A Verified Optimizer for Quantum Circuits

Kesha Hietala
University of Maryland
kesha@cs.umd.edu

Robert Rand
University of Chicago
rand@uchicago.edu

Shih-Han Hung
University of Maryland
shung@umd.edu

Xiaodi Wu
University of Maryland
xwu@cs.umd.edu

Michael Hicks
University of Maryland
mwh@cs.umd.edu

POPL 2021
Writing Quantum Programs is Hard

- Quantum indeterminacy $\Rightarrow$ quantum programs are **probabilistic**

- Quantum programs are often written as **circuits**

- Quantum programs use **new primitives**
  - E.g. “prepare a uniform superposition”, “perform a Fourier transform”

Image from https://en.wikipedia.org/wiki/Quantum_phase_estimation_algorithm
Quantum Machines are Limited

• Machines today have a **few, unreliable qubits**
  ▸ Typically 15-50 qubits in total
  ▸ In the near future, we can expect machines with a few hundred qubits, able to run up to 1000 two-qubit gates

• They also have **hardware-specific constraints**
  ▸ Limited set of available operations
  ▸ Only allow two-qubit gates between certain pairs of qubits

*Noisy, Intermediate Scale Quantum (NISQ) Computing* - Preskill

Image from [https://github.com/Qiskit/ibmq-device-information](https://github.com/Qiskit/ibmq-device-information)
Quantum Compilers are Complicated

• Quantum compilers need to perform **sophisticated transformations** to account for limited resources, hardware constraints

• These transformation are hard to write… and harder to debug
  ‣ Is an unexpected result due to a program bug? machine error? quantum indeterminacy?
Verified Compiler Stack

• End goal: *verified compiler stack* for quantum programs

![Diagram showing the verified compiler stack process]

- **High-level Language**
  - E.g. Quipper, Q#

- **Unoptimized IR**
  - E.g. OpenQASM, Quil

- **Optimized IR**

- **Hardware Instructions**

---

- **Challenge:** The semantics of quantum programs is very different from classical programs
  - States represented as matrices of complex numbers
  - Programs involve probabilities, trigonometry

- **Requires development of new frameworks, libraries, and automation**
SQIR and VOQC

• Our paper: **VOQC**, a *Verified Optimizer for Quantum Circuits*, which is built on top of **SQIR**, a *Small Quantum Intermediate Representation* designed for proof.
SQIR and VOQC

• SQIR and VOQC are implemented in around 11k lines of Coq code
  ▪ 3.5k for core SQIR, source program proofs
  ▪ 7.5k for VOQC libraries, optimizations, circuit mapper
  ▪ We extend QWIRE’s matrix & complex number libraries by 3k lines

• Long version of the paper available at https://arxiv.org/abs/1912.02250

• Code available at https://github.com/inQWIRE/SQIR

• Artifact available at https://zenodo.org/record/4268896
Outline

• Intro to Quantum Programming
• SQIR
• VOQC
• Future Work
Qubits

\[
\begin{align*}
|0\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\
|1\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\end{align*}
\]

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

*Superposition*: Qubits can be in multiple states (0 or 1) at once.
Measurement: Looking at a qubit probabilistically turns it into a bit.
Measurement: Looking at a qubit probabilistically turns it into a bit.
Operators

A unitary operator transforms, or *evolves*, a state

\[ H \left| 0 \right> = \left| + \right> \]

\[ H \left| + \right> = \left| 0 \right> \]

This is the *Hadamard* operator, H

(which is its own inverse)
Operators

Operators are represented as unitary matrices

\[
\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}
\]

\[
\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}
\]
Multiple Qubits

\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{2}}
\end{pmatrix}
\otimes
\begin{pmatrix}
1 \\
0
\end{pmatrix}
= 
\begin{pmatrix}
\frac{1}{\sqrt{2}} \\
0 \\
\frac{1}{\sqrt{2}} \\
0
\end{pmatrix}
\]

\[|+\rangle \otimes |0\rangle = |+\rangle |0\rangle \quad \text{or} \quad |+ 0\rangle\]

