A Platform for Quantum Programming

Will Zeng
Rigetti Computing

Stanford Platform Lab Seminar
16.30-17.30 in Gates 415
April 3, 2018
Q1. Why program a quantum computer?

Q2. How do I program a quantum computer?
Why program a quantum computer?

New power | New opportunity | Fundamental curiosity
Why program a quantum computer?

**New power** | **New opportunity** | **Fundamental curiosity**

Quantum computing power* scales exponentially with qubits.

*N bits* can exactly simulate *log N qubits*

This compute unit can exactly simulate:

- **Commodore 64**: 10 Qubits
- **AWS M4 Instance**: 30 Qubits
- **Entire Global Cloud**: 60 Qubits

* We will be more precise later in the talk
Why program a quantum computer?

Quantum computing power* scales exponentially with qubits

N bits can exactly simulate $\log N$ qubits

This compute unit....

- **Commodore 64**
  - 10 Qubits

- **AWS M4 Instance**
  - 30 Qubits
  - 1 Million x Commodore 64

- **Entire Global Cloud**
  - 60 Qubits
  - 1 Billion x (1 Million x Commodore 64)

**Rigetti 19 qubits**

available since Dec 2017

* We will be more precise later in the talk
### Why program a quantum computer?

**New power** | **New opportunity** | **Fundamental curiosity**

<table>
<thead>
<tr>
<th><strong>Alternative Energy Research</strong></th>
<th><strong>Robotic Manufacturing</strong></th>
<th><strong>Supply Chain Optimization</strong></th>
<th><strong>Computational Materials Science</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Efficiently convert atmospheric CO₂ to methanol</td>
<td>- Reduce manufacturing time and cost</td>
<td>- Forecast and optimize for future inventory demand</td>
<td>- Design of better catalysts for batteries</td>
</tr>
<tr>
<td>- Powered by existing hybrid quantum-classical algorithms + machine learning</td>
<td>- Maps to a Traveling Salesman Problem addressable by quantum constrained optimization</td>
<td>- NP-hard scheduling and logistics map into quantum applications</td>
<td>- Quantum algorithms for calculating electronic structure</td>
</tr>
</tbody>
</table>

![Carbon Capture & Recycling](image1)

![Robotic Manufacturing](image2)

![Supply Chain Optimization](image3)

![Computational Materials Science](image4)
Why program a quantum computer?

New power | **New opportunity** | Fundamental curiosity

Quantum processors are scaling up quickly

![Graph showing quantum processors scaling up](image-url)
Why program a quantum computer?

New power | **New opportunity** | Fundamental curiosity

Investments across academia, government, and industry are global and growing

---

**No small effort**
Estimated annual spending on non-classified quantum-technology research, 2015, €m

Source: McKinsey

*Combined estimated budget of EU countries
Why program a quantum computer?

New power | New opportunity | **Fundamental curiosity**
Why program a quantum computer?

New power | New opportunity | Fundamental curiosity
Why program a quantum computer?

New power | New opportunity | **Fundamental curiosity**

Quantum computing reorients the relationship between physics and computer science.

*Every “function which would *naturally* be regarded as computable” can be computed by the universal Turing machine.* - Turing

“...*nature isn't classical, dammit...*” - Feynman
Why program a quantum computer?

New power | New opportunity | **Fundamental curiosity**

Quantum computing reorients the relationship between physics and computer science.

*Every “function which would naturally be regarded as computable” can be computed by the universal Turing machine.* - Turing

“... *nature isn't classical, dammit...*” - Feynman

Physical phenomenon apply to information and computation as well.

> Superposition  > No-cloning  > Teleportation
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

Quantum computers have quantum processor(s) and classical processors.
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

Quantum computers have quantum processor(s) and classical processors.

Chip goes here

Classical control racks

Quantum processor

Full quantum computing system

Otterbach et al. arXiv:1712.05771
How do I program a quantum computer?

Practical, valuable quantum computing is **Hybrid** Quantum/Classical Computing.

**Hybrid Quantum Computers** | Quantum Programming | Hybrid Programming | Hybrid Algorithms

---

How do I program a quantum computer?

