Software Impacts 7 (2021) 100051

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

QCEC: A JKQ tool for quantum circuit equivalence checking

Lukas Burgholzer [a,*], Robert Wille [a,b]

[a] Institute for Integrated Circuits, Johannes Kepler University Linz, 4040 Linz, Austria
[b] Software Competence Center Hagenberg GmbH (SCCH), 4232 Hagenberg, Austria

ARTICLE INFO

Keywords:
Quantum computing
Equivalence checking
Decision diagrams

ABSTRACT

Quantum computing is gaining serious momentum in these days. With increasing capabilities of corresponding devices also comes the need for efficient and automated tools to design them. Verification, i.e., ensuring that the originally intended functionality of a quantum algorithm/circuit is preserved throughout all layers of abstraction during the design process, is a vital part of the quantum software stack. In this work, we present *QCEC*, a tool for quantum circuit equivalence checking which is part of the JKQ toolset for quantum computing. By exploiting characteristics unique to quantum computing, the tool allows users to efficiently verify the equivalence of two quantum circuits using a variety of methods and strategies.

Code metadata

Current code version	v1.6.2
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2020-66
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/8165627/tree/v1
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	C++, Python
Compilation requirements, operating environments & dependencies	cmake ≥ 3.10, C++-14 compatible compiler
If available Link to developer documentation/manual	
Support email for questions	iic-quantum@jku.at

Software metadata

Current software version	v1.6.2
Permanent link to executables of this version	https://pypi.org/project/jkq.qcec/1.6.2/
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/8165627/tree/v1
Legal Software License	MIT
Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows
Installation requirements & dependencies	C++-14 compatible compiler
If available, link to user manual–if formally published include a reference to the publication in the reference list	
Support email for questions	iic-quantum@jku.at

1. Introduction

Quantum computers promise to speed up certain key applications, including integer factorization [1], unstructured search [2], chemistry [3], finance [4], etc. To this end, quantum computers utilize quantum-mechanical effects, such as superposition, entanglement and interference [5]. In these days, quantum computing is gaining serious momentum and quantum computers might overtake conventional computers on some of these tasks in the near future [6].

Quantum algorithms are typically described by so-called quantum circuits which specify a sequence of operations to be applied to a quantum system. Initially, quantum algorithms are described in a rather

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: lukas.burgholzer@jku.at (L. Burgholzer), robert.wille@jku.at (R. Wille).

https://doi.org/10.1016/j.simpa.2020.100051
Received 21 December 2020; Accepted 22 December 2020

2665-9638/© 2020 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

high-level, device-agnostic fashion—think of it as a description in a high-level programming language. Just as classical software, several steps are necessary to *compile* a desired quantum algorithm to a representation that can be executed on the target device—similar to assembly code of a program only using a specific instruction set. Corresponding methods have been developed for this task and are described, e.g., in [7–12].

However, it is of utmost importance that, during all corresponding abstraction levels, the originally intended functionality is indeed preserved. This motivates the development of methods for verification or, more precisely, equivalence checking. Here, given two quantum circuits G and G', the task is to check whether they realize the same functionality. Since each operation in quantum computing can be described by a complex-valued unitary matrix, the functionality of the whole circuit can be determined by successive matrix–matrix multiplication. Comparing the resulting matrices then solves the equivalence checking problem.

Unfortunately, the underlying descriptions (i.e., the resulting matrices) grow exponentially with respect to the systems' size, i.e., with respect to the number of qubits of the quantum circuit/algorithm. In fact, it has been shown that equivalence checking of quantum circuits is QMA-complete[1] in general [13]. Several approaches to this problem (e.g., [14–20]) have been proposed.

In this article, we consider solutions that rely on decision diagrams (a data structure which frequently allows to represent quantum functionality in a very compact fashion) and their clever utilization, as well as schemes that exploit the power of simulation in quantum computing. More precisely, we present QCEC, a tool for quantum circuit equivalence checking which is part of the JKQ [21] toolset for quantum computing. By explicitly exploiting characteristics unique to quantum computing and employing the data structure in a clever fashion, the tool allows users to efficiently verify the equivalence of two quantum circuits using a variety of methods and strategies.

