JKQ: JKU Tools for Quantum Computing

(Talk Proposal)

ROBERT WILLE, Johannes Kepler University Linz, Austria and Software Competence Center Hagenberg GmbH (SCCH), Austria
STEFAN HILLMICH, Johannes Kepler University Linz, Austria
LUKAS BURGHOLZER, Johannes Kepler University Linz, Austria

1 INTRODUCTION

In this work, we present *JKQ*—a set of corresponding tools developed at the *Johannes Kepler University (JKU) Linz*. *JKQ* provides a variety of software solutions for quantum computing that addresses the respective tasks and challenges, e.g., for quantum circuit simulation, compilation, or verification in a fashion that explicitly utilizes the potential and the expertise of the design automation community. More precisely, the methods realized in JKQ rest on dedicated data-structures (e.g., decision diagrams) or solving methods (e.g., reasoning engines) which are established and got perfected for design automation of conventional circuits in the past decades, but have been re-developed in order to take the particular characteristics of quantum computing into account.

The *JKQ* tools are publicly available as open-source implementations under the MIT license at https://github.com/iic-jku/. In this talk proposal, we briefly summarize the main tools (covering quantum circuit simulation, compilation, and verification). We particularly focus on how to use the tools, but additionally provide references/links that offer detailed descriptions of the underlying methods as well as summaries of corresponding case studies and evaluations demonstrating the performance.

2 SIMULATION

Simulating quantum circuits on a classical machine is essential for the development of (prototypes of) quantum algorithms, for considering the behavior of physical quantum computers, as well as for studying error models. In fact, classical simulations of quantum computations provide deeper insights, since it allows to observe the individual amplitudes of a quantum state (which is impossible when executing a quantum computation on a quantum device). Moreover, costly executions on real quantum devices—whose availability is still rather limited—can be avoided when using simulators.

The classical simulation of quantum circuits is commonly conducted by performing consecutive matrix-vector multiplication, which many simulators realize by storing a dense representation of the complete state vector in memory and evolving it correspondingly (see, e.g., [1–5]). This approach quickly becomes intractable due to the exponential growth of the quantum state with respect to the number of qubits—quickly rendering such simulations infeasible even on supercomputer clusters. Simulation methodologies based on decision diagrams [6, 7] are a promising complementary approach that frequently allows to reduce the required memory by exploiting redundancies in the simulated quantum state.

JKQ offers the decision diagram-based quantum circuit simulator *DDSIM* which allows to simulate quantum circuits defined in either *.qasm* [8] or *.real* [9] format alongside, e.g., parametrized instances of Grover's [10] or Shor's [11] algorithm. Using JKQ DDSIM requires cloning the GitHub repository https://github.com/iic-jku/ddsim and building the tool as follows[1]:

```
$ git clone https://github.com/iic-jku/ddsim
$ cd ddsim
$ cmake -DCMAKE_BUILD_TYPE=Release -S . -B build
[...]
$ cmake --build build --config Release --target ddsim_simple
[...]
$ ./build/ddsim_simple --help
[displays available commands]
```

Example 1. *Simulating Grover's algorithm with a two qubit oracle using the JKQ DDSIM simulator can be conducted as follows:*

```
$ ./ddsim_simple simulate --simulate_grover 2 --shots 1000 --ps
{
  "measurements": {
    "000": 503,
    "100": 497
  },
  "non_zero_entries": 2,
  "statistics": {
    "simulation_time": 0.125837,
    "measurement_time": 0.000180,
    "benchmark": "grover_2",
    "shots": 1000,
    "n_qubits": 3,
    "applied_gates": 16,
    "max_nodes": 8,
    "seed": 0
  }
}
```

Here, the parameters define the number of measurements to be performed on the final quantum state (`--shots 1000`*) and cause the simulator to print statistics (*`--ps`*). The output is formatted according to the JSON standard and, hence, machine readable for further processing.*

Simulations of a given circuit file can be started with the parameter `--simulate_file <file>` (the respective format is derived from the extension). The full set of parameters can be listed via `./ddsim_simple --help`. This includes advanced techniques such as emulation [12] (which enable significant speedups for certain quantum algorithms), weak simulation [13] (which more faithfully mimics a physical quantum computer), approximate simulation [14] (which allows to perform simulations as efficient as possible while remaining as accurate as needed), and stochastic noise-aware simulation [15] (which allows to classically simulate the effects of noise on a circuit's execution).