Practical, valuable quantum computing is **Hybrid** Quantum/Classical Computing

**Forest** is optimized for this.

---

How do I program a quantum computer?

Practical, valuable quantum computing is **Hybrid** Quantum/Classical Computing

**Forest** is optimized for this with our Quil [01] instruction set.

### Bits vs. Probabilistic Bits vs. Qubits

<table>
<thead>
<tr>
<th>State (single unit)</th>
<th>Bit $b \in {0, 1}$</th>
<th>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</th>
<th>$a + b \in \mathbb{R}^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
<td>$a + b = 1$</td>
</tr>
</tbody>
</table>

**How do I program a quantum computer?**

**Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms**
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (single unit)</td>
<td>Bit $b \in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Probability of 0

Probability of 1
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (single unit)</td>
<td>Bit $b \in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b = 1$</td>
</tr>
</tbody>
</table>
How do I program a quantum computer?

Hybrid Quantum Computers | **Quantum Programming** | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>State (single unit)</th>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>$b \in {0, 1}$</td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
<td>$\alpha, \beta \in \mathbb{C}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b \in \mathbb{R}^+$</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b = 1$</td>
<td></td>
</tr>
</tbody>
</table>

**CLASSICAL BIT**

```
    0
      |
      v
    1
```
### How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (single unit)</td>
<td>Bit $b \in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td>$a + b \in \mathbb{R}_+$</td>
<td>$a + b = 1$</td>
<td>$\alpha, \beta \in \mathbb{C}$</td>
</tr>
<tr>
<td>$</td>
<td>\alpha</td>
<td>^2 +</td>
</tr>
</tbody>
</table>

**Diagram:**

- Classical Bit
- $|\alpha|^2 = \text{Probability of 0}$
- $|\beta|^2 = \text{Probability of 1}$
How do I program a quantum computer?

Hybrid Quantum Computers | **Quantum Programming** | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>State (single unit)</th>
<th>Bit</th>
<th>Real vector</th>
<th>Probabilistic Bits</th>
<th>Complex vector</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$b \in {0, 1}$</td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
<td>$a + b \in \mathbb{R}^+$</td>
<td>$\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$</td>
<td>$\alpha, \beta \in \mathbb{C}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b = 1$</td>
<td></td>
<td>$</td>
<td>\alpha</td>
</tr>
</tbody>
</table>

$$\text{coin} = \frac{1}{2}\vec{0} + \frac{1}{2}\vec{1}$$
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>State (single unit)</th>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bit</td>
<td>Real vector</td>
<td>Complex vector</td>
</tr>
<tr>
<td></td>
<td>$b \in {0, 1}$</td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
<td>$\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$</td>
</tr>
<tr>
<td></td>
<td>$a + b \in \mathbb{R}_+$</td>
<td>$a + b = 1$</td>
<td>$</td>
</tr>
</tbody>
</table>

\[
\text{coin} = \frac{1}{2} \vec{0} + \frac{1}{2} \vec{1}
\]

\[
\text{qcoin}(\theta) = \frac{1}{\sqrt{2}} \vec{0} + \frac{e^{i\theta}}{\sqrt{2}} \vec{1}
\]
# How do I program a quantum computer?

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (single unit)</td>
<td>Bit ( b \in {0, 1} )</td>
<td>Real vector ( \vec{b} = a\vec{0} + b\vec{1} )</td>
</tr>
<tr>
<td>( a + b \in \mathbb{R}^+ )</td>
<td>( a + b = 1 )</td>
<td>(</td>
</tr>
</tbody>
</table>

- \( |\alpha|^2 \) = Probability of 0
- \( |\beta|^2 \) = Probability of 1

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms
# How do I program a quantum computer?

Hybrid Quantum Computers | **Quantum Programming** | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (single unit)</td>
<td>Bit $b \in {0, 1}$</td>
<td>Real vector $\vec{b} = a \vec{0} + b \vec{1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b \in \mathbb{R}_+$</td>
</tr>
<tr>
<td>State (multi-unit)</td>
<td>Bitstring $x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector) $\vec{p} = p_{0,0}, \ldots, p_{x,x}, \ldots, p_{1,1}$</td>
</tr>
</tbody>
</table>

\[
\vec{p} = \bigotimes_{i=1}^{n} b_i \\
\vec{\psi} = \bigotimes_{i=1}^{n} \psi_i
\]
### How do I program a quantum computer?