*Multi-qubit states* are constructed via the tensor product
Measurement 2.0

\[ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} \rightarrow \frac{1}{2} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \]
Measurement 2.0

\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} \\
0 \\
0 \\
\frac{1}{\sqrt{2}}
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 \\
0
\end{pmatrix} \otimes \begin{pmatrix}
1 \\
0
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 \\
1
\end{pmatrix} \otimes \begin{pmatrix}
0 \\
1
\end{pmatrix}
\]

\[
|00\rangle \\
|11\rangle
\]
Entangled qubits are not probabilistically independent — they cannot be decomposed. Connection at a distance!
Multi-Qubit Unitaries

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
\frac{1}{\sqrt{2}} \\
0 \\
\frac{1}{\sqrt{2}} \\
0
\end{pmatrix}
= 
\begin{pmatrix}
\frac{1}{\sqrt{2}} \\
0 \\
0 \\
\frac{1}{\sqrt{2}}
\end{pmatrix}
\]

CNOT $|+0\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle)$
A universal set of unitaries can be used to approximate any unitary operator using a finite sequence of gates.

\[
X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{bit flip} \\
R_{z}(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \quad \text{phase shift}
\]

\[|0\rangle \leftrightarrow |1\rangle \]
\[|1\rangle \leftrightarrow |0\rangle \]
\[|0\rangle \leftrightarrow |0\rangle \]
\[|1\rangle \leftrightarrow e^{i\theta} |1\rangle \]
General Quantum States

• So far we have seen pure states
  - E.g. $|0\rangle$, $|1\rangle$, $|+\rangle$

• A mixed state is a (classical) probability distribution over pure states
  - E.g.\(\begin{cases} |0\rangle \text{ with probability } 1/2 \\ |1\rangle \text{ with probability } 1/2 \end{cases}\)

• Density matrices allow us to describe both pure and mixed states

\[
\rho = |0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \rho = \frac{1}{2} |0\rangle\langle 0| + \frac{1}{2} |1\rangle\langle 1| = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}
\]
Circuits

Quantum programs are often written as circuits

\[ \begin{array}{c}
|0\rangle & \quad H & \quad |0\rangle \\
|0\rangle & \quad |0\rangle & \quad |0\rangle \\
|0\rangle & \quad |0\rangle & \quad |0\rangle \\
\end{array} \]

three qubits

three gates
Circuits

Quantum programs are often written as circuits

\[ |0\rangle \]

\[ H \]

\[ |0\rangle \]

\[ |0\rangle \]

\[ |0\rangle \]
Circuits

Quantum programs are often written as circuits

![Circuit Diagram](image)
Quantum Programming

Many “high-level” quantum programming languages (e.g. PyQuil, Cirq, Qiskit, Quipper, QWIRE) are libraries for constructing circuits.
SQIR: Small Quantum Intermediate Representation

- SQIR programs, embedded in Coq, are assigned a \textit{denotational semantics} of matrices.
- Two variations of SQIR:
  - \textit{Unitary SQIR}: No measurement.
  - \textit{Full SQIR}: Adds branching measurement operator.

High-level Language
E.g. Quipper, Q#

Unoptimized OpenQASM

Optimized OpenQASM

Hardware Instructions

Unoptimized SQIR

Optimized SQIR

VOQC
Unitary SQIR

- Semantics parameterized by gate set $G$ and dimension $d$ of a global register

$$U ::= U_1; U_2 \mid G/q \mid G/q_1 q_2$$

- The denotation (semantics) of $U$ is a $2^d \times 2^d$ unitary matrix

$$[[U_1; U_2]]_d = [[U_2]]_d \times [[U_1]]_d$$

$$[[G_1 \ q]]_d = \begin{cases} \text{apply}_1(G_1, q, d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

$$[[G_2 \ q_1 \ q_2]]_d = \begin{cases} \text{apply}_2(G_2, q_1, q_2, d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