[1]Building requires a C++17-compatible compiler, CMake ≥ 3.14, and boost (`program_options`).

3 COMPILATION

Since superconducting quantum computers in the NISQ era are bound by connectivity constraints, only support a limited set of elementary gates, and are heavily affected by noise, high-level descriptions of quantum algorithms have to be *compiled* through different layers of abstraction before being executable on the actual quantum computer. A major part of this *compilation* flow consists of *mapping*, i.e., making an already decomposed circuit conform to the device's connectivity constraints (usually provided as a coupling map).

A circuit is typically *mapped* to the actual device by inserting *SWAP* operations into the circuit—dynamically permuting the location of the logical qubits on the device's physical qubits such that each operation conforms to the coupling map. Due to the inherent influence of noise and short coherence times of today's quantum computers, it is of utmost importance to keep the overhead induced by the mapping procedure as low as possible. As this problem has been shown to be NP-complete [16], there is a high demand for automated and efficient solutions.

JKQ offers the quantum circuit mapping tool *QMAP* that allows to generate circuits which satisfy all constraints given by the targeted architecture and, at the same time, keep the overhead in terms of additionally required quantum gates as low as possible. More precisely, two different approaches based on design automation techniques are provided, which are both generic and can be easily configured for future architectures: The first one is a general solution for arbitrary circuits based on informed-search algorithms [17]. The second one is a solution for obtaining mappings ensuring minimal overhead with respect to SWAP gate insertions [18][2]. Similar to our simulator, JKQ QMAP can be obtained from the GitHub repository https://github.com/iic-jku/qmap and can subsequently be used by either building the `qmap_heuristic` or the `qmap_exact` CMake target. However, there is an even easier way for users to get started with the JKQ QMAP tool. Specifically, the tool is also provided as a Python package, which can be easily installed with `pip install jkq.qmap` and also provides native integration with IBM Qiskit.

Example 2. *Assume we want to perform the computation simulated in Ex. 1 on the five-qubit IBMQ London quantum computer and assume the circuit has already been decomposed into a sequence of single-qubit and CNOT gates. Then, compiling the circuit merely requires the following lines of Python:*

```
1 from jkq import qmap

3 qmap.compile("grover_2.qasm", qmap.Arch.IBMQ_London)
```

A complete list of the available methods as well as additional configuration options can be listed via `help(qmap.compile)`. This includes the recently published work on exploiting quantum teleportation in quantum circuit mapping [19].

[2]In order to use this method the Z3 theorem prover library (https://github.com/Z3Prover/z3) needs to be installed.

4 VERIFICATION

Compiling quantum algorithms results in different representations of the considered functionality, which significantly differ in their basis operations and structure but are still supposed to be functionally equivalent. Consequently, checking whether the originally intended functionality is indeed maintained throughout all these different abstractions becomes increasingly relevant in order to guarantee an efficient, yet correct design flow. Existing solutions for equivalence checking of quantum circuits suffer from significant shortcomings due to the immense complexity of the underlying problem—which has been proven to be QMA-complete [20]. However, certain quantum mechanical characteristics provide impressive potential for efficient equivalence checking of quantum circuits.

JKQ offers the quantum circuit equivalence checking tool *QCEC* which explicitly exploits these characteristics based on the ideas outlined in [21–23]. Similar to other JKQ tools, JKQ QCEC can be obtained from the GitHub repository https://github.com/iic-jku/qcec and can subsequently be used by building the `qcec_app` CMake target. Additionally, as for QMAP, the tool is also provided as a Python package, which can be easily installed with `pip install jkq.qcec` and also provides native integration with IBM Qiskit.

