CS269: Quantum Computer Programming

Dan Boneh & Will Zeng + Guests
Honey, I think you're old enough to know the truth about quantum mechanics. Quantum superposition... it doesn't mean 0 and 1 at the same time. At least, not the way you think.

The important thing for you to understand is that quantum computing isn't just a matter of trying all the answers in parallel.
IF YOU DON'T TALK TO YOUR KIDS ABOUT QUANTUM COMPUTING...

SOMEONE ELSE WILL.

Quantum computing and consciousness are both weird and therefore equivalent.
This course is:

At the leading edge of a new technology, discipline, and industry

A programming-first approach

A great way to challenge yourself to think about computation in a totally new way

A way to learn “just enough” quantum physics

An experiment!
Course details

Online at: http://cs269q.stanford.edu

Two lectures per week. Tuesday, Thursday 10:30-11:50, McCullough 115

There will be two written problem sets, three programming projects, one final programming project and one final exam.


Readings: posted online with the syllabus for each lecture. These are critical.
Quantum Computing isn’t the answer to everything.

But it will almost certainly free us to solve more problems.
Today’s lecture:

Q1. Why program a quantum computer?

Q2. How do I program a quantum computer?
Classical computers have fundamental limits

Transistor scaling

<table>
<thead>
<tr>
<th>Year</th>
<th>Intel First Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>180 nm</td>
</tr>
<tr>
<td>2001</td>
<td>130 nm</td>
</tr>
<tr>
<td>2003</td>
<td>90 nm</td>
</tr>
<tr>
<td>2005</td>
<td>65 nm</td>
</tr>
<tr>
<td>2007</td>
<td>45 nm</td>
</tr>
<tr>
<td>2009</td>
<td>32 nm</td>
</tr>
<tr>
<td>2011</td>
<td>22 nm</td>
</tr>
<tr>
<td>2014</td>
<td>14 nm</td>
</tr>
<tr>
<td>2016</td>
<td>10 nm</td>
</tr>
<tr>
<td>2017</td>
<td>10 nm</td>
</tr>
<tr>
<td>2018</td>
<td>10 nm?</td>
</tr>
<tr>
<td>2019</td>
<td>10 nm!</td>
</tr>
</tbody>
</table>

Classical computers have fundamental limits

- Transistor scaling: Economic limits with 10bn for next node fab
  - Ultimate single-atom limits
- Returns to parallelization: Amdahl's law
- Energy consumption: Exascale computing project has its own power plant
  - Power density can melt chips
But Requirements for Compute Continue to Grow

Source: https://blog.openai.com/ai-and-compute/
And there’s more we want to do

Simulation Driven Drug Design  Organic Batteries & Solar Cells  Artificial General Intelligence
Why build 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....

Commodore 64

AWS M4 Instance

1 Million x Commodore 64

Entire Global Cloud

1 Billion x

(1 Million x Commodore 64)

can exactly simulate:

10 Qubits

30 Qubits

60 Qubits

Size of today’s systems.

Note these are imperfect qubits.

* We will be more precise later in the lecture
Why build a quantum computer?

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

For **N qubits** every time step (~100ns*) is an exponentially large $2^N \times 2^N$ complex **matrix multiplication**

**Crucial details:**
- limited number of multiplications (hundreds to thousands) due to noise
- not arbitrary matrices (need to be easily constructed on a QC)
- small I/O, **N-bits in and N-bits out**

The “big-memory small pipe” mental model for quantum computing

*for superconducting qubit systems*
### Why build a quantum computer?

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

<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>

---

**What isn’t on here:** breaking RSA with Shor’s algorithm
Quantum hardware development is accelerating

Plotted by number of qubits in development

1927: Quantum Theory
1982: Quantum Computer
2000: 7 qubits Los Alamos
2006: 128 qubits DWave
2011: 12 qubits DWave
2013: 7 qubits DWave
2015: 17 qubits IBM
2017: 50 qubits IBM
2018: 1152 qubits DWave
2018: 2048 qubits Google
2018: 128 qubits Rigetti

55 YEARS
18 YEARS
6 YEARS
1 YEAR

Image: Strangeworks
Quantum Hardware comes in many forms
Photonic Quantum Computers Use Light