Hybrid Quantum Computers | **Quantum Programming** | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State (single unit)</strong></td>
<td>Bit ( b \in {0, 1} )</td>
<td>Real vector ( \vec{b} = a\vec{0} + b\vec{1} ) ( a + b \in \mathbb{R}_+ ) ( a + b = 1 )</td>
</tr>
<tr>
<td><strong>State (multi-unit)</strong></td>
<td>Bitstring ( x \in {0, 1}^n )</td>
<td>Prob. Distribution (stochastic vector) ( \vec{p} = p_0...0, \ldots, p_x, \ldots, p_{1...1} )</td>
</tr>
</tbody>
</table>

\[
\vec{p} = \bigotimes_{i} b_i \quad \quad \vec{\psi} = \bigotimes_{i} \psi_i
\]

\( |\alpha_x|^2 = \text{Probability of bitstring } x \)
### How do I program a quantum computer?

**Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State (single unit)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit</td>
<td>$b \in {0, 1}$</td>
<td>Real vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td><strong>State (multi-unit)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitstring</td>
<td>$x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector)</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>Boolean Logic</td>
<td>Stochastic Matrices</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram:

```
\begin{tikzpicture}
  \node (P1) at (0,0) {$P_1$};
  \node (P2) at (2,0) {$P_2$};
  \node (p_t0) at (0,-1) {$\vec{p}_{t0}$};
  \node (p_t) at (1,-1) {$\vec{p}_t$};
  \node (p_ij) at (2,-1) {$\vec{p}_{ij}$};

  \draw[->] (P1) edge (p_t0);
  \draw[->] (P1) edge (p_t);
  \draw[->] (P2) edge (p_t);
  \draw[->] (P2) edge (p_ij);
\end{tikzpicture}
```
**How do I program a quantum computer?**

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>State (single unit)</th>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>$b \in {0, 1}$</td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$, $a + b = 1$</td>
<td>Complex vector $\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$, $</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State (multi-unit)</th>
<th>Bitstring</th>
<th>Prob. Distribution (stochastic vector)</th>
<th>Wavefunction (complex vector)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \in {0, 1}^n$</td>
<td>$\vec{p} = p_{0\ldots0}, \ldots, p_x, \ldots, p_{1\ldots1}$</td>
<td>$\vec{\psi} = \alpha_{0\ldots0}, \ldots, \alpha_x, \ldots, \alpha_{1\ldots1}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations</th>
<th>Boolean Logic</th>
<th>Stochastic Matrices</th>
<th>Unitary Matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{j=1}^{S} P_{i,j} = 1.$</td>
<td>$U^\dagger U = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How do I program a quantum computer?

Hybrid Quantum Computers | **Quantum Programming** | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State (single unit)</strong></td>
<td><strong>State (multi-unit)</strong></td>
<td><strong>Operations</strong></td>
</tr>
<tr>
<td>Bit</td>
<td>Bitstring</td>
<td>Boolean Logic</td>
</tr>
<tr>
<td>$b \in {0,1}$</td>
<td>$x \in {0,1}^n$</td>
<td>Stochastic Matrices</td>
</tr>
<tr>
<td>Real vector $\vec{b} = a\hat{0} + b\hat{1}$, $a + b = 1$</td>
<td>Prob. Distribution (stochastic vector) $\vec{p} = p_0, \ldots, p_x, \ldots, p_{1\ldots1}$</td>
<td>Unitary Matrices $U^\dagger U = 1$</td>
</tr>
<tr>
<td>Complex vector $\vec{\psi} = \alpha\hat{0} + \beta\hat{1}$, $</td>
<td>\alpha</td>
<td>^2 +</td>
</tr>
</tbody>
</table>