E.g. $\text{apply}_1(X, q, d) = I_{2q} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes I_{2(d-q-1)}$

$q < d$

$q_1 < d \land q_2 < d \land q_1 \neq q_2$
Non-Unitary SQIR

• Semantics parameterized by gate set $G$ and dimension $d$ of a global register

$$P ::= \text{skip} \mid P_1; P_2 \mid U \mid \text{meas } q \ P_1 \ P_2$$

• The denotation of $P$ is a function over $2^d \times 2^d$ density matrices

\[
\begin{align*}
\{\text{skip}\}_d(\rho) &= \rho \\
\{P_1; P_2\}_d(\rho) &= (\{P_2\}_d \circ \{P_1\}_d)(\rho) \\
\{U\}_d(\rho) &= [U]_d \times \rho \times [U]^\dagger_d \\
\{\text{meas } q \ P_1 \ P_2\}_d(\rho) &= \{P_2\}_d(|0\rangle_q \langle 0| \times \rho \times |0\rangle_q \langle 0|) \\
&\quad + \{P_1\}_d(|1\rangle_q \langle 1| \times \rho \times |1\rangle_q \langle 1|)
\end{align*}
\]

Standard semantics; also used in QHL\(^1\) and QWIRE\(^2\)

\(^1\) Ying. Floyd-Hoare logic for quantum programs. TOPLAS 2012.
SQIR Metaprogramming

- SQIR programs just express circuits. We can express parameterized circuit families using Coq as a meta programming language.

\[
\begin{align*}
|0\rangle & \xrightarrow{H} |0\rangle \\
|0\rangle & \xrightarrow{\text{CNOT}} |0\rangle \\
\vdots & \\
|0\rangle & \xrightarrow{\text{CNOT}} |0\rangle \\
|0\rangle & \xrightarrow{\text{CNOT}} |0\rangle \\
|0\rangle & \xrightarrow{\text{CNOT}} |0\rangle \\
\end{align*}
\]

\[
\text{Fixpoint } \text{ghz} \ (n : \mathbb{N}) : \text{ucom base } n := \\
\begin{align*}
\text{match } n \text{ with} \\
| 0 & \Rightarrow \text{SKIP} \\
| 1 & \Rightarrow \text{H } 0 \\
| S \ n' & \Rightarrow \text{ghz } n'; \text{CNOT } (n'-1) \ n' \\
\text{end.}
\end{align*}
\]

- The \text{ghz} Coq function returns a SQIR program (of type \text{ucom base } n) whose semantics is the \(n\)-qubit GHZ state.
Proofs of Correctness in Coq

- We might like to prove that evaluating $\text{ghz } n$ on $|0angle^\otimes n$ produces $|\text{GHZ}^n\rangle$
  - where $|\text{GHZ}^n\rangle = \frac{1}{\sqrt{2}}(|0angle^\otimes n + |1\rangle^\otimes n)$

```coq
Definition GHZ (n : N) : Vector (2 ^ n) :=
  match n with
  | 0 => I 1
  | S n' => \frac{1}{\sqrt{2}} * |0\rangle^\otimes n + \frac{1}{\sqrt{2}} * |1\rangle^\otimes n
  end.

Lemma ghz_correct : \forall n : N, 
  n > 0 -> [\text{ghz } n]_n \times |0\rangle^\otimes n = \text{GHZ } n.
Proof.
  ...
Qed.
```
Designed for Proof

• SQIR was conceived as a simplified version of QWIRE\(^1\); we use QWIRE’s libraries for matrices and complex numbers

• SQIR proofs are simpler that QWIRE’s because we:
  1. Reference **qubits using concrete indices** (\texttt{CNOT 2 1} vs. \texttt{CNOT x y})
     - Easy to map gate arguments to the right column/row in the matrix
     - Disjointness is **syntactic**; important for proving equivalences
  2. **Separate the unitary core from the full language** with measurement
     - Unitary matrix semantics simpler than **density matrix** formulation
  3. Assign a **denotation of the zero-matrix to ill-typed programs**
     - E.g., \texttt{CNOT 1 1}, which violates no-cloning