Image: Xanadu
Superconducting Qubits are Supercooled RF Circuits
Why build 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

- **United States**: 360
- **Canada**: 100
- **Brazil**: 11
- **China**: 220
- **Germany**: 120
- **France**: 52
- **Japan**: 63
- **South Korea**: 13

**Source**: McKinsey

*Combined estimated budget of EU countries

Plus approximately $400M in global VC investment
Large Companies are involved
In a growing ecosystem of startups and incumbents
QUANTUM COMPUTING PATENT FAMILIES BY CATEGORY AND PUBLICATION YEAR

- Qubit
- Hardware
- Applications
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?

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

Images: Rigetti
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
Images: Rigetti
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

---

How do I program a quantum computer?

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

The Quil\(^{01}\) instruction set is optimized for this.

---

How do I program a quantum computer?

Quantum programming is preparing and sampling from complicated distributions

1. Send program
   e.g.
   X 0
   CNOT 0 1

2. Prep Distribution

3. Sample
### 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 $\in {0, 1}$</th>
<th>Probabilistic Bits</th>
<th>Qubits $a + b \in \mathbb{R}^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
<td>$a + b = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits**
- State (single unit)
- Bit $\in \{0, 1\}$

**Probabilistic Bits**
- Real vector $\vec{b} = a\vec{0} + b\vec{1}$
- $a + b = 1$
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 $\in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td></td>
<td>Probability of 0</td>
<td>$a + b = 1$</td>
</tr>
</tbody>
</table>

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

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

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 $\in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td></td>
<td>$a + b \in \mathbb{R^+}$</td>
<td>$a + b = 1$</td>
</tr>
<tr>
<td></td>
<td>Complex vector $\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$</td>
<td>$</td>
</tr>
</tbody>
</table>

Classical Bit

0

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

\( |\alpha|^2 = \text{Probability of } 0 \)

\( |\beta|^2 = \text{Probability of } 1 \)
| State (single unit) | Bits $\in \{0, 1\}$ | Probabilistic Bits $a + b \in \mathbb{R}_+$ | Qubits Complex vector $|\alpha|^2 + |\beta|^2 = 1$ |
|---------------------|---------------------|---------------------------------|---------------------------------|
|                     | $\vec{b} = a\vec{0} + b\vec{1}$ | $a + b = 1$ | $\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$ |

$$\text{coin} = \frac{1}{2}\vec{0} + \frac{1}{2}\vec{1}$$
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>Real vector [ \vec{b} = a\vec{0} + b\vec{1} ]</td>
<td>Complex vector [ \vec{\psi} = \alpha\vec{0} + \beta\vec{1} ]</td>
</tr>
<tr>
<td>Bit ( \in {0, 1} )</td>
<td>( a + b \in \mathbb{R}_+ ) ( a + b = 1 )</td>
<td>( \alpha, \beta \in \mathbb{C} )</td>
</tr>
</tbody>
</table>

\[ |\alpha|^2 + |\beta|^2 = 1 \]

CLASSICAL BIT

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

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

\[
\text{qcoin} = \frac{1}{\sqrt{2}}\vec{0} - \frac{1}{\sqrt{2}}\vec{1}
\]

\[
\text{qcoin} = \frac{1}{\sqrt{2}}\vec{0} - \frac{i}{\sqrt{2}}\vec{1}
\]

...
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 $\in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td></td>
<td>$a + b = 1$</td>
<td>$</td>
</tr>
</tbody>
</table>

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

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


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

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

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

<table>
<thead>
<tr>
<th>State (single unit)</th>
<th>Bit $\in {0, 1}$</th>
<th>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</th>
<th>Complex vector $\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a + b \in \mathbb{R}_+$</td>
<td>$a + b = 1$</td>
<td>$</td>
</tr>
<tr>
<td>State (multi-unit)</td>
<td>Bitstring $x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector) $\vec{s} = {p_x}_{x \in {0, 1}^n}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\vec{s} = \bigotimes_{i} b_i$</td>
<td></td>
</tr>
</tbody>
</table>

Probability of bitstring $x$
# How do I program a quantum computer?

