Quantum Operating System

Applications
Dedicated quantum processor fab
Fremont, CA
Rigetti’s superconducting quantum processors
Access to **actual quantum computers**, in addition to a simulated environment.
Superconducting qubit performance has increased by $>10^6$ in the last 15 years.

Advances in quantum-classical hybrid algorithms have unlocked near-term applications.

"Schoelkopf's Law" by O'Malley et al. 1512.06860

E.g., Variational Quantum Eigensolver
Quantum advantage will be achieved in the next 2-5 years.

Noisy Intermediate Scale Quantum computing

NISQ application areas:
- Quantum chemistry
- Optimization
- Machine learning

Fault-tolerant QC

Error correction threshold

physical error rate

number of physical qubits

"Quantum computing in the NISQ era and beyond" Preskill, 2018 https://arxiv.org/abs/1801.00862
Quantum advantage will be achieved in the next 2-5 years

Noisy Intermediate Scale Quantum computing

NISQ application areas:
- Quantum chemistry
- Optimization
- Machine learning

Error correction threshold

physical error rate

number of physical qubits

"Quantum computing in the NISQ era and beyond" Preskill, 2018 https://arxiv.org/abs/1801.00862
Quantum advantage will be achieved in the next 2-5 years.

Noisy Intermediate Scale Quantum computing (NISQ) application areas:
- Quantum chemistry
- Optimization
- Machine learning

"Quantum computing in the NISQ era and beyond" Preskill, 2018
https://arxiv.org/abs/1801.00862

Where we are today within 5 years.

Error correction threshold.

Physical error rate vs. number of physical qubits graph.
### Some Potential Quantum Applications

**Potential advantages across different use-cases**

<table>
<thead>
<tr>
<th><strong>Machine Learning</strong></th>
<th><strong>Supply Chain Optimization</strong></th>
<th><strong>Robotic Manufacturing</strong></th>
<th><strong>Computational Materials Science</strong></th>
<th><strong>Alternative Energy Research</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; Development of new training sets and algorithms</td>
<td>&gt; Forecast and optimize for future inventory demand</td>
<td>&gt; Reduce manufacturing time and cost</td>
<td>&gt; Design of better catalysts for batteries</td>
<td>&gt; Efficiently convert atmospheric CO$_2$ to methanol</td>
</tr>
<tr>
<td>&gt; Classification and sampling of large data sets</td>
<td>&gt; NP-hard scheduling and logistics map into quantum applications</td>
<td>&gt; Maps to a Traveling Salesman Problem addressable by quantum constrained optimization</td>
<td>&gt; Quantum algorithms for calculating electronic structure</td>
<td>&gt; Powered by existing hybrid quantum-classical algorithms + machine learning</td>
</tr>
</tbody>
</table>

![Machine Learning Image](image1.png)  
![Supply Chain Optimization Image](image2.png)  
![Robotic Manufacturing Image](image3.png)  
![Computational Materials Science Image](image4.png)  
![Alternative Energy Research Image](image5.png)
Forest

The only product that is architected and engineered across the stack for Hybrid Quantum/Classical Computing
Outline

■ A brief look at Rigetti 19Q
■ Forest programming API
■ Solving a clustering problem on 19Q
Rigetti 19Q

Aluminum circuit on Silicon

Device Properties

- 4x5 lattice of transmon qubits and resonators
- Fixed capacitive coupling between qubits
- Alternating arrangement of fixed-frequency and tunable qubits
- “19Q” because one tunable qubit was out of spec

Circuit QED:
Rigetti 19Q

Device Properties

- 4x5 lattice of transmon qubits and resonators
- Fixed capacitive coupling between qubits
- Alternating arrangement of fixed-frequency and tunable qubits
- $T_1 = 8-30 \mu s$, $T_{2^*} = 5-25 \mu s$

Circuit QED:
Qubit-qubit interactions

- 2-qubit *parametric gates* use RF flux modulation to turn on effective resonance conditions
- Typical 2-qubit error rate of 5 - 15%
- Usable circuit depth of 8-10

**Rigetti 19Q**

19Q connectivity graph

**Parametric gates:**

**Inspired by:**
- **Flux qubit intermediary:** Niskanen et al, *Science* 316, p. 723 (2007)
- **B-tune gate:** McKay et al *Phys Rev Applied* 6, 064007 (2016)
- **FM gate theory:** Beaudoin et al *PRA* 86, 022305 (2012)
- **FM gate experiment:** Strand et al, *PRB* 87, 220505(R) (2013)
How do you use quantum computers today?
API endpoint

We have 2 API endpoints:

Cloud Compute Backends

19Q

Quantum Virtual Machine
We have 2 API endpoints:

Cloud Compute Backends

- 19Q
- Quantum Virtual Machine

**Simulator. Ideal for testing and development.**
We have 2 API endpoints:

- **Quantum Virtual Machine**
- **19Q Cloud Compute Backends**
Sign up in one click

rigetti.com/forest
Setup

Setup & Install

1. pip install pyquil
   a. Run `setup-pyquil-config`

2. pip install quantum-grove
from pyquil.quil import Program
from pyquil.gates import H, CNOT
from pyquil.api import QPUConnection

program = Program(H(1), H(5), CNOT(1, 5), H(5))

qpu = QPUConnection("19Q-Acorn")
results = qpu.run(program, classical_addresses=[0, 1], trials=10)

Generate entanglement in minutes.
Quantum Physics vs. Quantum Computing
Quantum Physics vs. Quantum Computing

Crazy physics
Quantum Physics vs. Quantum Computing

Crazy physics

Forest

Store to classical register

Averaging & filters

Single-shot readout

Qubit operations
Quantum Physics vs. Quantum Computing

Crazy physics

Forest

15 years of research
Quantum Physics vs. Quantum Computing

Abstraction layer

Crazy physics

15 years of research
Quantum Physics vs. Quantum Computing

What you write

```
program = Program(H(θ), CNOT(θ, 1))
results = qpu.run(program, classical_addresses=[0, 1], trials=10)
```

Abstraction layer

Crazy physics

Forest

15 years of research
Grove

Open source applications library
Grove

Open source applications library

Quantum Approximate Optimization Algorithm (QAOA)
Quantum Approximate Optimization Algorithm (QAOA)

Applications in graph theory and quantum machine learning

\[ \hat{C}_{ij} = \frac{1}{2} (I - \sigma_i^Z \sigma_j^Z) \]

Score +1
Score 0

Score = +8 (max)

**Open source applications library**

**Quantum Approximate Optimization Algorithm (QAOA)**

Applications in graph theory and quantum machine learning.

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

- Score +1
- Score 0

**Variational Quantum Eigensolver (VQE)**

Score = +8 (max)

Grove

Open source applications library

Quantum Approximate Optimization Algorithm (QAOA)

Applications in graph theory and quantum machine learning.

\[ \hat{C}_{ij} = \frac{1}{2} (I - \sigma_i^Z \sigma_j^Z) \]

Score+1

Score 0

Score = +8 (max)

Variational Quantum Eigensolver (VQE)

Applications in quantum simulation


Dumitrescu et al. arxiv: 1801.03897 (2018)
Grove

Open source applications library

Quantum Approximate Optimization Algorithm (QAOA)

Applications in graph theory and quantum machine learning.

\[ \hat{C}_{ij} = \frac{1}{2} (I - \sigma_i^Z \sigma_j^Z) \]

Score+1

Score 0

Score = +8 (max)

from grove.pyqaoa import qaoa


Variational Quantum Eigensolver (VQE)

Applications in quantum simulation

Quantum Approximate Optimization Algorithm (QAOA)

Applications in graph theory and quantum machine learning.

\[ \hat{C}_{ij} = \frac{1}{2} (I - \sigma_i^Z \sigma_j^Z) \]

Score+1

Score 0

Score = +8 (max)

Variational Quantum Eigensolver (VQE)

Applications in quantum simulation

from grove.pyqaoa import qaoa

from grove.pyvqe import vqe


Dumitrescu et. al. arxiv: 1801.03897 (2018)
Clustering

Given an unlabeled set of points
Clustering

Given an unlabeled set of points, find labels based upon similarity metric (e.g. Euclidean distance).
Clustering as MAXCUT

Construct a graph $G=(V,E)$ where the edge weights $w_{i,j}$ are determined by the distance metric. Then, MAXCUT is a clustering algorithm for the original points.

$$\text{MAXCUT} = \max_{\text{cut } S \subseteq E} \sum_{(i,j) \in S} w_{ij}$$
Construct a graph $G=(V,E)$ where the edge weights $w_{i,j}$ are determined by the distance metric. Then, MAXCUT is a clustering algorithm for the original points.

Clustering transformed into an **optimization** problem solvable with QAOA.
Sparse graphs with 19Q connectivity

Generate a family of sparse graphs with random weights matching the connectivity of 19Q.
Sparse graphs with 19Q connectivity

This family of graphs...

...allows implementation of $H_C$ in a circuit of depth 3 (becomes depth 6 after compilation of $e^{-i\gamma \omega_{ij} Z_i Z_j}$)
Clustering on 19Q

83 trials for a fixed problem instance

In many such trials, the algorithm actually finds the optimal solution.

From these trials we calculate an empirical CDF.
Clustering performance

Success probability monotonically increases with number of steps.
Clustering performance

Success probability monotonically increases with number of steps.