Proofs so Far

- We have formally verified several source programs correct
  - Quantum teleportation / superdense coding
  - GHZ state preparation
  - Deutsch-Jozsa algorithm
  - Simon’s algorithm
  - Grover’s search algorithm
  - Quantum phase estimation (key part of Shor’s algorithm)

- These proofs as well as the basic support of SQIR (lemmas, tactics, etc.) constitute about 3500 lines of Coq code

- For more details see https://arxiv.org/abs/2010.01240
VOQC: A Verified Optimizer for Quantum Circuits

- Transformations are represented as Coq functions over SQIR circuits
  - Extracted to executable OCaml code

- We prove (verify) that transformations are semantics-preserving
  - Can also prove that the output program respects machine constraints
Most of VOQC (2200 LOC) consists of verified implementations of optimizations developed by Nam et al.¹

- Replacement (peephole optimizations)
- Propagation (commutation) and cancellation
- Rotation merging (non-local coalescing)

Some optimizations for non-unitary programs, inspired by Qiskit (800 LOC)

- Remove $z$-rotations before measurement
- Classical state propagation

Another 2100 LOC for program manipulation; 2100 more for circuit mapping

¹Nam, Ross, Su, Childs and Maslov. *Automated Optimization of Large Quantum Circuits with Continuous Parameters*. npj 2018.
Example: X Propagation

- Based on Nam et al\textsuperscript{1} “not propagation”

- We verify **semantics-preservation**
  - At each step, the denotation of the program (i.e. unitary matrix) does not change

- We prove this via induction on the structure of the input program
  - \(\sim\)30 lines to implement optimization
  - \(\sim\)270 lines to prove semantics-preservation

\textsuperscript{1}Nam, Ross, Su, Childs and Maslov. *Automated Optimization of Large Quantum Circuits with Continuous Parameters*. npj 2018.
Verifying Matrix Equivalences

• Many proofs use unitary equivalences; e.g., X propagation’s proof uses:
  ▶ X gates cancel: \( X \ m; X \ m \equiv I \ m \)
  ▶ X commutes with CNOT control: \( X \ m; CNOT \ m \ n \equiv CNOT \ m \ n; X \ m; X \ n \)
  ▶ X commutes with CNOT target: \( X \ n; CNOT \ m \ n \equiv CNOT \ m \ n; X \ n \)
  ▶ H transforms X to Z: \( X \ m; H \ m \equiv H \ m; Z \ m \)

• We prove these as lemmas
  ▶ Doing so is tedious, so we developed Coq tactics to convert matrix expressions into a grid normal form to facilitate automation
Grid Normal Form

- Consider the equivalence $X n; \text{CNOT} \ m \ n \equiv \text{CNOT} \ m \ n; X n$

- Per our semantics, this requires proving

$$apply_1(X, n, d) \times apply_2(\text{CNOT}, m, n, d) = apply_2(\text{CNOT}, m, n, d) \times apply_1(X, n, d)$$

  - where

    $$apply_1(X, n, d) = I_{2^n} \otimes \sigma_x \otimes I_{2^q}$$

    $$apply_2(\text{CNOT}, m, n, d) = I_{2^m} \otimes |1\rangle\langle 1| \otimes I_{2^p} \otimes \sigma_x \otimes I_{2^q} + I_{2^m} \otimes |0\rangle\langle 0| \otimes I_{2^p} \otimes I_2 \otimes I_{2^q}$$

- Our automation reduces both sides of the equality to grid normal form

  $$I_{2^m} \otimes |1\rangle\langle 1| \otimes I_{2^p} \otimes I_2 \otimes I_{2^q} + I_{2^m} \otimes |0\rangle\langle 0| \otimes I_{2^p} \otimes \sigma_x \otimes I_{2^q}.$$
More Interesting: Rotation Merging