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

<table>
<thead>
<tr>
<th></th>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State (single unit)</strong></td>
<td>Bit $\in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$</td>
<td>Complex vector $\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$</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 $x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector) $\vec{s} = {p_x}_{x \in {0,1}^n}$</td>
<td>Wavefunction (complex vector) $\vec{\psi} = {\alpha_x}_{x \in {0,1}^n}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\vec{s} = \bigotimes_i b_i$</td>
<td>$\vec{\psi} = \bigotimes_i \psi_i$</td>
</tr>
</tbody>
</table>
How do I program a quantum computer?

<table>
<thead>
<tr>
<th></th>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State (single unit)</strong></td>
<td><strong>Bit</strong> $\in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$, $a + b \in \mathbb{R}_+$</td>
<td>Complex vector $\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$, $\alpha, \beta \in \mathbb{C}$, $</td>
</tr>
<tr>
<td><strong>State (multi-unit)</strong></td>
<td><strong>Bitstring</strong> $x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector) $\vec{s} = {p_x}_{x \in {0,1}^n}$</td>
<td>Wavefunction (complex vector) $\vec{\psi} = {\alpha_x}_{x \in {0,1}^n}$</td>
</tr>
</tbody>
</table>

$\vec{s} = \bigotimes_{i=1}^{n} b_i$

$\vec{\psi} = \bigotimes_{i=1}^{n} \psi_i$

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

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

<table>
<thead>
<tr>
<th></th>
<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 $\vec{b} = a\vec{0} + b\vec{1}$ $a + b = 1$</td>
<td>Complex vector $\vec{\psi} = \alpha\vec{0} + \beta\vec{1}$ $</td>
</tr>
<tr>
<td><strong>State (multi-unit)</strong></td>
<td>Bitstring $x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector) $\vec{s} = {p_x}_{x \in {0,1}^n}$</td>
<td>Wavefunction (complex vector) $\vec{\psi} = {\alpha_x}_{x \in {0,1}^n}$</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>Boolean Logic</td>
<td>Stochastic Matrices</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sum_{j=1}^{s} P_{i,j} = 1.$</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of quantum states and operations](image-url)
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 $\in {0, 1}$</td>
<td>Real vector $\vec{b} = a\vec{0} + b\vec{1}$, $a + b \in \mathbb{R}_+$, $a + b = 1$</td>
</tr>
<tr>
<td>State (multi-unit)</td>
<td>Bitstring $x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector) $\vec{s} = {p_x}_{x \in {0,1}^n}$</td>
</tr>
<tr>
<td>Operations</td>
<td>Boolean Logic</td>
<td>Stochastic Matrices $\sum_{j=1}^s P_{i,j} = 1.$</td>
</tr>
</tbody>
</table>

![Diagram](image)
# 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>$\text{Bit } \in {0, 1}$</td>
<td>Real vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\vec{b} = a\vec{0} + b\vec{1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a + b \in \mathbb{R}_+$</td>
</tr>
<tr>
<td>State (multi-unit)</td>
<td>$x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector)</td>
</tr>
<tr>
<td></td>
<td>$\vec{s} = {p_x}_{x \in {0,1}^n}$</td>
<td>$\psi = {\alpha_x}_{x \in {0,1}^n}$</td>
</tr>
<tr>
<td>Operations</td>
<td>Boolean Logic</td>
<td>Stochastic Matrices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sum_{j=1}^{s} P_{i,j} = 1.$</td>
</tr>
<tr>
<td>Component Ops</td>
<td>Boolean Gates</td>
<td>Tensor products of matrices</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>State (single unit)</td>
<td>Bit $\in {0, 1}$</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></td>
<td></td>
<td>$a + b = 1$</td>
</tr>
<tr>
<td>State (multi-unit)</td>
<td>Bitstring $x \in {0, 1}^n$</td>
<td>Prob. Distribution (stochastic vector) $\vec{s} = {p_x}_{x \in {0,1}^n}$</td>
</tr>
<tr>
<td>Operations</td>
<td>Boolean Logic</td>
<td>Stochastic Matrices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sum_{j=1}^{s} P_{i,j} = 1.$</td>
</tr>
<tr>
<td>Component Ops</td>
<td>Boolean Gates</td>
<td>Tensor products of matrices</td>
</tr>
</tbody>
</table>
### How do I program a quantum computer?

