MITRE’s Research in Quantum Software Engineering and Workforce Education

Joe Clapis

October 20, 2021

Approved for Public Release; Distribution Unlimited. Public Release Case Number 21-2797
# About MITRE’s Quantum Software Group

<table>
<thead>
<tr>
<th>Paper Name</th>
<th>Use Case</th>
<th>Speedup</th>
<th>SDK</th>
<th>MCX Mode</th>
<th>Backend</th>
<th>Circuit Creation Time</th>
<th>Compile Time</th>
<th>Width</th>
<th>NEQR Depth</th>
<th>QDP Depth</th>
<th>QFT Depth</th>
<th>Total Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Quantum Pattern Recognition Method for Improving Pairwise Sequence Alignment</td>
<td>Aligning multiple sequences of data</td>
<td>Exponential</td>
<td>QDK</td>
<td>---</td>
<td>Resource</td>
<td>00:01.8</td>
<td>---</td>
<td>39</td>
<td>150,239</td>
<td>5</td>
<td>178</td>
<td>150,417</td>
</tr>
<tr>
<td>Bayesian deep learning on a quantum computer</td>
<td>Machine Learning</td>
<td>Polynomial</td>
<td>Qiskit</td>
<td>CCNOT</td>
<td>Chain</td>
<td>00:45.2</td>
<td>05:07.3</td>
<td>39</td>
<td>384,312</td>
<td>4</td>
<td>49</td>
<td>384,350</td>
</tr>
<tr>
<td>Quantum algorithm for visual tracking</td>
<td>Tracking an object through video frames</td>
<td>Quadratic</td>
<td>Qiskit</td>
<td>Single-Qubit</td>
<td>Aer Sim</td>
<td>03:54.6</td>
<td>18:38.7</td>
<td>30</td>
<td>1,940,132</td>
<td>4</td>
<td>49</td>
<td>1,940,173</td>
</tr>
<tr>
<td>An improved quantum algorithm for ridge regression</td>
<td>ML analyzing data suffering from &quot;multicollinearity&quot; with linear regression</td>
<td>Exponential / Polynomial, Depends on Input</td>
<td>Qiskit</td>
<td>CCNOT Chain</td>
<td>Rochester</td>
<td>00:53.9</td>
<td>4:34:26.7</td>
<td>53</td>
<td>1,027,772</td>
<td>20</td>
<td>1108</td>
<td>1,028,758</td>
</tr>
<tr>
<td>Quantum image matching (This paper is wrong, see next row)</td>
<td>Finding a smaller &quot;target&quot; image within a larger overall image</td>
<td>Quadratic?</td>
<td>Qiskit</td>
<td>Single-Qubit</td>
<td>Rochester</td>
<td>DNF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis and improvement of the quantum image matching</td>
<td>Finding a smaller &quot;target&quot; image within a larger overall image</td>
<td>Quadratic</td>
<td>Qiskit</td>
<td>CCNOT Chain</td>
<td>AQT Innsbruck</td>
<td>00:43.8</td>
<td>30:20.3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCM-based quantum artificial bee colony algorithm for image segmentation</td>
<td>Classifying pixels based on which region (e.g. foreground, background) they belong to</td>
<td>???</td>
<td>Qiskit</td>
<td>Single-Qubit</td>
<td>AQT Innsbruck</td>
<td>02:53.0</td>
<td>2:50:15.7</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image classification based on quantum K-Nearest-Neighbor algorithm</td>
<td>Image classification (selecting which of a pre-defined set of groups a picture belongs to)</td>
<td>Quadratic?</td>
<td><a href="https://doi.org/10.1007/s11128-017-1723-7">https://doi.org/10.1007/s11128-017-1723-7</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="https://doi.org/10.1145/3240676.3240907">https://doi.org/10.1145/3240676.3240907</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2021 THE MITRE CORPORATION. ALL RIGHTS RESERVED. APPROVED FOR PUBLIC RELEASE.
Origins

March 6, 2017

IBM Building First Universal Quantum Computers for Business and Science
- IBM unveils roadmap for commercial “IBM Q” quantum systems
- Releases API for developers to build interfaces between quantum computers and classical computers
- Following Watson and blockchain, quantum computing to deliver next powerful set of services on IBM Cloud platform

December 11, 2017

Announcing the Microsoft Quantum Development Kit

Just a few months back, Microsoft CEO Satya Nadella shared our vision of empowering the quantum revolution with bold investments towards a scalable end-to-end solution, revolutionary topological approach, and a global team. Today, we take the next step in this journey with the Microsoft Quantum Development Kit to help you get started with quantum development.