**Sampling**
### How do I program a quantum computer?

**Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms**

<table>
<thead>
<tr>
<th></th>
<th><strong>Bits</strong></th>
<th><strong>Probabilistic Bits</strong></th>
<th><strong>Qubits</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State (single unit)</strong></td>
<td>Bit</td>
<td>Real vector</td>
<td>Complex vector</td>
</tr>
<tr>
<td></td>
<td>$b \in {0, 1}$</td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
<td>$\psi = \alpha\vec{0} + \beta\vec{1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b \in \mathbb{R}^+$</td>
<td>$\alpha, \beta \in \mathbb{C}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b = 1$</td>
<td>$</td>
</tr>
<tr>
<td><strong>State (multi-unit)</strong></td>
<td>Bitstring</td>
<td>Prob. Distribution (stochastic vector)</td>
<td>Wavefunction (complex vector)</td>
</tr>
<tr>
<td></td>
<td>$x \in {0, 1}^n$</td>
<td>$\vec{p} = p_0...0, \ldots, p_x, \ldots, p_1...1$</td>
<td>$\psi = \alpha_0...0, \ldots, \alpha_x, \ldots, \alpha_1...1$</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>Boolean Logic</td>
<td>Stochastic Matrices</td>
<td>Unitary Matrices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sum_{j=1}^{S} P_{i,j} = 1.$</td>
<td>$U^\dagger U = 1$</td>
</tr>
<tr>
<td><strong>Component Ops</strong></td>
<td>Boolean Gates</td>
<td>Tensor products of matrices</td>
<td>Tensor products of matrices</td>
</tr>
</tbody>
</table>

**Sampling**

$|\alpha_x|^2 = \text{Probability of bitstring } x$

**Born rule**
# How do I program a quantum computer?

## Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

<table>
<thead>
<tr>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (single unit)</td>
<td>Bit ( b \in {0, 1} )</td>
<td>Real vector ( \vec{b} = a\vec{0} + b\vec{1} ) ( a + b \in \mathbb{R}_+ ) ( a + b = 1 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complex vector ( \vec{\psi} = \alpha\vec{0} + \beta\vec{1} ) ( \alpha, \beta \in \mathbb{C} ) (</td>
</tr>
<tr>
<td>State (multi-unit)</td>
<td>Bitstring ( x \in {0, 1}^n )</td>
<td>Prob. Distribution (stochastic vector) ( \vec{p} = p_{00...0}, ..., p_x, ..., p_{1...1} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wavefunction (complex vector) ( \vec{\psi} = \alpha_{00...0}, ..., \alpha_x, ..., \alpha_{1...1} )</td>
</tr>
<tr>
<td>Operations</td>
<td>Boolean Logic</td>
<td>Stochastic Matrices ( \sum_{j=1}^{S} P_{i,j} = 1. )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unitary Matrices ( U^\dagger U = 1 )</td>
</tr>
<tr>
<td>Component Ops</td>
<td>Boolean Gates</td>
<td>Tensor products of matrices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tensor products of matrices</td>
</tr>
</tbody>
</table>

## Sampling

- **Born rule**
- **Measurement**
Quil (Quantum Instruction Language) gives each quantum operation an instruction

\[
\text{<instruction> <qubit targets>}
\]

Start in 0

\[
\psi = [1, 0, 0, 0]
\]

X 0 # “quantum NOT”
Quil (Quantum Instruction Language) gives each quantum operation an instruction

<instruction> <qubit targets>

X 0 # “quantum NOT”

\[ \psi = \begin{bmatrix} 1, 0, 0, 0 \\ 0, 1, 0, 0 \end{bmatrix} \]

\[ X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \]

Apply X instr to 0th qubit

\[ \psi = \begin{bmatrix} 0, 1, 0, 0 \\ 0, 0, 1, 0 \end{bmatrix} \]
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

Quil (Quantum Instruction Language) gives each quantum operation an instruction

\[ <\text{instruction}> \ <\text{qubit targets}> \]

\[
\begin{align*}
\psi &= [1, 0, 0, 0] \\
&= [0, 1, 0, 0] \\
&= [0, 0, 1, 0] \\
&= [0, 0, 0, 1] \\
\end{align*}
\]

\[
X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}
\]

\[
H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}
\]

X 0 # “quantum NOT”