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

#### Bits
- **State (single unit)**
  - Bit: $\in \{0, 1\}$
  - Real vector: \[ \overrightarrow{b} = a\overrightarrow{0} + b\overrightarrow{1} \quad a + b = 1 \]
- **State (multi-unit)**
  - Bitstring: $x \in \{0, 1\}^n$
  - Prob. Distribution: \( \overrightarrow{s} = \{p_x\}_{x \in \{0,1\}^n} \)
- **Operations**
  - Boolean Logic
  - Stochastic Matrices: \( \sum_{j=1}^{s} P_{i,j} = 1. \)
- **Component Ops**
  - Boolean Gates

#### Probabilistic Bits
- **State (single unit)**
  - Bit: $\in \{0, 1\}$
  - Real vector: \[ \overrightarrow{b} = a\overrightarrow{0} + b\overrightarrow{1} \quad a + b = 1 \]
- **State (multi-unit)**
  - Bitstring: $x \in \{0, 1\}^n$
  - Prob. Distribution: \( \overrightarrow{s} = \{p_x\}_{x \in \{0,1\}^n} \)
- **Operations**
  - Boolean Logic
  - Stochastic Matrices: \( \sum_{j=1}^{s} P_{i,j} = 1. \)
- **Component Ops**
  - Boolean Gates

#### Qubits
- **State (single unit)**
  - Bit: $\in \{0, 1\}$
  - Complex vector: \[ \overrightarrow{\psi} = a\overrightarrow{0} + b\overrightarrow{1} \quad |a|^2 + |b|^2 = 1 \]
- **State (multi-unit)**
  - Bitstring: $x \in \{0, 1\}^n$
  - Wavefunction: \( \overrightarrow{\psi} = \{\alpha_x\}_{x \in \{0,1\}^n} \)
- **Operations**
  - Boolean Logic
  - Unitary Matrices: \( U^†U = 1 \)
- **Component Ops**
  - Boolean Gates

#### Sampling
- **Born rule**
  - $|\alpha_x|^2 = \text{Probability of bitstring } x$

---

**Note:** The table summarizes the differences between classical bits, probabilistic bits, and qubits in the context of quantum computing. The Born rule is derived from the quantum state and is used to calculate the probability of measuring a particular bitstring.
### How do I program a quantum computer?

<table>
<thead>
<tr>
<th></th>
<th>Bits</th>
<th>Probabilistic Bits</th>
<th>Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State (single unit)</strong></td>
<td>Bit</td>
<td>Real vector ( \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>
<tr>
<td></td>
<td>( \ell \in {0, 1} )</td>
<td></td>
<td>( \alpha, \beta \in \mathbb{C} )</td>
</tr>
<tr>
<td><strong>State (multi-unit)</strong></td>
<td>Bitstring ( x \in {0, 1}^n )</td>
<td>Prob. Distribution (stochastic vector) ( \vec{s} = {p_x}_{x \in {0, 1}^n} )</td>
<td>Wavefunction (complex vector) ( \vec{\psi} = {\alpha_x}_{x \in {0, 1}^n} )</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>Boolean Logic</td>
<td>Stochastic Matrices ( \sum_{j=1}^{s} P_{i,j} = 1 ).</td>
<td>Unitary Matrices ( 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
- Born rule
- Measurement
Quil (Quantum Instruction Language) gives each quantum operation an instruction

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

Start in 0
How do I program a quantum computer?

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

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

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

\[
\text{X \ 0 \ # “quantum NOT”}
\]
How do I program a quantum computer?

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

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

Apply X instr to 0th qubit

X \( \theta \) # “quantum NOT”

\[ \psi = [0, 1, 0, 0] \]
How do I program a quantum computer?