• Based on Nam et al rotation merging

• Combines $Rz$ gates in arbitrary \{$Rz, CNOT$\} sub-circuits
  ▶ ~100 lines to implement optimization
  ▶ ~920 lines to prove semantics-preservation

Input state is $|abc\rangle$ for $a, b, c \in \{0,1\}$
Also: Circuit Mapping

• Given an input program & description of machine connectivity, circuit mapping produces a program that satisfies connectivity constraints

  ‣ Usually uses SWAP gates to “move” qubits by exchanging their values

    ![Diagram]

    $|\psi\rangle$  
    $|\phi\rangle$

  ‣ E.g  CNOT 0 2 ,  

    ![Diagram]

    $\rightarrow$ SWAP 0 1; CNOT 1 2

• We prove that the output program is equivalent to the original, up to permutation of indices

  ‣ Above,  $[[\text{CNOT 0 2}]]_3 = P \times [[\text{SWAP 0 1; CNOT 1 2}]]_3$  where $P$ implements the permutation  $\{0 \rightarrow 1, \ 1 \rightarrow 0, \ 2 \rightarrow 2\}$
Evaluation

• Is VOQC any good? Maybe we just verified simple optimizations

• So: Compared our verified optimizer against existing unverified optimizers
  ▶ IBM Qiskit Terra v0.15.12¹
  ▶ Cambridge CQC tket v0.6.0²
  ▶ Nam et al,³ both L and H levels (used by IonQ)
  ▶ Amy et al⁴
  ▶ PyZX v0.6.0⁵

¹ https://qiskit.org/
⁵ https://github.com/Quantomatic/pyzx
Benchmark

- Used benchmark suite of Amy et al\textsuperscript{1}
  - 28 programs: Arithmetic circuits, implementations of multiple-control Toffoli gates, and Galois field multiplier circuits
  - Ranging from 45 to 13,593 gates and 5 to 96 qubits
  - Uses the Clifford+T gate set (CNOT, H, S and T)

- We measured \textbf{effectiveness in terms of gate reductions}
  - Both T gate and total

- Measured optimization time (not parsing or printing)

\textsuperscript{1}Amy, Maslov and Mosca. \textit{Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning}. TCAD 2013.
Results

<table>
<thead>
<tr>
<th>Geo. mean compilation times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qiskit$^1$</td>
</tr>
<tr>
<td>0.812s</td>
</tr>
</tbody>
</table>

VOQC is the same ballpark

<table>
<thead>
<tr>
<th>Geo. mean reduction in gate count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qiskit</td>
</tr>
<tr>
<td>10.1%</td>
</tr>
</tbody>
</table>

VOQC only outperformed by Nam

<table>
<thead>
<tr>
<th>Geo mean. reduction in T gate count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
</tr>
<tr>
<td>39.7%</td>
</tr>
</tbody>
</table>

VOQC only outperformed by PyZX

---

1 https://qiskit.org/
5 https://github.com/Quantomatic/pyzx
No Bugs!