H 0 # Hadamard gate
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

Quil (Quantum Instruction Language) gives each quantum operation an instruction

\[ \text{<instruction> <qubit targets>} \]

\[ X 0 \# "quantum NOT" \]

\[ X 0 \]

\[ H 0 \# Hadamard gate \]

\[ \text{Apply H instr to 0th qubit} \]

\[ \psi = [1, 0, 0, 0] \]

\[ \psi = [0, 1, 0, 0] \]

\[ \psi = [0, 1, 0, 0] \]

\[ \psi = [1/\sqrt{2}, 1/\sqrt{2}, 0, 0] \]

\[ X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \]

\[ H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \]
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

Quil (Quantum Instruction Language) gives each quantum operation an instruction

\[ \langle \text{instruction} \rangle \ \langle \text{qubit targets} \rangle \]

\( X \ 0 \ # \ \text{“quantum NOT”} \)

\( X \ 0 \)

\( H \ 0 \ # \ \text{Hadamard gate} \)

\( \text{CNOT} \ 0 \ 1 \)

\[\psi = \begin{bmatrix} 1, & 0, & 0, & 0 \\ 0, & 1, & 0, & 0 \end{bmatrix}\]

\[\psi = \begin{bmatrix} 0, & 1, & 0, & 0 \end{bmatrix}\]

\[\psi = \begin{bmatrix} 1/\sqrt{2}, & 1/\sqrt{2}, & 0, & 0 \end{bmatrix}\]

\[X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\]

\[H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\]

\[\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}\]
How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

Quil (Quantum Instruction Language) gives each quantum operation an instruction

<instruction> <qubit targets>

\[ \psi = \begin{bmatrix} 1, 0, 0, 0 \\ 0, 1, 0, 0 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 0, 1, 0, 0 \\ 0, 1, 0, 0 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 1/\sqrt{2}, 1/\sqrt{2}, 0, 0 \\ 0, 0, 0, 1 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 1/\sqrt{2}, 0, 0, 1/\sqrt{2} \\ 0, 0, 0, 1 \end{bmatrix} \]

\[ X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \]

\[ H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \]

\[ \text{CNOT} = cX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \]

X 0 # “quantum NOT”

X 0

H 0 # Hadamard gate

CNOT 0 1

Apply CNOT instr to 0 and 1 qubits
How do I program a quantum computer?

Quil (Quantum Instruction Language) gives each quantum operation an instruction

\( \langle \text{instruction} \rangle \ \langle \text{qubit targets} \rangle \)

\( X \ 0 \ # \ \text{“quantum NOT”} \)

\( H \ 0 \ # \ \text{Hadamard gate} \)

\( \text{CNOT} \ 0 \ 1 \)

\[ \psi = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 0, & 1, & 0, & 0 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 1/\sqrt{2}, & 1/\sqrt{2}, & 0, & 0 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 1/\sqrt{2}, & 0, & 0, & 1/\sqrt{2} \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 0, & 1, & 0, & 0 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 0, & 0, & 1, & 1 \end{bmatrix} \]

\[ \psi = \begin{bmatrix} 1/\sqrt{2}, & 0, & 0, & 1/\sqrt{2} \end{bmatrix} \]

Qubits 0 and 1 are ENTANGLED
Quil (Quantum Instruction Language) gives each quantum operation an instruction

<instruction> <qubit targets>

X 0 # “quantum NOT”
X 0
H 0 # Hadamard gate
CNOT 0 1

\[ \psi = \left[ \frac{1}{\sqrt{2}}, 0, 0, \frac{1}{\sqrt{2}} \right] \]
How do I program a quantum computer?

Quil (Quantum Instruction Language) gives each quantum operation an instruction

\(<\text{instruction}> \ <\text{qubit targets}>\)

\(X \ 0 \ # \ "\text{quantum NOT}"\)
\(X \ 0\)
\(H \ 0 \ # \ \text{Hadamard gate}\)
\(CNOT \ 0 \ 1\)

\[\psi = \frac{1}{\sqrt{2}} [0, 0, 0, 1]\]

# Move quantum data to classical data
# MEASURE <qubit register> [ <bit register> ]

MEASURE 0 [2]
How do I program a quantum computer?