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

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

\[ \psi = \begin{bmatrix} 0_0 & 0_1 & 0_{10} & 0_{11} \end{bmatrix} \]

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

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

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

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

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

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

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

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

X 0 # “quantum NOT”

H 0 # Hadamard gate

Apply H instr to 0th qubit

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

\[ H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \]
Quil (Quantum Instruction Language) gives each quantum operation an instruction

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

\[X \; 0 \; \# \; \text{“quantum NOT”}\]

\[X \; 0\]

\[H \; 0 \; \# \; \text{Hadamard gate}\]

\[\text{CNOT} \; 0 \; 1\]

\[
\psi = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix}_{00 \; 01 \; 10 \; 11}
\]

\[
\psi = \begin{bmatrix} 0, & 1, & 0, & 0 \end{bmatrix}_{00 \; 01 \; 10 \; 11}
\]

\[
\psi = \begin{bmatrix} 1/\sqrt{2}, & 1/\sqrt{2}, & 0, & 0 \end{bmatrix}_{00 \; 01 \; 10 \; 11}
\]

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

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

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

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

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

Apply CNOT instr to 0 and 1 qubits

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

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

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

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

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

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

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

\( X \, \theta \) # “quantum NOT”

\( X \theta \)

H \( \theta \) # Hadamard gate

\( \text{CNOT} \, \theta \, 1 \)

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

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

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

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

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

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

Qubits 0 and 1 are ENTANGLED.
How do I program a quantum computer?

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

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

\begin{verbatim}
X 0 # “quantum NOT”
X 0
H 0 # Hadamard gate
CNOT 0 1
\end{verbatim}

\[
\psi = \begin{bmatrix} 1/\sqrt{2}, 0, 0, 1/\sqrt{2} \\ 00 & 01 & 10 & 11 \end{bmatrix}
\]
How do I program a quantum computer?

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

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

\begin{align*}
X 0 & \quad \text{“quantum NOT”} \\
X 0 \\
H 0 & \quad \text{Hadamard gate} \\
\text{CNOT} 0 1 & \\
\text{MEASURE} & \text{<qubit register> [<bit register>]}
\end{align*}
How do I program a quantum computer?

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

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

MEASURE 0 [2]

ψ = [1/√2, 0, 0, 1/√2]
How do I program a quantum computer?

Some more examples of MEASUREMENT

\[ \psi = \begin{bmatrix} 1/2, & 0, & 0, & \sqrt{3}/4 \end{bmatrix} \]

Quantum Memory

Classical Memory

25%

75%

MEASURE 1 [3]
How do I program a quantum computer?

Some more examples of MEASUREMENT

\[ \psi = \begin{bmatrix} 1/2, & 0, & 0, & \sqrt{3}/4 \end{bmatrix} \]

MEASURE 1 [3]

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

Quantum Memory

Classical Memory

25%

75%
How do I program a quantum computer?

Some more examples of MEASUREMENT

\[ \psi = \begin{bmatrix} 1/2, \ 0, \ 0, \ \sqrt{3}/4 \end{bmatrix} \]

\[ \rightarrow \text{ MEASURE 1}[3] \]

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

\[ \rightarrow \text{ MEASURE 1}[3] \]

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

Quantum Memory

Classical Memory

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{array}
\]
How do I program a quantum computer?

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

Some more examples of MEASUREMENT

\[ \psi = \begin{bmatrix} 1/2, \ 0, \ 0, \ \sqrt{3}/4 \end{bmatrix}_{00 \ 01 \ 10 \ 11} \]

25% MEASURE 1 [3]

\[ \psi = \begin{bmatrix} 1, \ 0, \ 0, \ 0 \end{bmatrix}_{00 \ 01 \ 10 \ 11} \]

00 00 00 00 ...

[0] [1] [2] [3]

0 0 0 0 1 ...

[0] [1] [2] [3]

75%

\[ \psi = \begin{bmatrix} 0, \ 0, \ 0, \ 1 \end{bmatrix}_{00 \ 01 \ 10 \ 11} \]

MEASURE 1 [3]

\[ \psi = \begin{bmatrix} 0, \ 0, \ 0, \ 1 \end{bmatrix}_{00 \ 01 \ 10 \ 11} \]

100% MEASURE 1 [3]

\[ \psi = \begin{bmatrix} 1/\sqrt{2}, \ 1/\sqrt{2}, \ 0, \ 0 \end{bmatrix}_{00 \ 01 \ 10 \ 11} \]
How do I program a quantum computer?