June 20, 2017

Introducing Forest 1.0

Today, I’m extremely excited to announce the public beta availability of Forest 1.0, the world’s first full-stack programming and execution environment for quantum/classical computing.
Quantum full-stack libraries

C++

- XACC - Extreme-scale programming model for quantum acceleration within high-performance computing.

JavaScript

- Qiskit-JS - Quantum information software kit for JavaScript (supported by IBM).

Python

- Cirq - Framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits.
- Forest - Rigetti's software library for writing, simulating, compiling and executing quantum programs.
- Ocean - D-Wave System's suite of tools for solving hard problems with quantum computers.
- ProjectQ - Hardware-agnostic framework with compiler and simulator with emulation capabilities.
- Qiskit - Framework for working with noisy quantum computers at the level of pulses, circuits, and algorithms (supported by IBM).
- Strawberry Fields - Xanadu's software library for photonic quantum computing.

Q#

- Q# - Microsoft’s quantum programming language with Visual Studio integration.

List maintained by the Quantum Open Source Foundation: https://github.com/qosf/os_quantum_software
Initial Exploration

Quantum Software Framework Evaluation

This project contains the code written by MITRE during our evaluation of the most prominent quantum computing software frameworks in 2019. The evaluation is meant to gauge the software engineering experience in using each framework on a daily basis, to assess their strengths, weaknesses, and applicability to our work program.

Evaluation Criteria

Each framework is evaluated and scored according to these criteria:

- **Ease of Use** (learning curve, documentation, language features, control flow, development tooling, debugging support)
- **Maturity and Activity** (community size, frequency of updates, support, standard library functionality)
- **Flexibility and Modularity** (algorithm support, platform independence, compilation process, open source status)
- **Additional Criteria** (compiler and simulator performance, quantum error correction support, hardware tooling support, cost of access and training)

Source: https://github.com/jclapis/qsfe
# Framework Analysis

**Q#**

```qsharp
operation MultiControl(Qubits : Qubit[]) : Unit {
    let length = Length(Qubits);
    let controls = Qubits[0..length - 2];
    let target = Qubits[length - 1];

    // ApplyToEach is a helper function that basically runs the given operation
    // (in this case, the H gate) on each qubit in the given register.
    ApplyToEach(H, controls);

    // Controlled X means the controlled variant of the X gate, using the first
    // argument as the list of control qubits.
    Controlled X(controls, target);
}
```

**Qiskit**

```python
# Hadamard the first three qubits - these will be the controls
for control in controls:
    program += H(control);

# pyQuil supports gates that are controlled by arbitrary many qubits, so
# we don't need to mess with Toffoli gates or custom multi-control implementations.
# We just have to chain a bunch of controlled() calls for each control qubit.
gate = X(target)
for control in controls:
    gate = gate.controlled(control)
program += gate

# Run the test
self.run_test("multi-controlled operation", program, controls + [target], 1000, valid_states)

ancilla = QuantumRegister(1)
circuit.add_register(ancilla)
circuit.x(qubits[0], qubits[1], ancilla[0])
circuit.x(qubits[2], ancilla[0], qubits[3])
circuit.ccx(qubits[0], qubits[1], ancilla[0])

# Run the test
self.run_test("multi-controlled operation", circuit, 1000, valid_states)
```
Framework Analysis