Quil (Quantum Instruction Language) gives each quantum operation an instruction

\[ \psi = \frac{1}{\sqrt{2}} [1, 0, 0, 1] \]

\[ \psi = [1, 0, 0, 0] \]

\[ \psi = [0, 0, 0, 1] \]

Classical Bit Register

0 0 0 0 ... 0 0 1 0 ...
Quantum programming is preparing and sampling from complicated distributions.

1. Send program
   e.g.
   \[X 0\]
   \[CNOT 0 1\]

2. Prep Distribution

3. Sample
The Quil Programming Model

Targets a Quantum Abstract Machine (QAM) with a syntax for representing state transitions

\[ \Psi: \text{Quantum state (qubits)} \rightarrow \text{quantum instructions} \]

\[ C: \text{Classical state (bits)} \rightarrow \text{classical and measurement instructions} \]

\[ \kappa: \text{Execution state (program)} \rightarrow \text{control instructions (e.g., jumps)} \]

# Quil Example

```
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
```
The Quil Programming Model

Targets a Quantum Abstract Machine (QAM) with a syntax for representing state transitions

\[ \Psi: \text{Quantum state (qubits)} \rightarrow \text{quantum instructions} \]
\[ C: \text{Classical state (bits)} \rightarrow \text{classical and measurement instructions} \]
\[ \kappa: \text{Execution state (program)} \rightarrow \text{control instructions (e.g., jumps)} \]

0. Initialize into zero states

QAM: \( \Psi_0, C_0, \kappa_0 \)

1. Hadamard on qubit 3

QAM: \( \Psi_1, C_0, \kappa_1 \)

# Quil Example

\[
\begin{align*}
\text{H 3} \\
\text{MEASURE 3 [4]} \\
\text{JUMP-WHEN @END [5]} \\
. \\
. \\
. 
\end{align*}
\]
The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)** with a syntax for representing state transitions

\[
\Psi : \text{Quantum state (qubits)} \rightarrow \text{quantum instructions}
\]

\[
C : \text{Classical state (bits)} \rightarrow \text{classical and measurement instructions}
\]

\[
\kappa : \text{Execution state (program)} \rightarrow \text{control instructions (e.g., jumps)}
\]

QAM: \(\Psi_0, C_0, \kappa_0\)

1. Hadamard on qubit 3

2. Measure qubit 3 into bit #4

# Quil Example

\[H \ 3\]

\[\text{MEASURE } 3 \ [4]\]

\[\text{JUMP-WHEN } @\text{END} \ [5]\]

•

•

•
The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)** with a syntax for representing state transitions

- **\(\Psi\):** Quantum state (qubits) \(\rightarrow\) quantum instructions
- **\(C\):** Classical state (bits) \(\rightarrow\) classical and measurement instructions
- **\(\kappa\):** Execution state (program) \(\rightarrow\) control instructions (e.g., jumps)

### Initial state
QAM: \(\Psi_0, C_0, \kappa_0\)

### State transitions
1. **Hadamard on qubit 3**

   \(\Psi_1, C_0, \kappa_1\)

2. **Measure qubit 3 into bit #4**

3. **Jump to end of program if bit #5 is TRUE**

   \(\Psi_2, C_0(\kappa_3)\)

### Quil Example

```
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
```

```
... 
```

```
... 
```
The Quil Programming Model

Quantum/classical synchronization with *shared classical state*.

Quantum Processor

- H 0
- CNOT 0 1
- MEASURE 0 [7]
- MEASURE 1 [3]
- WAIT

Classical Processor

```python
if C[3] + C[7] == 2:
    theta = 3*pi/7

... continue_from_wait()
```

C: Classical Shared Memory (bits)

0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
Forest

A Hybrid Cloud Quantum Programming Environment

Superconducting Processors & Simulators

Cloud Compute Backends

19Q
Quantum Virtual Machine

Software Dev Kit

Quil Instr. Set
pyQuil
grove (VQE, QAOA, etc...)

Open Source Python libraries

Join us on Github

github.com/rigetticomputing
PyQuil

Control Computer

Pulse program

QPU

Qubit operations

01010110001...

