A Verified Optimizer for Quantum Circuits

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

University of Maryland, College Park, USA
{kesha, rrand, shung, xwu, mwh}@cs.umd.edu
December 9, 2019

Abstract

We present VOQC, the first fully verified compiler for quantum circuits, written using the Coq proof assistant. Quantum circuits are expressed as programs in a simple, low-level language called SQIR, which is deeply embedded in Coq. Optimizations and other transformations are expressed as Coq functions, which are proved correct with respect to a semantics of SQIR programs. We evaluate VOQC's verified optimizations on a series of benchmarks, and it performs comparably to industrial-strength compilers. VOQC's optimizations reduce total gate counts on average by 17.7% on a benchmark of 29 circuit programs compared to a 10.7% reduction when using IBM's Qiskit compiler.

A draft of our full paper is available at `https://arxiv.org/abs/1912.02250`.
Our code is available at `https://github.com/inQWIRE/SQIR`.

Programming quantum computers will be challenging, at least in the near term. Qubits will be scarce, and gate pipelines will need to be short to prevent decoherence. Fortunately, optimizing compilers can transform a source algorithm to work with fewer resources. Where compilers fall short, programmers can optimize their algorithms by hand.

Of course, both compiler and by-hand optimizations will inevitably have bugs. For evidence of the former, consider that higher optimization levels of IBM's Qiskit compiler are known to have mistakes (as is evident from its issue tracker[1]). Kissinger and van de Wetering (2019) discovered mistakes in the optimized outputs produced by the circuit compiler by Nam et al. (2018). And Nam et al. themselves found that the optimization library they compared against sometimes produced incorrect results. Making mistakes when optimizing by hand is also to be expected: as put well by Zamdzhiev (2016), quantum computing can be frustratingly unintuitive.

Unfortunately, the very factors that motivate optimizing quantum compilers make it difficult to test their correctness. Comparing runs of a source program to those of its optimized version may be impractical due to the indeterminacy of typical quantum algorithms and the substantial expense involved in executing or simulating them. Indeed, resources may be too scarce, or the qubit connectivity too constrained, to run the program without optimization!

An appealing solution to this problem is to apply rigorous *formal methods* to prove that an optimization or algorithm always does what it is intended to do. As an example, consider CompCert (Leroy et al., 2004), which is a compiler for C programs that is written and proved correct using the Coq proof assistant (Coq Development Team, 2019). CompCert includes sophisticated optimizations whose proofs of correctness are verified to be valid by Coq's type checker.

In this paper, we apply CompCert's approach to the quantum setting. We present VOQC (pronounced "vox"), a *verified optimizer for quantum circuits*. VOQC takes as input a quantum program written in a language we call SQIR ("squire"). While SQIR is designed to be a *simple quantum intermediate representation*, it is not very different from languages such as PyQuil (Rigetti Computing, 2019), which are used to construct quantum source programs as circuits. VOQC applies a series of optimizations to SQIR programs, ultimately

[1]`https://github.com/Qiskit/qiskit-terra/issues/2752`

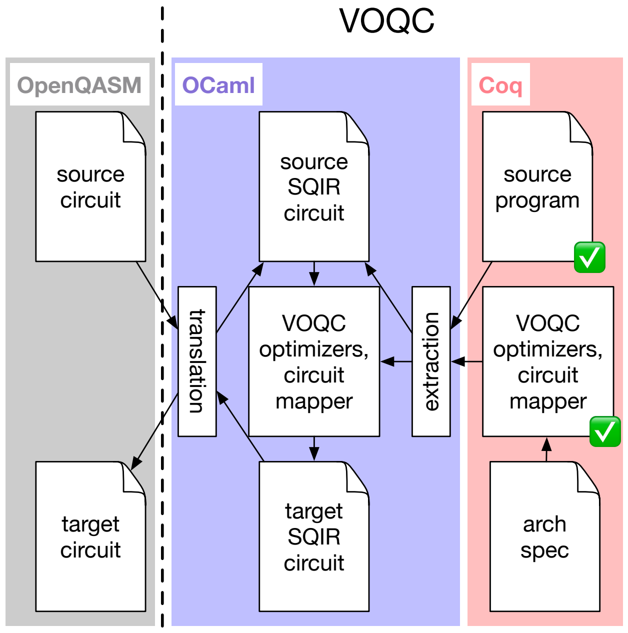

Figure 1: VOQC architecture

producing a result that is compatible with the specified quantum architecture. For added convenience, VOQC provides translators between SQIR and OpenQASM, the de facto standard format for quantum circuits (Cross et al., 2017). The structure of VOQC is shown in Figure 1.