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

Some more examples of MEASUREMENT

\[ \psi = \begin{bmatrix} 1/2 & 0 & 0 & \sqrt{3}/4 \end{bmatrix} \]

\( \xrightarrow{\text{25\%}} \)  
\( \psi \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \)

\( \psi = \begin{bmatrix} 1, 0, 0, 0 \end{bmatrix} \)

\( \xrightarrow{\text{MEASURE 1 [3]}} \)

\( \psi = \begin{bmatrix} 0, 0, 0, 1 \end{bmatrix} \)

\( \xrightarrow{\text{75\%}} \)

\( \psi = \begin{bmatrix} 0, 0, 0, 1 \end{bmatrix} \)

\( \xrightarrow{\text{100\%}} \)

\( \psi = \begin{bmatrix} 1/\sqrt{2}, 1/\sqrt{2}, 0, 0 \end{bmatrix} \)

\( \xrightarrow{\text{MEASURE 1 [3]}} \)

\( \psi = \begin{bmatrix} 1/\sqrt{2}, 1/\sqrt{2}, 0, 0 \end{bmatrix} \)

\( \xrightarrow{\text{100\%}} \)

\( \psi = \begin{bmatrix} 1/\sqrt{2}, 1/\sqrt{2}, 0, 0 \end{bmatrix} \)
How do I program a quantum computer?

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

\( \Psi_1, C_0, \kappa_1 \)

# Quil Example

\texttt{H 3}

\texttt{MEASURE 3 [4]}

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

\texttt{.}

\texttt{.}

\texttt{.}
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

\( \Psi_1, C_0, \kappa_1 \)

2. Measure qubit 3 into bit #4

Outcome 0

Outcome 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

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

QAM: Ψ₀, C₀, κ₀

1. Hadamard on qubit 3
(Ψ₁, C₀, κ₁)

2. Measure qubit 3 into bit #4

Outcome 0
(Ψ₂, C₀, κ₀)

Outcome 1
(Ψ₃, C₁, κ₀)

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

Ψ₂, C₀(κ₃)

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

...
Quantum Computing Programming Languages

**Quantum Universal Languages**
- Pennylane
- XACC
- ProjectQ
- Cirq

**Full-stack libraries**
- IBM: QISKit
- Rigetti: Forest
- DWave: QAme
- Xanadu: Strawberry Fields
- Google: Cirq
- Microsoft*: Cirq
- Qilimanjaro*: Quantum Development Kit

**Quantum algorithms**
- QISKit Aqua
- Grove
- QSage ToQ
- OpenFermion - Cirq
- Q#

**Quantum circuits**
- QISKit Terra
- pyquil
- qbsolv

**Assembly language**
- Open QASM
- Quill
- QMASM
- Blackbird
- Other Quantum Machine Instruction Languages

**Hardware**
- Qibo

* Hardware under development. Quantum programs are run on their own simulators.

"Quantum Language" is referred with no distinction both as a quantum equivalence of a programming language and as a library to write quantum programs supported by some well-known classical programming language.
Main tools in this course. All OSS Apache v2

Quantum Computing Programming Languages

Quantum Universal Languages

Full-stack libraries

Quantum algorithms

Quantum circuits

Assembly language

Hardware

Quantum device

* Hardware under development. Quantum programs are run on their own simulators.

“Quantum Language” is referred with no distinction both as a quantum equivalence of a programming language and as a library to write quantum programs supported by some well-known classical programming language.

© Alba Cervera-Lierta for the QWA (2018)
1. From the composer/QIS Kit, the job is sent to the cloud where it is queued and then sent to a control/measurement computer.

2. Microwave electronics mix signal down to a frequency that can be digitized.

3. Measurement pulses go down the same cables after the control pulses.

4. Measurement pulses interact with qubits via readout resonators and are reflected back.

5. Measurement pulses are routed by circulators, and isolators prevent noise from getting to the qubits.

6. Amplifiers at 4K.

7. Microwave electronics mix signal down to a frequency that can be digitized.

8. Mixed-down signals are digitized by a classical computer and the result is classified as 0 or 1.

9. Results are sent back to you over the cloud.
We need hybrid programming because of errors

Chance of hardware error in a classical computer:

0.000,000,000,000,000,000,000,000,000.1 %

Chance of hardware error in a quantum computer:

0.1%
How do I program a quantum computer?