Readout
Sign up in one click

Forest

An API for quantum computing in the cloud

Get started for free

Full Name
Email address

EMAIL API KEY

rigetti.com/forest

Forest is a developer environment for quantum programming...
We need hybrid programming because of errors

Chance of hardware error in a classical computer:

0.000,000,000,000,000,000,000,000,1 %

Chance of hardware error in a quantum computer:

0.1%
Quantum programming is preparing and sampling from complicated distributions.

1. Send program
   e.g.
   \[ X 0 \]
   \[ \text{CNOT } 0 1 \]

2. Prep Distribution

3. Sample
How do I program a quantum computer?

By parameterizing quantum programs we can train them to be robust to noise.

1. Send program e.g. RX(θ) 2
2. Prep Distribution
3. Sample
4. Optimize choice of θ against some objective
Quantum Machine Learning

- Quantum neuron: an elementary building block for machine learning on quantum computers. (Cao et al. 2017)

- Quantum circuit learning. (Mitarai et al. 2018)

- Quantum machine learning in feature Hilbert spaces. (Schuld and Killoran 2018)

A generative modeling approach for benchmarking and training shallow quantum circuits. (Benedetti et al. 2018)
The Variational Quantum Eigensolver

1. MOLECULAR DESCRIPTION
e.g. Electronic Structure Hamiltonian

\[ H = \sum_{i<j}^{N_e} \frac{Z_i Z_j}{|R_i - R_j|} + \sum_{i}^{N_e} -\frac{\nabla^2 \rho_i}{2} - \sum_{i<j}^{N_e} \frac{Z_i}{|R_i - R_j|} + \sum_{i<j}^{N_e} \frac{1}{|r_i - r_j|}. \]

2. MAP TO QUBIT REPRESENTATION
e.g. Bravyi-Kitaev or Jordan-Wigner Transform
e.g. DI-HYDROGEN

\[ H = f_0 \mathbf{1} + f_1 Z_0 + f_2 Z_1 + f_3 Z_2 + f_4 Z_0 Z_1 + f_5 Z_1 Z_3 + f_6 X_0 Z_1 X_2 + f_7 Y_0 Z_1 Y_2 + f_8 Z_0 Z_1 Z_2 + f_9 Z_0 Z_2 Z_3 + f_{10} Z_1 Z_2 Z_3 + f_{11} X_0 Z_1 X_2 Z_3 + f_{12} Y_0 Z_1 Y_2 Z_3 + f_{13} Z_0 Z_1 Z_2 Z_3 \]

3. PARAMETERIZED ANSATZ
e.g. Unitary Coupled Cluster Variational Adiabatic Ansatz

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


VQE Simulations on Quantum Hardware

Peruzzo et al. 1304.3061

Kandala et al. 1704.05018

O’Malley et al. 1512.06860
Quantum Approximate Optimization Algorithm

[QAOA] Hybrid algorithm used for constraint satisfaction problems

Given binary constraints:

\[ \begin{aligned}
  z & \in \{0, 1\}^n, \\
  C_a(z) &= \begin{cases} 
    1 & \text{if } z \text{ satisfies the constraint } a \\
    0 & \text{if } z \text{ does not} 
  \end{cases}.
\end{aligned} \]

\[ \text{MAXIMIZE} \quad \sum_{a=1}^{m} C_a(z) \]

Traveling Salesperson  Scheduling  K-means clustering  Boltzmann Machine Training

QAOA in **Forest**

In 14 lines of code

```python
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)]
nodes = range(4)

init_state_prog = sum([H(i) for i in nodes], Program())
h_cost = -0.5 * sum(sI(nodes[0]) - sZ(i) * sZ(j) for i, j in 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.])

qvm = QPUConnection()
qvm.run_and_measure(program, qubits=nodes, trials=10)
```
Open areas in quantum programming

> Debuggers
> Optimizing compilers
> Application specific packages

> Adoption and implementations
Q1. Why program a quantum computer?

New power | New opportunity | Fundamental curiosity

Q2. How do I program a quantum computer?

Hybrid quantum programming using Forest and Quil