Example 3. *Verifying that the quantum circuit from Ex. 1 has been compiled correctly in Ex. 2 merely requires the following lines of Python:*

```
from jkq.qcec import Configuration, verify

# potentially set configuration options
config = Configuration()

# verify the compilation result
result = verify("grover_2.qasm", "grover_2m.qasm", config)
```

A complete list of the available methods as well as configuration options can be listed via `help(Configuration)` in Python or `qcec_app --help` when using the commandline app. This includes a strategy especially suited for verifying compilation results [24], as well as dedicated random stimuli generation schemes [25].

5 CONCLUSIONS

In this talk proposal, we presented *JKQ*—a set of tools for quantum computing utilizing the potential and expertise of design automation. The descriptions focused on how to use the tools, but references/links have been provided that allow for a more in-depth treatment. The *JKQ* toolset is available at https://github.com/iic-jku.

REFERENCES

[1] T. Häner and D. S. Steiger, "0.5 petabyte simulation of a 45-Qubit quantum circuit," in *Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, 2017.

[2] J. Doi, H. Takahashi, R. Raymond, T. Imamichi, and H. Horii, "Quantum computing simulator on a heterogenous HPC system," in *Int'l Conf. on Computing Frontiers*, 2019, pp. 85–93.

[3] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "QuEST and high performance simulation of quantum computers," in *Scientific Reports*, 2018.

[4] G. G. Guerreschi, J. Hogaboam, F. Baruffa, and N. P. D. Sawaya, "Intel Quantum Simulator: a cloud-ready high-performance simulator of quantum circuits," *Quantum Science and Technology*, 2020.

[5] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," in *Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–24.

[6] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2019.

[7] G. F. Viamontes, I. L. Markov, and J. P. Hayes, *Quantum Circuit Simulation.* Springer, 2009.

[8] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017.

[9] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: an online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008, pp. 220–225.

[10] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Theory of computing*, 1996.

[11] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Jour. of Comp.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[12] A. Zulehner and R. Wille, "Matrix-vector vs. matrix-matrix multiplication: Potential in DD-based simulation of quantum computations," in *Design, Automation and Test in Europe*, 2019.

[13] S. Hillmich, I. L. Markov, and R. Wille, "Just like the real thing: Fast weak simulation of quantum computation," in *Design Automation Conf.*, 2020.

[14] S. Hillmich, R. Kueng, I. Markov, and R. Wille, "As accurate as needed, as efficient as possible: Approximations in dd-based quantum circuit simulation," in *Design, Automation and Test in Europe*, 2021.

[15] T. Grurl, R. Kueng, J. Juß, and R. Wille, "Stochastic quantum circuit simulation using decision diagrams," in *Design, Automation and Test in Europe*, 2021.

[16] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Symp. on Combinatorial Search*, 2018.

[17] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2018.

[18] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Design Automation Conf.*, 2019.

[19] S. Hillmich, A. Zulehner, and R. Wille, "Exploiting quantum teleportation in quantum circuit mapping," in *Asia and South Pacific Design Automation Conf.*, 2021.

[20] D. Janzing, P. Wocjan, and T. Beth, ""Non-identity check" is QMA-complete," *Int'l Journal of Quantum Information*, 2005.

[21] L. Burgholzer and R. Wille, "Advanced equivalence checking of quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2021.

[22] L. Burgholzer and R. Wille, "Improved DD-based equivalence checking of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2020.

[23] L. Burgholzer and R. Wille, "The power of simulation for equivalence checking in quantum computing," in *Design Automation Conf.*, 2020.

[24] L. Burgholzer, R. Raymond, and R. Wille, "Verifying results of the IBM Qiskit quantum circuit compilation flow," in *Int'l Conf. on Quantum Computing and Engineering*, 2020.

[25] L. Burgholzer, R. Kueng, and R. Wille, "Random stimuli generation for the verification of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2021.