Quantum Software Framework Releases

<table>
<thead>
<tr>
<th>Evaluation Criterion</th>
<th>Qiskit</th>
<th>QDK</th>
<th>Cirq</th>
<th>Forest</th>
<th>ProjectQ</th>
<th>XACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation Process</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>None</td>
</tr>
<tr>
<td>Learning Curve</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Background Theory Training Content</td>
<td>Good</td>
<td>Excellent</td>
<td>Poor</td>
<td>Fair</td>
<td>Poor</td>
<td>Poor</td>
</tr>
<tr>
<td>Documentation (Usage and Examples)</td>
<td>Good</td>
<td>Excellent</td>
<td>Fair</td>
<td>Excellent</td>
<td>Poor</td>
<td>Fair</td>
</tr>
<tr>
<td>Language Features</td>
<td>Fair</td>
<td>Excellent</td>
<td>Fair</td>
<td>Good</td>
<td>Excellent</td>
<td>?</td>
</tr>
<tr>
<td>Debugging Support</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Poor</td>
<td>Good</td>
<td>?</td>
</tr>
<tr>
<td>Community Size</td>
<td>Excellent</td>
<td>Good</td>
<td>Fair</td>
<td>Good</td>
<td>Poor</td>
<td>Poor</td>
</tr>
<tr>
<td>Frequency of Updates</td>
<td>Excellent</td>
<td>Good</td>
<td>Poor</td>
<td>Excellent</td>
<td>Poor</td>
<td>None</td>
</tr>
<tr>
<td>Support Availability</td>
<td>Excellent</td>
<td>Good</td>
<td>Fair</td>
<td>Excellent</td>
<td>Poor</td>
<td>Poor</td>
</tr>
<tr>
<td>Standard Library</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Fair</td>
<td>Good</td>
<td>Fair</td>
<td>?</td>
</tr>
<tr>
<td>Control Flow</td>
<td>Good</td>
<td>Excellent</td>
<td>None</td>
<td>Good</td>
<td>Excellent</td>
<td>?</td>
</tr>
<tr>
<td>Quantum Hardware Independence</td>
<td>Fair</td>
<td>N/A</td>
<td>Good</td>
<td>Poor</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>Open Source Status</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Simulator Performance</td>
<td>Fair</td>
<td>Excellent</td>
<td>Good</td>
<td>Poor</td>
<td>Good</td>
<td>N/A</td>
</tr>
<tr>
<td>Noise Simulation and Error Connection</td>
<td>Excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Excellent</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td>Hardware Tooling Support</td>
<td>Excellent</td>
<td>Poor</td>
<td>Good</td>
<td>Good</td>
<td>Poor</td>
<td>?</td>
</tr>
</tbody>
</table>

*Table 23: Summary of the frameworks' relative scores for each evaluation criterion.*
Now that we know the tools…
What can we build with them?
## Exploring Algorithms

<table>
<thead>
<tr>
<th>Paper Name</th>
<th>Use Case</th>
<th>Speedup</th>
<th>Pub Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Quantum Pattern Recognition Method for Improving Pairwise Sequence Alignment</td>
<td>Aligning multiple sequences of data</td>
<td>Exponential</td>
<td><a href="https://doi.org/10.1038/s41598-019-43697-3">https://doi.org/10.1038/s41598-019-43697-3</a></td>
</tr>
<tr>
<td>Bayesian deep learning on a quantum computer</td>
<td>Machine Learning</td>
<td>Polynomial</td>
<td><a href="https://doi.org/10.1007/s42484-019-00004-7">https://doi.org/10.1007/s42484-019-00004-7</a></td>
</tr>
<tr>
<td>Quantum algorithm for visual tracking</td>
<td>Tracking an object through video frames</td>
<td>Quadratic</td>
<td><a href="https://doi.org/10.1103/PhysRevA.99.022301">https://doi.org/10.1103/PhysRevA.99.022301</a></td>
</tr>
<tr>
<td>An improved quantum algorithm for ridge regression</td>
<td>ML analyzing data suffering from “multicollinearity” with linear regression</td>
<td>Exponential / Polynomial, Depends on Input</td>
<td><a href="https://doi.org/10.1109/TKDE.2019.2937491">https://doi.org/10.1109/TKDE.2019.2937491</a></td>
</tr>
<tr>
<td>Quantum-image-matching (This paper is wrong, see next row)</td>
<td>Finding a smaller “target” image within a larger overall image</td>
<td>Quadratic?</td>
<td><a href="https://doi.org/10.1007/s11128-016-1364-2">https://doi.org/10.1007/s11128-016-1364-2</a></td>
</tr>
<tr>
<td>Analysis and improvement of the quantum image matching</td>
<td>Finding a smaller “target” image within a larger overall image</td>
<td>Quadratic</td>
<td><a href="https://doi.org/10.1007/s11128-017-1723-7">https://doi.org/10.1007/s11128-017-1723-7</a></td>
</tr>
<tr>
<td>FCM-based quantum artificial bee colony algorithm for image segmentation</td>
<td>Classifying pixels based on which region (e.g. foreground, background) they belong to</td>
<td>???</td>
<td><a href="https://doi.org/10.1145/3240876.3240907">https://doi.org/10.1145/3240876.3240907</a></td>
</tr>
<tr>
<td>Image classification based on quantum K-Nearest-Neighbor algorithm</td>
<td>Image classification (selecting which of a pre-defined set of groups a picture belongs to)</td>
<td>Quadratic?</td>
<td><a href="https://doi.org/10.1007/s11128-018-2004-9">https://doi.org/10.1007/s11128-018-2004-9</a></td>
</tr>
</tbody>
</table>
The “Paper Problem”