2. Description and features

QCEC is a software package for quantum circuit equivalence checking that is mainly developed in C++, runs under any major operating system, and also provides Python bindings in order to be as accessible as possible to its community. From a user perspective, the tool takes two quantum circuits (in either of a multitude of file formats, such as *.qasm*, *.real*, *.qc*, etc.) and returns whether the provided circuits are equivalent or not. To this end, several different methods are available—each providing their own advantages and disadvantages. At its core, these methods are based on two complementary observations about the verification of quantum circuits (for details we refer to [22] and [23], respectively):

1. Due to the inherent reversibility of quantum circuits, if two quantum circuits G and G' are equivalent, then concatenating the first circuit G with the inverse G'^{-1} of the second circuit would realize the identity function $\mathbb{I}$. The potential now lies in the order in which the operations from either circuit are applied. Whenever a strategy can be employed so that the respective gates from G and G' are applied in a fashion frequently yielding the identity, the entire procedure can be conducted rather efficiently. Experimental results have shown that this leads to speed-ups of several factors or even magnitudes.
2. In the classical realm, verification methods based on simulation often require a complete consideration of *all* possible input states or sophisticated schemes for constraint-based stimuli generation, fuzzing, etc., due to the inevitable information loss introduced by many logic gates and the resulting masking effects. In contrast, we observed that, again due to the inherent reversibility of quantum operations, even small differences in quantum circuits frequently affect the *entire* functional representation. Hence, it may not always be necessary to check the complete functionality, but it is highly likely that the simulation of both circuits with a couple of randomly-chosen input states will already provide a counterexample showing the non-equivalence.

A summary of all available methods, including a brief description and short remarks on each technique, is provided in Table 1.

3. Impact overview

QCEC started out based on the initial ideas in [22] and [23]. Since then, it enabled several new ideas which have been integrated into the tool afterwards. For example:

1. By explicitly incorporating knowledge about the compilation flow, a strategy has been designed that allows to keep the overhead of verifying compilation results minimal [25]. Experimental evaluations confirm that the proposed strategy consistently allows to verify instances with more than ten-thousand gates within seconds—while state-of the-art techniques often require substantial runtime or even time-out in these tasks.
2. The simulation capabilities have been further refined by introducing dedicated quantum stimuli generation schemes [26]. They offer a trade-off between error detection rate and efficiency. This eventually shows that simulation-based verification offers huge potential in the verification of quantum circuits.
3. The equivalence checking methodology has also been integrated into the JKQ DDVis tool, which visualizes decision diagrams for quantum computing and their applications [27]. It allows users unfamiliar with the concept of decision diagrams to visually explore and learn how they can be utilized in the verification of quantum circuits.[2]

Overall, the tool allows users to check the equivalence of two quantum circuits in an efficient fashion. A comprehensive description of the main concepts has been published in [28]. The underlying methodology is generic and can be fine-tuned to specific application scenarios (e.g., for the verification of compilation results as mentioned above). Moreover, QCEC is designed with accessibility in mind. Assume that you want to check the equivalence of two quantum circuits described by files `circ1.qasm` and `circ2.qasm`. Then, using our tool for this task is as easy as executing the following statements in Python:

```
from jkq import qcec
result = qcec.verify("circ1.qasm", "circ2.qasm")
print(result["equivalence"])
```

Listing 1: Checking the equivalence of two circuits using QCEC

4. Conclusions and outlook

In this article, we presented the easy-to-use and efficient quantum circuit equivalence checking tool QCEC that is part of the JKQ toolset for quantum computing. The tool is available at https://github.com/iic-jku/qcec and detailed descriptions of the corresponding methods are provided in [25,26,28]. Due to the generic nature of the underlying methodology, the tool's capabilities can be tailored to more specific application scenarios in the future (e.g., verifying specific optimization methods for quantum circuits). Moreover, the tool should be easy to incorporate in other (industrial) toolkits such as, e.g., IBM's Qiskit, Google's CirQ, Rigetti's Forest, etc.

[1] QMA can be regarded as the quantum analogue of the classical complexity class NP.

[2] An instance of the visualization tool is hosted at https://iic.jku.at/eda/research/quantum_dd/tool/.

Table 1
Overview of methods provided by QCEC.

Method	Strategy	Description	Remarks
Standard DD	Ref. [24]	Construct and compare the decision diagram for both circuits	Potentially constructs two large decision diagrams
$G \rightarrow \mathbb{I} \leftarrow G'$	Naive [22]	Alternate between applications of G and G'	The most simple strategy
	Proportional [22]	Proportionally apply gates according to the gate count ratio of G and G'	The most reliable general-purpose strategy
	Lookahead [22]	Always apply the gate yielding the smaller decision diagram	High gain, high risk strategy
	Compilation Flow [25]	A dedicated scheme for verifying results of the IBM Qiskit Compilation Flow explicitly exploiting certain knowledge about the compilation process	Best performance for specific application scenario
Simulation	Classical [23]	Computational basis states	Very fast, but cannot detect certain rotations
	Local Quantum [26]	Each qubit value is independently chosen from any of the six basis states ($\|0\rangle, \|1\rangle, \|+\rangle, \|-\rangle, \|L\rangle, \|R\rangle$)	Fast and reliable
	Global Quantum [26]	Random stabilizer states	Slow, but very reliable