Noise in 19Q has a significant impact on performance.
Advancing Quantum Algorithms and Applications to Reality

Rigetti scientists and engineers:

- An unsupervised machine learning problem using clustering. This is the largest demonstration ever of a hybrid algorithm on a gate-model processor (arXiv:1712.05771)

- The OpenFermion package for molecular and materials simulations with smaller quantum computers, in collaboration with Google and others (arXiv:1710.07629)

```python
from pyquil.quil import Program
from pyquil.gates import X
from pyquil.paulis import exponentiate
localized_electrons_program = Program()
localized_electrons_program.inst([(X(0), X(1))])
pyquil_program = Program()
for term in pyquil.hubbard_generator.terms:
    pyquil_program += exponentiate(0.1 * term)
print(localized_electrons_program + pyquil_program)
```

Forest users working with our applications team:

- Quantum-startup Everettian developed an algorithm to train Boltzmann machine neural network using a hybrid approach on Forest (arXiv:1712.05304)

- Scientists at Oak Ridge National Lab used Forest to simulate deuteron binding energy, the first step toward scalable nuclear structure computation. (arXiv:1801.03897)

- Scientists at Los Alamos National Lab used Forest to demonstrate a machine learning approach to reduce the error rate of an important near-term quantum algorithm.
Apply for QPU access today.

Request Access to QPU

Thanks for your interest in upgrading to developer access. Please fill out the following information and we’ll get back to you with details on the access process.

Full Name

Email address
Organization name

Reason for QPU Access. Detailed descriptions that indicate results have already been tested on the QVM will be prioritized.

REQUEST ACCESS

rigetti.com/qpu-request
Thank you

www.rigetti.com
Appendix
Hybrid quantum computing

**CPU**
- Create program with selected parameters
- Select angles
- Evaluate termination criterion
- Analyze Samples & Evaluate real valued objective function

**QPU**
- Compile to hardware instructions
- Execute instructions
- Measure & Record Sample

Entry

Output
Programming QAOA thru Forest

QAOA
In 15 lines of code

from pyquil.quil import Program
from pyquil.gates import H
from pyquil.paulis import sI, sX, sZ, exponentiate_commuting_pauli_sum
from pyquil.api import QPUConnection

graph = [((0, 1), (1, 2), (2, 3))]
weights = [0.5, 1.0, 0.5]
nodes = range(4)

init_state_prog = sum([H(i) for i in nodes], Program())
h_cost = -0.5 * sum(w * sI(nodes[i]) - sZ(i) * sZ(j) for w, (i, j) in zip(weights, graph))
h_driver = -1. * sum(sX(i) for i in nodes)

def qaoa_ansatz(betas, gammas):
    return sum([exponentiate_commuting_pauli_sum(h_cost)(g) + exponentiate_commuting_pauli_sum(h_driver)(b) 
                for g, b in zip(gammas, betas)], Program())

program = init_state_prog + qaoa_ansatz([0., 0.5], [0.75, 1.])

qpu = QPUConnection(‘19Q-Acorn’)
qpu.run_and_measure(program, qubits=nodes, trials=10)
from pyquil.quil import Program
from pyquil.gates import H
from pyquil.paulis import sI, sX, sZ, exponentiate_commuting_pauli_sum
from pyquil.api import QPUConnection

graph = [(0, 1), (1, 2), (2, 3)]
weights = [0.5, 1.0, 0.5]
nodes = range(4)

init_state_prog = sum([H(i) for i in nodes], Program())
h_cost = -0.5 * sum(w * sI(nodes[i]) - sZ(i) * sZ(j) for w, (i, j) in zip(weights, graph))
h_driver = -1.0 * sum(sX(i) for i in nodes)

def qaoa_ansatz(betas, gammas):
    return sum([exponentiate_commuting_pauli_sum(h_cost)(g) + exponentiate_commuting_pauli_sum(h_driver)(b) 
    for g, b in zip(gammas, betas)], Program())

program = init_state_prog + qaoa_ansatz([0.0, 0.5], [0.75, 1.0])

qpu = QPUConnection('19Q-Acorn')
qpu.run_and_measure(program, qubits=nodes, trials=10)

replace with QVMConnection() to run on simulator

Compiler handles transformation to native gate set
Clustering performance

Success probability monotonically increases with number of steps.

Noise in 19Q has a significant impact on performance.

Approach clearly outperforms random sampling.
Welcome to Rigetti Forest!

We're glad to have you. Here are the connection details for your beta access to Forest:

[Rigetti Forest]
url: https://api.rigetti.com/qvm
key: nmRPfkln5odDeldGsl5odsjreovFoofrkuher376u
user_id: 43dae850-4b10-4b4d-8d0b-68b4a86b5046

Place this information in a file called .pyquil_config in your home directory and you'll be set up to connect.

—The Rigetti Software & Applications Team

Hold on to this for use during the next step.
Quantum advantage will be achieved soon

- In the next ~2-5 years, hybrid quantum/classical computers will demonstrate quantum advantage
- Forest is optimized for this Noisy Intermediate Scale Quantum era
- Longer-term, fault tolerant quantum computers will emerge and unlock computational capabilities such as exact protein folding

"Quantum computing in the NISQ era and beyond" Preskill, 2018 https://arxiv.org/abs/1801.00862