A Verified Optimizer for Quantum Circuits

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks

University of Maryland & University of Chicago

University of Maryland DEPARTMENT OF COMPUTER SCIENCE

THE UNIVERSITY OF CHICAGO DEPARTMENT OF COMPUTER SCIENCE

Abstract

We present VOQC (pronounced "vox"), a fully verified compiler for quantum circuits. Circuits are written in a simple quantum language we call SQIR (pronounced "squire"), which is deeply embedded in the Coq proof assistant. This allows us to formally verify properties of SQIR programs and program transformations. We demonstrate the power of SQIR by proving the correctness of various protocols, and by verifying optimizations used in a state-of-the-art compiler.

Our verified compiler performs comparably to industrial-strength compilers. VOQC's optimization reduces the gate counts by 17.7% on a benchmark of 29 programs compared to a 10.7% reduction when using IBM's Qiskit compiler.

The full paper [1] and code [2] are available online.

Formal Verification

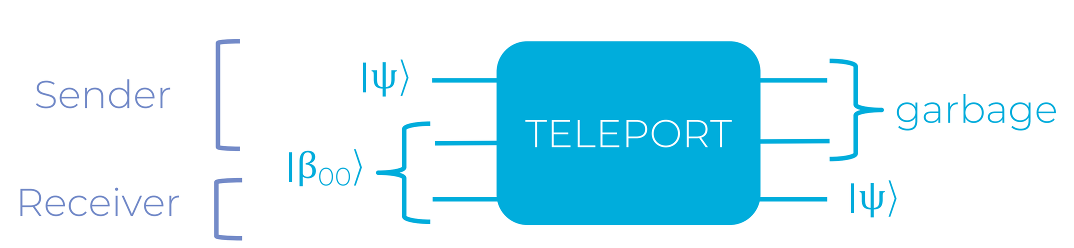
The process of mathematically proving the correctness of a piece of software is known as *formal verification*. Formal verification is particularly useful in the field of quantum computing, where standard software assurance techniques such as unit testing and runtime debugging are infeasible.

Examples of formal verification that have been applied in the field of quantum computing include:

- Model checking
- Equivalence checking
- Program logics
- Direct proofs about semantics
- Diagrammatic reasoning

For the most part, these techniques are used to prove that a quantum program satisfies some specification. For example, given a program describing the quantum teleportation protocol, the goal may be to prove that the program correctly "teleports" any input qubit to the receiver.

Example. Prove that the following holds for any state $|\psi\rangle$.



Another useful application of formal verification, which has received relatively little attention from the quantum formal verification community, is proving the correctness of program *transformations*. Verifying program transformations allows the construction of *certified compilers*, which guarantee that the executable code they output behaves as specified by the input source program. A notable example of a certified compiler (for classical programs) is CompCert [3], an optimizing compiler for C proved correct using the Coq proof assistant [4].

SQIR: A Small Quantum IR

SQIR supports four quantum programming constructs: skip, sequencing, unitary application, and branching measurement (i.e., measurement on the qubit q and either performing P1 or P2 depending on the result). Unitary SQIR programs allow sequencing and unitary gate application to one or two qubits.

$$P \;:=\; \texttt{skip} \mid P_1;\, P_2 \mid U \mid \texttt{meas}\; q\; P_1\; P_2$$
$$U \;:=\; U_1;\, U_2 \mid G\; q \mid G\; q_1\; q_2$$

Other constructs can be defined in terms of these. For example, measurement and reset can be defined in terms of branching measurement.

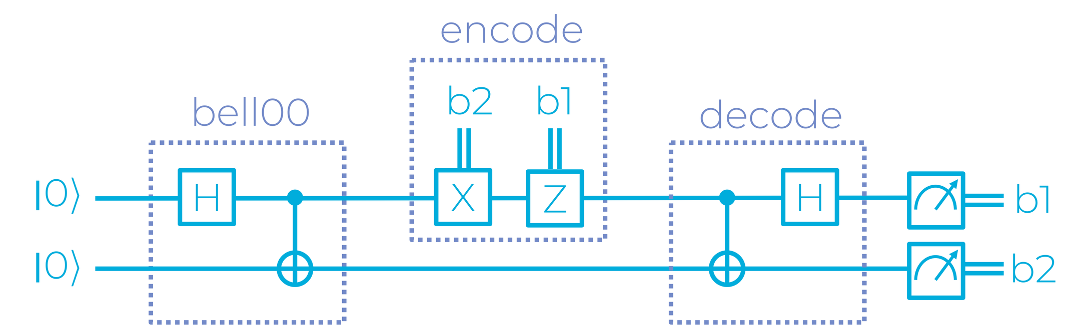
$$\texttt{measure}\; q = \texttt{meas}\; q\; \texttt{skip}\; \texttt{skip} \qquad \texttt{reset}\; q = \texttt{meas}\; q\; (X\; q)\; \texttt{skip}$$