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab, Austria funded by the State of Upper Austria as well as by the BMK, Austria, BMDW, Austria, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (5) (1997) 1484–1509.
[2] L.K. Grover, A fast quantum mechanical algorithm for database search, Proc. ACM (1996) 212–219.
[3] S. McArdle, S. Endo, A. Aspuru-Guzik, S.C. Benjamin, X. Yuan, Quantum computational chemistry, Rev. Modern Phys. 92 (1) (2020) 015003.
[4] P. Rebentrost, B. Gupt, T.R. Bromley, Quantum computational finance: Monte Carlo pricing of financial derivatives, Phys. Rev. A 98 (2018).
[5] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2010.
[6] J. Preskill, Quantum computing in the NISQ era and beyond, Quantum 2 (2018) 79.
[7] R. Wille, L. Burgholzer, A. Zulehner, Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations, in: Design Automation Conf., 2019.
[8] K.N. Smith, M.A. Thornton, A quantum computational compiler and design tool for technology-specific targets, in: Int'L Symp. on Computer Architecture, 2019, pp. 579–588.
[9] A. Zulehner, A. Paler, R. Wille, An efficient methodology for mapping quantum circuits to the IBM QX architectures, IEEE Trans. CAD Int. Circuits Syst. 38 (7) (2019) 1226–1236.
[10] P. Murali, J.M. Baker, A. Javadi-Abhari, F.T. Chong, M. Martonosi, Noise-adaptive compiler mappings for Noisy Intermediate-Scale Quantum computers, in: Int'L Conf. on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 1015–1029.
[11] A. Matsuo, W. Hattori, S. Yamashita, Reducing the overhead of mapping quantum circuits to IBM Q system, in: IEEE International Symposium on Circuits and Systems, 2019.
[12] G. Li, Y. Ding, Y. Xie, Tackling the qubit mapping problem for NISQ-era quantum devices, in: Int'L Conf. on Architectural Support for Programming Languages and Operating Systems, 2019.
[13] D. Janzing, P. Wocjan, T. Beth, "Non-identity check" is QMA-complete, Int. J. Quantum Inform. 03 (03) (2005) 463–473.
[14] G.F. Viamontes, I.L. Markov, J.P. Hayes, Checking equivalence of quantum circuits and states, in: Int'L Conf. on CAD, 2007.
[15] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, S.-Y. Kuo, An XQDD-based verification method for quantum circuits, in: IEICE Trans. Fundamentals, 2008, pp. 584–594.
[16] S. Yamashita, I.L. Markov, Fast equivalence-checking for quantum circuits, in: Int'L Symp. on Nanoscale Architectures, 2010.
[17] P. Niemann, R. Wille, R. Drechsler, Equivalence checking in multi-level quantum systems, in: Int'L Conf. of Reversible Computation, 2014.
[18] Y. Shi, X. Li, R. Tao, A. Javadi-Abhari, A.W. Cross, F.T. Chong, R. Gu, Contract-based verification of a realistic quantum compiler, 2019, arXiv:1908.08963.
[19] W. Shi, Q. Cao, Y. Deng, H. Jiang, Y. Feng, Symbolic reasoning about quantum circuits in Coq, 2020, arXiv:2005.11023.
[20] K. Hietala, R. Rand, S.-H. Hung, L. Li, M. Hicks, Proving quantum programs correct, 2020, arXiv:2010.01240.
[21] R. Wille, S. Hillmich, L. Burgholzer, JKQ: JKU tools for quantum computing, in: Int'L Conf. on CAD, 2020.
[22] L. Burgholzer, R. Wille, Improved DD-based equivalence checking of quantum circuits, in: Asia and South Pacific Design Automation Conf., 2020, pp. 127–132.
[23] L. Burgholzer, R. Wille, The power of simulation for equivalence checking in quantum computing, in: Design Automation Conf., 2020.
[24] P. Niemann, R. Wille, D.M. Miller, M.A. Thornton, R. Drechsler, QMDDS: Efficient quantum function representation and manipulation, IEEE Trans. CAD Int. Circuits Syst. 35 (1) (2016) 86–99.
[25] L. Burgholzer, R. Raymond, R. Wille, Verifying results of the IBM Qiskit quantum circuit compilation flow, in: Int'L Conf. on Quantum Computing and Engineering, 2020, pp. 356–365.
[26] L. Burgholzer, R. Kueng, R. Wille, Random stimuli generation for the verification of quantum circuits, in: Asia and South Pacific Design Automation Conf., 2021.
[27] R. Wille, L. Burgholzer, M. Artner, Visualizing decision diagrams for quantum computing, in: Design, Automation and Test in Europe, 2021.
[28] L. Burgholzer, R. Wille, Advanced equivalence checking for quantum circuits, IEEE Trans. CAD Int. Circuits Syst. (2021).