---


---

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>Qiskit</th>
<th>t(cet)</th>
<th>Nam (L)</th>
<th>Nam (H)</th>
<th>VOQC</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder_8</td>
<td>900</td>
<td>805</td>
<td>775</td>
<td>646</td>
<td>606</td>
<td>682</td>
</tr>
<tr>
<td>barenco_tof_3</td>
<td>58</td>
<td>51</td>
<td>51</td>
<td>42</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>barenco_tof_4</td>
<td>114</td>
<td>100</td>
<td>100</td>
<td>78</td>
<td>72</td>
<td>95</td>
</tr>
<tr>
<td>barenco_tof_5</td>
<td>170</td>
<td>149</td>
<td>149</td>
<td>114</td>
<td>104</td>
<td>140</td>
</tr>
<tr>
<td>barenco_tof_10</td>
<td>450</td>
<td>394</td>
<td>394</td>
<td>294</td>
<td>264</td>
<td>365</td>
</tr>
<tr>
<td>csla_mux_3</td>
<td>170</td>
<td>156</td>
<td>156</td>
<td>153</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>csun_mux_9</td>
<td>420</td>
<td>382</td>
<td>361</td>
<td>294</td>
<td>266</td>
<td>308</td>
</tr>
<tr>
<td>gf2^4_mult</td>
<td>225</td>
<td>206</td>
<td>206</td>
<td>187</td>
<td>187</td>
<td>192</td>
</tr>
<tr>
<td>gf2^5_mult</td>
<td>347</td>
<td>319</td>
<td>319</td>
<td>296</td>
<td>296</td>
<td>291</td>
</tr>
<tr>
<td>gf2^6_mult</td>
<td>495</td>
<td>454</td>
<td>454</td>
<td>403</td>
<td>403</td>
<td>410</td>
</tr>
<tr>
<td>gf2^7_mult</td>
<td>669</td>
<td>614</td>
<td>614</td>
<td>555</td>
<td>555</td>
<td>549</td>
</tr>
<tr>
<td>gf2^8_mult</td>
<td>883</td>
<td>804</td>
<td>806</td>
<td>712</td>
<td>712</td>
<td>705</td>
</tr>
<tr>
<td>gf2^9_mult</td>
<td>1095</td>
<td>1006</td>
<td>1009</td>
<td>891</td>
<td>891</td>
<td>885</td>
</tr>
<tr>
<td>gf2^10_mult</td>
<td>1347</td>
<td>1238</td>
<td>1240</td>
<td>1070</td>
<td>1070</td>
<td>1084</td>
</tr>
<tr>
<td>gf2^16_mult</td>
<td>3435</td>
<td>3148</td>
<td>3150</td>
<td>2707</td>
<td>2707</td>
<td>2695</td>
</tr>
<tr>
<td>gf2^32_mult</td>
<td>13593</td>
<td>12506</td>
<td>12507</td>
<td>10601</td>
<td>10601</td>
<td>10577</td>
</tr>
<tr>
<td>mod5_4</td>
<td>63</td>
<td>58</td>
<td>58</td>
<td>51</td>
<td>51</td>
<td>56</td>
</tr>
<tr>
<td>mod_mod_55</td>
<td>119</td>
<td>106</td>
<td>102</td>
<td>91</td>
<td>91</td>
<td>90</td>
</tr>
<tr>
<td>mod_red_21</td>
<td>278</td>
<td>227</td>
<td>224</td>
<td>184</td>
<td>180</td>
<td>214</td>
</tr>
<tr>
<td>qccla_adder_10</td>
<td>521</td>
<td>469</td>
<td>460</td>
<td>411</td>
<td>399</td>
<td>338</td>
</tr>
<tr>
<td>qccla_com_7</td>
<td>443</td>
<td>398</td>
<td>392</td>
<td>284</td>
<td>284</td>
<td>314</td>
</tr>
<tr>
<td>qccla_mod_7</td>
<td>884</td>
<td>793</td>
<td>780</td>
<td>636</td>
<td>624</td>
<td>723</td>
</tr>
<tr>
<td>rc_adder_6</td>
<td>200</td>
<td>170</td>
<td>172</td>
<td>142</td>
<td>140</td>
<td>157</td>
</tr>
<tr>
<td>tof_3</td>
<td>45</td>
<td>40</td>
<td>40</td>
<td>35</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>tof_4</td>
<td>75</td>
<td>66</td>
<td>66</td>
<td>55</td>
<td>55</td>
<td>65</td>
</tr>
<tr>
<td>tof_5</td>
<td>105</td>
<td>92</td>
<td>92</td>
<td>75</td>
<td>75</td>
<td>90</td>
</tr>
<tr>
<td>tof_10</td>
<td>253</td>
<td>222</td>
<td>222</td>
<td>175</td>
<td>175</td>
<td>215</td>
</tr>
<tr>
<td>vbe_adder_3</td>
<td>150</td>
<td>138</td>
<td>139</td>
<td>89</td>
<td>89</td>
<td>101</td>
</tr>
</tbody>
</table>