Like CompCert, VOQC is implemented using the Coq proof assistant. SQIR program ASTs are represented in Coq via a deep embedding, and optimizations are implemented as Coq functions, which are then extracted to OCaml. We define two semantics for SQIR programs. The simplest denotes every quantum circuit as a *unitary matrix*. However this is only applicable to unitary circuits, i.e., circuits without measurement. For non-unitary circuits, we provide a denotation of *density matrices*. Properties of SQIR programs, or transformations of them, can be proved using whichever semantics is most convenient. As examples, we have proved correctness properties about several source programs written in SQIR, including *GHZ state preparation*, *quantum teleportation*, *superdense coding*, and the *Deutsch-Jozsa algorithm*. Generally speaking, SQIR is designed to make proofs as easy as possible. For example, we initially contemplated SQIR programs accessing qubits as Coq variables via higher order abstract syntax (Pfenning and Elliott, 1988), but we found that proofs were far simpler when qubits were accessed as concrete indices into a global register.

We have implemented and proved correct several transformations over SQIR programs. In particular, we have developed verified versions of many of the optimizations used in a state-of-the-art compiler developed by Nam et al. (2018). We have also verified a circuit mapping routine that transforms SQIR programs to satisfy constraints on how qubits may interact on a particular target architecture. These transformations were reasonably straightforward to prove correct thanks to SQIR's design.

We find that VOQC performs comparably to unverified, state-of-the-art compilers when run on a benchmark of 29 circuit programs developed by Amy et al. (2013). These programs range from 45 and 61,629 gates and use between 5 and 192 qubits. VOQC reduced total gate counts on average by 17.7% compared to 10.7% by IBM's Qiskit compiler (Aleksandrowicz et al., 2019). There is still room for improvement: Nam et al. (2018) produced reductions of 26.5% using additional optimizations we expect we could verify. The full results of our evaluations are shown in Table 1.

VOQC is the first fully verified circuit optimizer for a realistic quantum circuit language. Amy et al. (2017) developed a verified optimizing compiler from source Boolean expressions to reversible circuits, but did not handle general quantum programs. Rand et al. (2018) developed a similar compiler for quantum circuits but without optimizations. Other low-level quantum languages (Cross et al., 2017; Smith et al., 2016) have not been developed with verification in mind, and prior circuit-level optimizations (Amy et al., 2013; Heyfron and T. Campbell, 2017; Nam et al., 2018) have not been formally verified. In concurrent work, Shi et al. (2019) developed CertiQ, which uses symbolic execution and SMT solving to verify some circuit transformations in the Qiskit compiler. CertiQ is limited to verifying correct application of local equivalences, rather than more general circuit transformations, and sometimes verification fails in which case CertiQ must validate the optimized ("translated") circuits on-line. The PyZX compiler (Kissinger and van de Wetering, 2019) likewise uses translation validation to check its rewrites of circuits using the equational theory of the ZX-Calculus (Coecke and Duncan, 2011).

Our work on voqc and sqir constitutes a step toward developing a full-scale verified compiler toolchain. Next steps include developing certified transformations from high-level quantum languages to sqir and implementing optimizations with different objectives, e.g., that aim to reduce the probability that a result is corrupted by quantum noise.

References

G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, J. M. Chow, A. D. Córcoles-Gonzales, A. J. Cross, A. Cross, J. Cruz-Benito, C. Culver, S. D. L. P. González, E. D. L. Torre, D. Ding, E. Dumitrescu, I. Duran, P. Eendebak, M. Everitt, I. F. Sertage, A. Frisch, A. Fuhrer, J. Gambetta, B. G. Gago, J. Gomez-Mosquera, D. Greenberg, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, H. Horii, S. Hu, T. Imamichi, T. Itoko, A. Javadi-Abhari, N. Kanazawa, A. Karazeev, K. Krsulich, P. Liu, Y. Luh, Y. Maeng, M. Marques, F. J. Martín-Fernández, D. T. McClure, D. McKay, S. Meesala, A. Mezzacapo, N. Moll, D. M. Rodríguez, G. Nannicini, P. Nation, P. Ollitrault, L. J. O'Riordan, H. Paik, J. Pérez, A. Phan, M. Pistoia, V. Prutyanov, M. Reuter, J. Rice, A. R. Davila, R. H. P. Rudy, M. Ryu, N. Sathaye, C. Schnabel, E. Schoute, K. Setia, Y. Shi, A. Silva, Y. Siraichi, S. Sivarajah, J. A. Smolin, M. Soeken, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, K. Trabing, M. Treinish, W. Turner, D. Vogt-Lee, C. Vuillot, J. A. Wildstrom, J. Wilson, E. Winston, C. Wood, S. Wood, S. Wörner, I. Y. Akhalwaya, and C. Zoufal. Qiskit: An open-source framework for quantum computing, 2019.

M. Amy, D. Maslov, and M. Mosca. Polynomial-time t-depth optimization of clifford+t circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33, 03 2013. doi: 10.1109/TCAD.2014.2341953.

M. Amy, M. Roetteler, and K. M. Svore. Verified compilation of space-efficient reversible circuits. In *Proceedings of the 28th International Conference on Computer Aided Verification (CAV 2017)*. Springer, July 2017. URL `https://www.microsoft.com/en-us/research/publication/verified-compilation-of-space-efficient-reversible-circuits/`.

