Computer-Aided Design for Quantum Computation

(Special Session Summary)

Robert Wille[1] Austin Fowler[2] Yehuda Naveh[3]
[1]Institute for Integrated Circuits, Johannes Kepler University, Linz, Austria
[2]Google Inc. Santa Barbara, USA
[3]IBM Research – Haifa, Israel
robert.wille@jku.at, agfowler@google.com, naveh@il.ibm.com

ABSTRACT

Quantum computation is currently moving from an academic idea to a practical reality. The recent past has seen tremendous progress in the physical implementation of corresponding quantum computers – also involving big players such as IBM, Google, Intel, Rigetti, Microsoft, and Alibaba. These devices promise substantial speedups over conventional computers for applications like quantum chemistry, optimization, machine learning, cryptography, quantum simulation, and systems of linear equations. The Computer-Aided Design and Verification (jointly referred as CAD) community needs to be ready for this revolutionizing new technology. While research on automatic design methods for quantum computers is currently underway, there is still far too little coordination between the CAD community and the quantum computation community. Consequently, many CAD approaches proposed in the past have either addressed the wrong problems or failed to reach the end users. In this summary paper, we provide a glimpse into both sides. To this end, we review and discuss selected accomplishments from the CAD domain as well as open challenges within the quantum domain. These examples showcase the recent state-of-the-art but also outline the remaining work left to be done in both communities.

1. INTRODUCTION

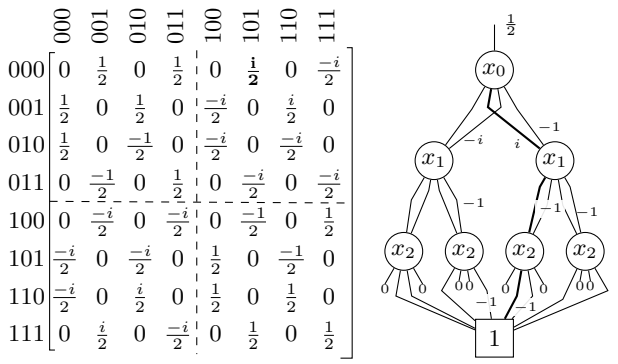
Quantum computations [20] promise substantial speedups over conventional computers for certain problems. This is because, in contrast to the bits of a conventional computer which can be either 0 or 1, the qubits of a quantum computer can be in an arbitrary superposition of both. Superposition serves as the basis for so-called quantum parallelism, which, in combination with other quantum mechanical effects such as entanglement and interference, allows for exponential speed-ups – in particular for applications like quantum chemistry, optimization, machine learning, cryptography, quantum simulation, and systems of linear equations.

While the theoretical power of quantum computation has already been known for a while (e.g., through the seminal work of Shor [30] showing that an important problem such as factorization can be solved in polynomial time on a quantum computer), actual quantum computers hardly exist yet. But the corresponding concepts are currently moving from an academic idea to a practical reality. In fact, the recent past has seen tremendous progress in the physical implementation of quantum computers. For example, IBM launched *IBM Q* [1], the first publicly available quantum processor, accessible through a cloud service. While the first processors (launched in 2017) started with 5 qubits, these have now increased to 20 qubit quantum processors – and a 50-qubit processor is reportedly being tested. Similarly, Google is working on a 72 qubit chip that it hopes will be able to demonstrate quantum advantage. Also additional big players such as Intel, Rigetti, Microsoft, and Alibaba heavily invest in quantum computation research.

The Computer-Aided Design and Verification (jointly referred as CAD) community needs to be ready for this revolutionizing new technology. While research on automatic design methods for quantum computers is currently underway, there is still far too little coordination between the CAD community and the quantum computation community. Consequently, many CAD approaches proposed in the past have either addressed the wrong problems, or failed to reach the end users. In this paper, we provide a glimpse into both sides: techniques and ideas recently proposed within the CAD domain as well as open challenges within the quantum domain. As covering all aspects of CAD for quantum computation certainly is beyond the scope of this summary paper, we focus on a selection of corresponding issues, namely

- recent accomplishments from the CAD community on data-structures and methods for the design of quantum computers,
- current attempts to build a reliable large-scale quantum computer and the resulting challenges, and
- challenges in the verification of quantum computers through the lenses of conventional verification methodologies.