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

Quantum programming is preparing and sampling from complicated distributions.

1. Send program
   e.g.
   X 0
   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

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

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

2. MAP TO QUBIT REPRESENTATION

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

**DI-HYDROGEN**

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

3. PARAMETERIZED ANSATZ

e.g. Unitary Coupled Cluster Variational Adiabatic Ansatz

\[ \frac{\langle \varphi(\bar{\theta}) | H | \varphi(\bar{\theta}) \rangle}{\langle \varphi(\bar{\theta}) | \varphi(\bar{\theta}) \rangle} \geq E_0 \]

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

**QUANTUM PROCESSOR**

PREPARE QUANTUM STATE \((\theta)\)

\[ \langle H_1 \rangle \]

\[ \langle H_2 \rangle \]

\[ \vdots \]

\[ \langle H_N \rangle \]

**CLASSICAL PROCESSOR**

SUM TERMS

\[ \sum_i \langle H_i \rangle \]

CLASSICAL OPTIMIZATION OF ANSATZ PARAMETER: \(\theta\)
VQE Simulations on Quantum Hardware

Peruzzo et al. 1304.3061

O’Malley et al. 1512.06860

Kandala et al. 1704.05018
Quantum Approximate Optimization Algorithm

[QAOA] Hybrid algorithm used for constraint satisfaction problems

Given binary constraints:

\[ z \in \{0, 1\}^n. \]

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

MAXIMIZE

\[ C(z) = \sum_{a=1}^{m} C_a(z) \]

---

**Traveling Salesperson**  **Scheduling**  **K-means clustering**  **Boltzmann Machine Training**

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

graph = [(0, 1), (1, 2), (2, 3), (3, 0)]
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) + \n                exponentiate_commuting_pauli_sum(h_driver)(b) \n                for g, b in zip(gammas, betas)], Program())

def qaoa_cost(params):
    half = int(len(params)/2)
    betas, gammas = params[:half], params[half:]
    program = init_state_prog + qaoa_ansatz(betas, gammas)
    return WavefunctionSimulator().expectation(prep_prog=program, pauli_terms=h_cost)

minimize(qaoa_cost, x0=[0., 0.5, 0.75, 1.], method='Nelder-Mead', options={'disp': True})
Open areas in quantum programming

- Debuggers
- Optimizing compilers
- Application specific packages
- Adoption and implementations

Diagram showing connections between different quantum computing frameworks and tools.
Q1. Why program a quantum computer?
   New power | New opportunity | Fundamental curiosity

Q2. How do I program a quantum computer?
   Hybrid quantum programming (usually) in Python!
Course Topics & Timeline

Introduction
Linear Algebra
Quantum Mechanics

Low Level Programming
Quantum Ops
Instruction Sets
Classical Ctrl

Noise & Benchmarking

Hybrid Algorithms
VQE Simulation
QAOA
Optimization

Hardware & Compilation

Hybrid Algs (QML)

Error Correction & Error Corrected Algorithms
Shor’s Factoring Algorithm
Grover’s Search Algorithm

Other Topics
MBQC
Blind QC

Problem Set 1
Programming Proj. 1
Programming Proj. 2
Programming Proj. 3
Final Programming Project

Exam

Lecture 1
Lecture 20
Actions for between now and the next lecture:

1. Read the syllabus.

2. Read Mike & Ike Chapters 1 & 2. Especially review Sections 2.2, 2.3 & 2.6.

3. Review Linear Algebra. You will need:
   - Vectors and linear maps
   - Bases and linear independence
   - Pauli Matrices
   - Inner Products
   - Eigenvalues & Eigenvectors
   - Adjoints
   - Hermitian Operators
   - Unitary Matrices
   - Tensor Products
   - Matrix Exponentials
   - Traces
   - Commutators and Anti-commutators

4. Download and install pyQuil: https://pyquil.readthedocs.io