where the summation symbol denotes the sum over all possible positions. The system can be implemented by repeating the global unitary operator

\[ \hat{U} = \hat{S} \hat{I} \otimes \hat{C}, \]

where \( \hat{I} \) is the identity operator and \( \hat{C} \) is the coin operator applied every \( t \) steps is expressed by

\[ |\psi_t\rangle = (\hat{U})^t |\psi_0\rangle = \sum_{x} \sum_{y} \langle x, y | (\hat{U})^t |\psi_0\rangle, \]

and the probability of locating the walker at position \( x \) after \( t \) steps

\[ P(x, t) = \sum_{y \in \{0, 1\}} \langle x, y | (\hat{U})^t |\psi_0\rangle. \]

where \( |\psi_0\rangle \) is the initial state of the total quantum system.

(3) Add one qubit and rotate it from \( |0\rangle \) to

\[ \sqrt{1 - C_1^2 h^2(\pm \lambda_j, \alpha)} |0\rangle + C_1 h(\pm \lambda_j, \alpha) |1\rangle \]

controlled on \( |\pm \lambda_j \rangle \) where \( h(\lambda, \alpha) := \frac{(N+M)\lambda}{N+1+\alpha} \) and \( C_1 = O(1/\kappa) \). As shown in appendix B, the maximum of \( h(\lambda_j, \alpha) \) as well as \( C_1 \) depends on the actual choice of \( \alpha \), but \( C_1 h(\lambda_j, \alpha) = O(1/\kappa) \) for all possible \( \alpha \). Then we undo phase estimation and obtain

\[ \sum_{j=1}^{R} \beta_j |u_j, \pm v_j\rangle \left( \sqrt{1 - C_1^2 h^2(\pm \lambda_j, \alpha)} |0\rangle + C_1 h(\pm \lambda_j, \alpha) |1\rangle \right). \]

(4) Measure the last qubit to get |1\rangle and project the first register onto the \( v_j \) part. The final state of the first register approximates

\[ |\phi_w\rangle := \frac{\sum_{j=1}^{R} C_1 \beta_j h(\lambda_j, \alpha) |v_j\rangle}{\sqrt{\sum_{j=1}^{R} C_1^2 \beta_j h^2(\lambda_j, \alpha)}} \propto w, \]

2. Performing phase estimation of \( e^{-iX_{to}} \) on the first two registers, we obtain the whole state

\[ \sum_{j=1}^{n} \beta_j \left( \frac{|w^+_j\rangle |\lambda_j\rangle + |w^-_j\rangle |-\lambda_j\rangle}{\sqrt{2}} \right) |0\rangle. \]

(11)

3. Performing a controlled rotation on the last register (qubit) conditioned on the eigenvalue register, we have

\[ \sum_{j=1}^{n} \beta_j \left( \frac{|w^+_j\rangle |\lambda_j\rangle \left( \frac{C_{\lambda_j}}{\sqrt{1 + \alpha}} |1\rangle + \sqrt{1 - \frac{C_{\lambda_j}^2}{1 + \alpha}} |0\rangle \right)}{\sqrt{2}} + \frac{|w^-_j\rangle |-\lambda_j\rangle \left( \frac{C_{\lambda_j}}{\sqrt{1 + \alpha}} |1\rangle + \sqrt{1 - \frac{C_{\lambda_j}^2}{1 + \alpha}} |0\rangle \right)}{\sqrt{2}} \right) |1\rangle. \]