B. Coecke and R. Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

Coq Development Team. The Coq proof assistant reference manual, version 8.9, 2019. Electronic resource, available from `https://coq.inria.fr/refman/`.

A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open Quantum Assembly Language. *arXiv e-prints*, art. arXiv:1707.03429, Jul 2017.

L. Heyfron and E. T. Campbell. An efficient quantum compiler that reduces t count. *Quantum Science and Technology*, 4, 12 2017. doi: 10.1088/2058-9565/aad604.

A. Kissinger and J. van de Wetering. Pyzx: Large scale automated diagrammatic reasoning. In *Proceedings of the 16th International Conference on Quantum Physics and Logic, QPL 2019*, June 2019.

X. Leroy et al. The compcert verified compiler. *Development available at http://compcert. inria. fr*, 2009, 2004.

Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):23, 2018. doi: 10.1038/s41534-018-0072-4. URL `https://doi.org/10.1038/s41534-018-0072-4`.

F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*, PLDI '88, pages 199–208, New York, NY, USA, 1988. ACM. doi: 10.1145/53990.54010.

R. Rand, J. Paykin, D.-H. Lee, and S. Zdancewic. ReQWIRE: Reasoning about reversible quantum circuits. In *Proceedings of the 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Nova Scotia, 3-7 June 2018*, 2018.

Rigetti Computing. Pyquil documentation, 2019. URL `http://pyquil.readthedocs.io/en/latest/`.

Y. Shi, X. Li, R. Tao, A. Javadi-Abhari, A. W. Cross, F. T. Chong, and R. Gu. Contract-based verification of a realistic quantum compiler. *arXiv e-prints*, art. arXiv:1908.08963, Aug 2019.

R. S. Smith, M. J. Curtis, and W. J. Zeng. A Practical Quantum Instruction Set Architecture. *arXiv e-prints*, art. arXiv:1608.03355, Aug 2016.

V. Zamdzhiev. Quantum computing: The good, the bad, and the (not so) ugly!, Mar. 2016. Invited talk, Tulane University.

Benchmark Name	**Original**	**Nam (L)**	**Nam (H)**	**Qiskit A**	**Qiskit B**	**VOQC**
adder_8	900	646	**606**	869	805	752
barenco_tof_10	450	294	**264**	427	394	380
barenco_tof_3	58	42	**40**	56	51	51
barenco_tof_4	114	78	**72**	109	100	98
barenco_tof_5	170	114	**104**	162	149	145
csla_mux_3	170	161	**155**	168	156	160
csum_mux_9	420	294	**266**	420	382	308
gf2^4_mult	225	**187**	**187**	213	206	192
gf2^5_mult	347	296	296	327	318	**291**
gf2^6_mult	495	**403**	**403**	465	454	410
gf2^7_mult	669	555	555	627	614	**549**
gf2^8_mult	883	712	712	819	804	**705**
gf2^9_mult	1095	891	891	1023	1006	**885**
gf2^10_mult	1347	**1070**	**1070**	1257	1238	1084
gf2^16_mult	3435	2707	2707	3179	3148	**2695**
gf2^32_mult	13562	10601	10601	12569	12506	**10577**
gf2^64_mult	61629	41563	N/A	49659	49532	**41515**
mod5_4	63	**51**	**51**	62	58	57
mod_mult_55	119	91	91	117	106	**90**
mod_red_21	278	184	**180**	261	227	214
qcla_adder_10	521	411	**399**	512	469	474
qcla_com_7	443	**284**	**284**	428	398	335
qcla_mod_7	884	636	**624**	853	793	764
rc_adder_6	200	142	**140**	195	170	167
tof_10	255	**175**	**175**	247	222	215
tof_3	45	**35**	**35**	44	40	40
tof_4	75	**55**	**55**	73	66	65
tof_5	105	**75**	**75**	102	92	90
vbe_adder_3	150	**89**	**89**	146	138	103
Average Reduction	–	25.2%	26.5%	4.6%	10.7%	17.7%

Table 1: Reduced gate counts on the Amy et al. (2013) benchmarks. **(L)** refers to Nam et al. light optimization and **(H)** refers to Nam et al. heavy optimization. **Qiskit A** includes all optimizations used in level 2 of IBM's Qiskit compiler over the gate set $\{H, X, R_{\pi/4}, CNOT\}$, where $R_{\pi/4}$ is a rotation about the z-axis by a multiple of $\pi/4$ (VOQC uses the same gate set). **Qiskit B** includes all optimizations used in Qiskit's highest level of optimization over the more general gate set $\{u1, u2, u3, CNOT\}$, which allows single-qubit gates to be expressed as arbitrary rotations. We used Qiskit version 0.13.0. In each row, we have marked in bold the gate count of the best-performing optimizer. The average gate count reduction for each optimizer is given in the last row.