SQIR is parameterized by choice of gate set. New gates can be defined in terms of built-in gates. For example, we define the swap operation as follows.

```
Definition SWAP a b := CNOT a b; CNOT b a; CNOT a b.
```

We can then state and prove properties about the semantics of the defined operations. For example, we can prove that the SWAP program swaps its arguments, as intended.

Example. Superdense coding is a protocol that allows a sender to transmit two classical bits, b1 and b2, to a receiver using a single quantum bit. The circuit and SQIR program for superdense coding are shown below.




```
Definition bell00 := H 0; CNOT 0 1.

Definition encode (b1 b2 : 𝔹) :=
    (if b2 then X 0 else skip);
    (if b1 then Z 0 else skip).

Definition decode := CNOT 0 1; H 0.

Definition superdense (b1 b2 : 𝔹) :=
    bell00; encode b1 b2; decode.
```

Although SQIR was designed to be used as an intermediate representation, we can also prove properties about SQIR programs directly, since these programs and their semantics are embedded in Coq. For example, we can prove that the result of evaluating the program superdense b1 b2 on an input state consisting of two qubits initialized to zero is the state |b1, b2⟩. We write this as follows.

```
Lemma superdense_correct : ∀ (b1 b2 : 𝔹),
    ⟦superdense b1 b2⟧ × |0,0⟩ = |b1,b2⟩.
```

In our development, we also verify the correctness of n-qubit GHZ state preparation, quantum teleportation, and the n-qubit Deutsch-Jozsa algorithm.

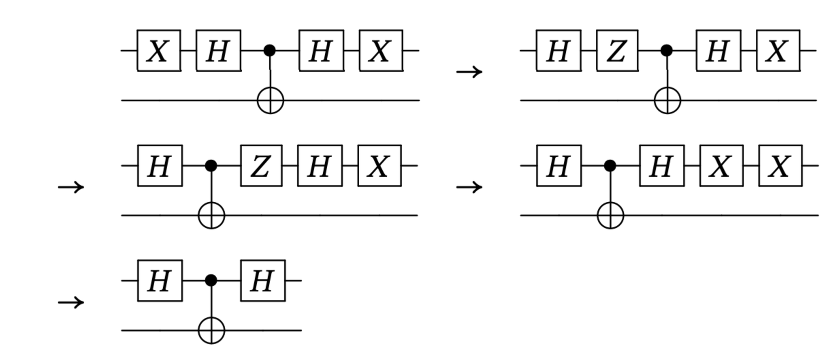
Verified Optimization

Because near-term quantum machines will only be able to perform small computations before decoherence takes effect, compilers for quantum programs must apply sophisticated optimizations to reduce resource usage. These optimizations can be complicated to implement and are vulnerable to programmer error. It is thus important to verify that the implementations of these optimizations are correct.

In general, we will be interested in proving that an optimization is *semantics-preserving*, meaning that it preserves the semantics of a program (its unitary matrix) up to a global phase.

Our optimizations on unitary programs include *X propagation*, *single- and two-qubit gate cancellation*, *Hadamard reduction*, and *rotation merging*, which are all adapted from Nam et al. [5].

Example. Below, *X propagation* is applied to a small example circuit.



To ensure that this optimization is semantics-preserving, we prove the validity of every step of the transformation in Coq. This requires proving circuit equivalences like X q; H q = H q; Z q and verifying that equivalences are applied correctly.