Geo. Mean Reduction = 10.1% 10.6% 23.3% 24.8% 17.8%

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>Amy</th>
<th>PyZX</th>
<th>Nam (L)</th>
<th>Nam (H)</th>
<th>VOQC</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder_8</td>
<td>399</td>
<td>215</td>
<td>173</td>
<td>215</td>
<td>215</td>
<td>215</td>
</tr>
<tr>
<td>barenco_tof_3</td>
<td>28</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>barenco_tof_4</td>
<td>56</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>barenco_tof_5</td>
<td>84</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>barenco_tof_10</td>
<td>224</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>csla_mux_3</td>
<td>70</td>
<td>62</td>
<td>62</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>csun_mux_9</td>
<td>196</td>
<td>112</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>gf2^4_mult</td>
<td>112</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>gf2^5_mult</td>
<td>175</td>
<td>111</td>
<td>115</td>
<td>115</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>gf2^6_mult</td>
<td>252</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>gf2^7_mult</td>
<td>343</td>
<td>217</td>
<td>217</td>
<td>217</td>
<td>217</td>
<td>217</td>
</tr>
<tr>
<td>gf2^8_mult</td>
<td>448</td>
<td>264</td>
<td>264</td>
<td>264</td>
<td>264</td>
<td>264</td>
</tr>
<tr>
<td>gf2^9_mult</td>
<td>567</td>
<td>351</td>
<td>351</td>
<td>351</td>
<td>351</td>
<td>351</td>
</tr>
<tr>
<td>gf2^10_mult</td>
<td>700</td>
<td>410</td>
<td>410</td>
<td>410</td>
<td>410</td>
<td>410</td>
</tr>
<tr>
<td>gf2^16_mult</td>
<td>1792</td>
<td>1040</td>
<td>1040</td>
<td>1040</td>
<td>1040</td>
<td>1040</td>
</tr>
<tr>
<td>gf2^32_mult</td>
<td>7168</td>
<td>4128</td>
<td>4128</td>
<td>4128</td>
<td>4128</td>
<td>4128</td>
</tr>
<tr>
<td>mod5_4</td>
<td>28</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>mod_mod_55</td>
<td>49</td>
<td>37</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>mod_red_21</td>
<td>119</td>
<td>73</td>
<td>73</td>
<td>73</td>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>qccla_adder_10</td>
<td>238</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>162</td>
</tr>
<tr>
<td>qccla_com_7</td>
<td>203</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>qccla_mod_7</td>
<td>413</td>
<td>249</td>
<td>237</td>
<td>237</td>
<td>235</td>
<td>249</td>
</tr>
<tr>
<td>rc_adder_6</td>
<td>77</td>
<td>63</td>
<td>47</td>
<td>47</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>tof_3</td>
<td>21</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>tof_4</td>
<td>35</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>tof_5</td>
<td>49</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>tof_10</td>
<td>119</td>
<td>71</td>
<td>71</td>
<td>71</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>vbe_adder_3</td>
<td>70</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

Geo. Mean Reduction = 39.7% 42.6% 41.4% 41.4% 41.4% 41.4%
Summary and Future Work

- **SQIR** and **VOQC**: Two building blocks of a verified quantum software stack
  - Powerful enough to verify state-of-the-art optimizations, and prove source programs correct (QPE; Grover’s)
  - Resulted in novel frameworks, libraries, automation for quantum program proofs

Ongoing work

1. New optimizations (e.g. inspired by Qiskit)
   - **Unoptimized SQIR**
   - **Optimized SQIR**
   - **VOQC**

2. New proofs (full Shor's)
   - **Unoptimized OpenQASM**
   - **Optimized OpenQASM**

3. Other verified elements of the stack (some of which involve challenging designs)
   - High-level Language
     - E.g. Quipper, Q#
   - **Unoptimized OpenQASM**
   - **Optimized OpenQASM**
   - **Hardware Instructions**

[github.com/inQWIRE/SQIR](https://github.com/inQWIRE/SQIR)