A Verified Optimizer for Quantum Circuits

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks
University of Maryland & University of Chicago

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 $\psi$.

$$\text{verifies: } \text{encode} (b_1 b_2) = b_1 \text{H} b_2$$

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 $\text{superdense} b_1 b_2$ on an input state consisting of two qubits initialized to zero is the state $b_1 b_2$. We write this as follows.

Lemmas:
- $\text{superdense}_{\text{correct}} : \forall b, (b_1 b_2) . \left[ \text{superdense} b_1 b_2 \right] = \left[ b_1 b_2 \right]$.

In our development, we also verify the correctness of $\text{qubit GHZ state preparation, quantum teleportation,}$ 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.

$$\begin{array}{c}
\text{ BASIS GATE}
\end{array}$$

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 $X \otimes I \equiv H \otimes H$ 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 quil [9] and tiqet [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