Verified Circuit Mapping

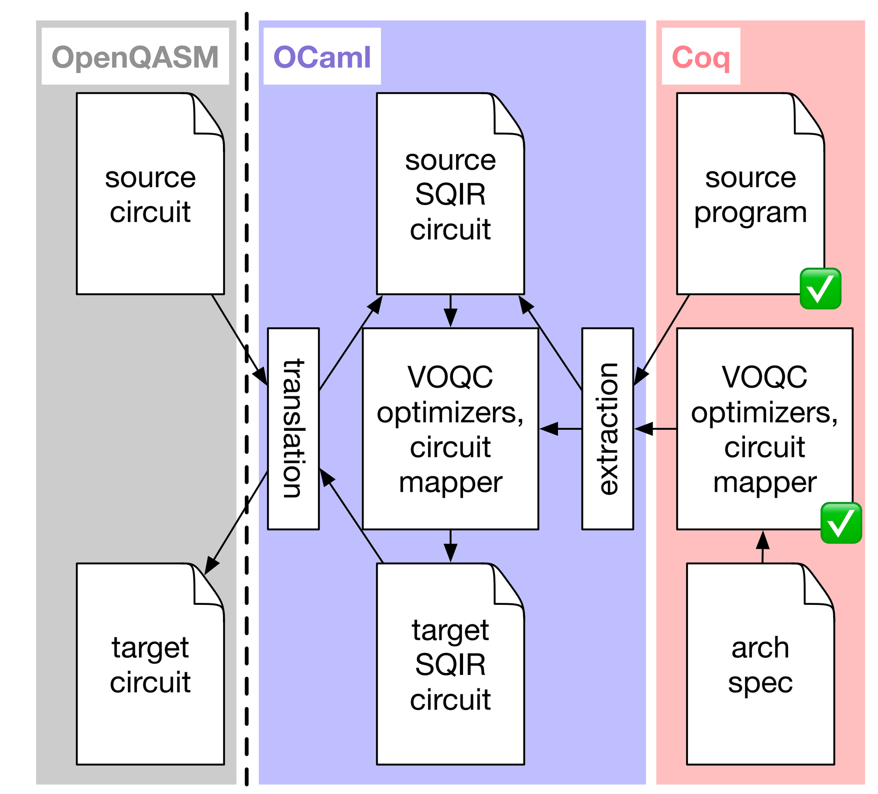
Similarly to how optimization aims to reduce resource usage to make programs more feasible to run on near-term machines, *circuit mapping* aims to address the connectivity constraints of near-term machines. Circuit mapping algorithms take as input an arbitrary program and output a program that respects the connectivity constraints of some underlying architecture.

We have implemented a simple routine that maps a program to an architecture by inserting SWAP operations before and after every CNOT so that the target and control are adjacent when the CNOT is performed and are returned to their original positions before the next operation. We have verified that this routine is semantics-preserving and produces a program satisfying the architecture's connectivity constraints. We provide mapping routines for linear nearest neighbor (LNN), LNN ring, and 2D nearest neighbor architectures as well as IBM's Tenerife machine.

Example. On the following 4-qubit LNN architecture, "CNOT 1 3" becomes "SWAP 1 2; CNOT 2 3; SWAP 1 2".

VOQC

VOQC is the first fully verified circuit optimizer for a realistic quantum circuit language. Our work constitutes a step toward developing a full-scale verified compiler toolchain. The architecture is shown below.



Results. On a benchmark of 29 programs from Amy et al. [6], VOQC reduces gate count by 17.7%, while level 3 optimization of IBM's Qiskit compiler [7] reduces gate count by 10.7%. There is still room for improvement: Nam et al. heavy optimization [5] reduces gate count by 26.5%.

Ongoing work.

- Verified translation to/from OpenQASM [8].
- Additional verified optimizations, taken from other compilers like quilc [9] and tket [10].

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research Quantum Testbed Pathfinder Program under Award Number DE-SC0019040.

References

[1] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. *A Verified Optimizer for Quantum Circuits*. arXiv:1912.02250.

[2] *SQIR Development*. Available at https://github.com/inQWIRE/SQIR.

[3] *The CompCert Verified Compiler*. Available at http://compcert.inria.fr/.

[4] *The Coq Proof Assistant*. Available at https://coq.inria.fr/.

[5] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. *Automated Optimization of Large quantum Circuits with Continuous Parameters.* npj Quantum Information 4(1). 2018.

[6] Matthew Amy, Dmitri Maslov, and Michele Mosca. *Polynomial Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning*. IEEE TCAD 33(10). 2013.

[7] *Qiskit*. Available at https://qiskit.org/.

[8] Kartik Singhal, Robert Rand, and Michael Hicks. *Verified Translation Between Low-Level Quantum Languages*. To appear at PLanQC 2020.

[9] *quilc*. Available at https://github.com/rigetti/quilc/.

[10] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. *Phase Gadget Synthesis for Shallow Circuits*. QPL 2019.