(12)
Practicality Assessment

## Practicality Assessment

<table>
<thead>
<tr>
<th></th>
<th>CUSHAW</th>
<th>SOAP2</th>
<th>Bowtie</th>
<th>BWA</th>
<th>QPR Method (by simulation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>90,00%</td>
<td>90,84%</td>
<td>93,47%</td>
<td>91,82%</td>
<td>94,24% (± 1.48)</td>
</tr>
<tr>
<td>Recall</td>
<td>97,51%</td>
<td>92,85%</td>
<td>84,75%</td>
<td>97,22%</td>
<td>97,83% (± 0.85)</td>
</tr>
<tr>
<td><strong>Average time per read mapping (in ms)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>1,154</td>
<td>2,044</td>
<td>1,343</td>
<td>3,721</td>
<td>0,00642</td>
</tr>
<tr>
<td>PE</td>
<td>2,898</td>
<td>2,310</td>
<td>3,013</td>
<td>3,929</td>
<td></td>
</tr>
<tr>
<td><strong>Approximated runtime (in s)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28,84</td>
</tr>
<tr>
<td>SE</td>
<td>1026</td>
<td>2617</td>
<td>1085</td>
<td>4764</td>
<td></td>
</tr>
<tr>
<td>PE</td>
<td>3711</td>
<td>2958</td>
<td>3858</td>
<td>5031</td>
<td></td>
</tr>
</tbody>
</table>
Practicality Assessment

Figure 1. Circuit model of R. Schützhold’s quantum pattern recognition method.

It is assumed that our aligner is a complex device which can handle large sequences on a lattice plane surface made by non-linear Kerr media and run R. Schützhold’s QPR algorithm in a quantum computing system. The
Practicality Assessment


Figure 6. Complete circuit diagram of QPR, using QDP as the Black Box.
# Practicality Assessment

<table>
<thead>
<tr>
<th>SDK</th>
<th>MCX Mode</th>
<th>Backend</th>
<th>Circuit Creation Time</th>
<th>Compile Time</th>
<th>Width</th>
<th>NEQR Depth</th>
<th>QDP Depth</th>
<th>QFT Depth</th>
<th>Total Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>QDK</td>
<td>---</td>
<td>Resource Counter</td>
<td>00:01.8</td>
<td>---</td>
<td>39</td>
<td>150,239</td>
<td>5</td>
<td>178</td>
<td>150,417</td>
</tr>
<tr>
<td>Qiskit</td>
<td>CCNOT Chain</td>
<td>Aer Sim</td>
<td>00:45.2</td>
<td>05:07.3</td>
<td>39</td>
<td>384,312</td>
<td>4</td>
<td>49</td>
<td>384,350</td>
</tr>
<tr>
<td>Qiskit</td>
<td>Single-Qubit</td>
<td>Aer Sim</td>
<td>03:54.6</td>
<td>18:38.7</td>
<td>30</td>
<td>1,940,132</td>
<td>4</td>
<td>49</td>
<td>1,940,173</td>
</tr>
<tr>
<td>Qiskit</td>
<td>CCNOT Chain</td>
<td>Rochester</td>
<td>00:53.9</td>
<td>4:34:26.7</td>
<td>53</td>
<td>1,027,772</td>
<td>20</td>
<td>1108</td>
<td>1,028,758</td>
</tr>
<tr>
<td>Qiskit</td>
<td>Single-Qubit</td>
<td>Rochester</td>
<td>DNF&lt;sup&gt;16&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qiskit</td>
<td>CCNOT Chain</td>
<td>AQT Innsbruck</td>
<td>00:43.8</td>
<td>30:20.3</td>
<td>39</td>
<td>761,606</td>
<td>34</td>
<td>506</td>
<td>762,002</td>
</tr>
<tr>
<td>Qiskit</td>
<td>Single-Qubit</td>
<td>AQT Innsbruck</td>
<td>02:53.0</td>
<td>2:50:15.7</td>
<td>30</td>
<td>9,524,360</td>
<td>34</td>
<td>506</td>
<td>9,524,819</td>
</tr>
</tbody>
</table>

Table 5. Resource estimates for dataset 3 (two 4096-element sequences).
1. Understanding the quantum paradigm is difficult for conventional engineers

2. Deriving code from theory tends to require significant experience
# Growing the Workforce

<table>
<thead>
<tr>
<th>Background Math</th>
<th>Classical Computing</th>
<th>Qubits and Quantum Gates</th>
<th>Multi-Qubit Systems</th>
<th>Quantum Circuits</th>
<th>Quantum Protocols</th>
<th>Quantum Algorithms</th>
<th>Quantum Error Correction</th>
<th>Execution on Quantum Computers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex Numbers</td>
<td>Digital Information</td>
<td>Qubits</td>
<td>Qubit Registers</td>
<td>Complex Superpositions</td>
<td>Quantum Interference</td>
<td>Deutsch-Jozsa Algorithm</td>
<td>Bit-Flip Error Correction</td>
<td>Intro to Qiskit</td>
</tr>
<tr>
<td>Vectors</td>
<td>Endianness</td>
<td>The Bloch Sphere</td>
<td>Multi-Qubit Gates</td>
<td>Quantum Circuit Diagrams</td>
<td>Superdense Coding</td>
<td>Simon's Algorithm</td>
<td>Steane ECC</td>
<td>Cloud-Based Machines</td>
</tr>
<tr>
<td>Matrices</td>
<td>Digital Logic</td>
<td>Single-Qubit Gates</td>
<td>The Quirk Tool</td>
<td>Grover's Algorithm</td>
<td>Quantum Fourier Transform</td>
<td>Closing Thoughts and Next Steps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bra-ket and Tensor Notation</td>
<td>Low-level Programming</td>
<td>Intro to Q#</td>
<td>Quantum Fourier Transform</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High-level Programming</td>
<td>Lab Tutorial: Single-Qubit Gates</td>
<td>Shor's Algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual Studio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Growing the Workforce

Qubits

Complex Number Reminder

In the refresher section, we briefly went over complex numbers. As a reminder, a complex number in the form $a + bi$ represents a point on the complex plane, where $a$ is the real part and $b$ is the imaginary part.

```csharp
/// # Summary
/// In this exercise, you are given a single qubit which is in a |0> state. Your objective is to flip the qubit. Use the single quantum gates that Q# provides to transform it into the |1> state.

/// # Input
/// ## target
/// The qubit you need to flip. It will be in the |0> state initially.

/// # Remarks
/// This will show you how to apply quantum gates to qubits in Q#. 

operation Exercise1 (target: Qubit) : Unit {
    // TODO
    fail "Not implemented."
}
```
“The way all of [the instructors and TAs] interacted with the class made it feel like I was a part of the group and not being taught down to…

[I] now hope my future career lies in quantum computing!!”
Bridging the Gap

\[ \lvert \psi \rangle = \lvert 1 \rangle \otimes \sum \sum \frac{1}{\sqrt{n}} (\lvert ij \rangle \otimes \lvert i \cdot j \rangle) \]

Authors? Engineers?

Software Engineers

Compilers

- X 0
- H 1
- H 2
- H 3
- CNOT 1 3
- T_ADJ 3
- CNOT 1 2
- ...

RZ(pi/2) 2
RX(pi/2) 2
RZ(2.510564325593089) 2
RX(pi/2) 2
RZ(-3*pi/4) 3
RX(pi/2) 3
RZ(0.9553166181245095) 3
- ...

© 2021 THE MITRE CORPORATION. ALL RIGHTS RESERVED. APPROVED FOR PUBLIC RELEASE.
Lessons Learned

1. Use the **right tool** for the job.

2. Understand **why** a particular algorithm is **useful** in the first place.

3. Look for **assumptions**, **omissions**, and **abstractions** that prevent you from building and testing new algorithms.

4. Connect with **experts** that can provide access to hardware.

5. Help your colleagues when they get stuck.
The Future

Azure Quantum
Experience quantum impact today on Azure

Start free  Login to Azure Quantum

Google Quantum AI  Software  Cirq  Tutorials

Get started with Quantum Computing Service

Run in Google Colab  View source on GitHub  Download notebook

Amazon Braket
Get started with quantum computing

Sign in to Quantum Cloud Services

Email address

Password

SIGN IN

Forgot your password?

© 2021 THE MITRE CORPORATION. ALL RIGHTS RESERVED. APPROVED FOR PUBLIC RELEASE.
The Future

qRAM Library for Q#

This library implements a variety of different proposals for memory for quantum computers, also commonly called qRAM.

Want to learn more about what qRAM is? Check out the primer on memory for quantum computers in our docs!

Source: https://github.com/qsharp-community/qram

Quantum Algorithm Zoo

This is a comprehensive catalog of quantum algorithms. If you notice any errors or omissions, please email me at stephen.jordan@microsoft.com. (Alternatively, you may submit a pull request to the repository on github.) Your help is appreciated and will be acknowledged.

Source: https://quantumalgorithmzoo.org/
Joe Clapis
jclapis@mitre.org
https://github.com/jclapis/