These issues showcase the recent state-of-the-art in the design of quantum computers but also outline the huge amount of work left to be done. Moreover, they also illustrate how the CAD community can help in these endeavors and where open potential is still available.



(a) Matrix (b) Decision diagr.

Figure 1: Matrix and dec. diagr. of a 3-qubit computation

2. DATA-STRUCTURES AND METHODS FOR QUANTUM COMPUTATION

In the past decades, the conventional CAD community was frequently faced with tremendously complex challenges that often required the efficient consideration of problems of exponential (or even larger) size. In order to tackle those, researchers and engineers developed sophisticated CAD methods employing, e.g., data-structures based on decision diagrams or powerful reasoning engines. In contrast, many design problems in the quantum domain are still addressed in a rather straight-forward fashion, e.g., by exponential array-based descriptions or enumerative search algorithms. This section illustrates how these established concepts from the conventional design of circuits and systems can be applied to improve the design of quantum computers. To this end, two examples are considered: (1) Compact data-structures that may allow to represent quantum computations in a less than exponential fashion and (2) methods for maping quantum functionality to certain quantum architectures in a fashion that is more efficient than enumeration or random searches.

2.1 Data-structures for Quantum Computation

Quantum computations obviously differ significantly from conventional computations. Instead of Boolean bits, they rely on so-called qubits which can assume not only the values 0 and 1 but also superpositions of them. More formally, a qubit is a two-level quantum system which can be described by a two-dimensional complex Hilbert space. The state of a qubit can be denoted by a *state vector* $\binom{\alpha}{\beta}$, where α and β are complex numbers representing the amplitudes of the current state with respect to the basis states $|0\rangle$ and $|1\rangle$, respectively, and where a normalization constraint $|\alpha|^2 + |\beta|^2 = 1$ must hold. If $n > 1$ qubits are involved, the quantum state is defined by the tensor product of the respective single-qubit state spaces – leading to a normalized vector of dimension 2^n. Similarly, operations on quantum states can be represented by *unitary matrices* of size $2^n \times 2^n$. As an example, Fig. 1a shows a quantum operation over $n = 3$ qubits represented by a $2^3 \times 2^3 = 8 \times 8$ unitary matrix.

Hence, the simplest way to define a data-structure for quantum computation is to straight-forwardly represent the state vectors and unitary matrices in terms of 1-dimensional and 2-dimensional arrays, respectively. But since those representations grow exponentially with the size of the quantum systems (i.e., the number n of qubits), they quickly become infeasible. This can be observed, e.g., in corresponding applications such as [11, 33, 16] which rely on array representations and, hence, are only applicable for quantum computations with a rather small number of qubits.

Motivated by that, alternative representations are currently investigated. Here, data-structures based on *decision diagrams* are considered a promising approach [34, 35, 24]. They are designed to exploit redundancies and, by this, gain a more compact representation that is much smaller in many practically relevant cases. This is accomplished by representing a state vector and/or a unitary matrix in terms of a directed acyclic graph. For example, sub-matrices which occur multiple times are represented by a *shared graph structure*.

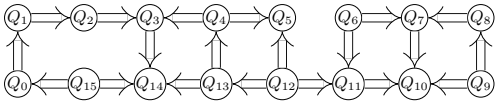
To illustrate that, consider Fig. 1b which shows a decision diagram-based representation of the unitary matrix of Fig. 1a. To obtain that, the matrix is partitioned, i.e., the $2^n \times 2^n$ matrix (represented by the top node) is split into four sub-matrices of dimension $2^{n-1} \times 2^{n-1}$ (represented by the top node's successors). This is recursively continued until only single matrix entries (represented by a terminal node $\boxed{1}$ or 0-stubs representing 0-entries) remain. Doing this partitioning allows for representing (structurally) equivalent sub-matrices by the same (shared) graph structure. For example, the top-left matrix and the bottom-left matrix (represented by the first and third successor) are structurally equivalent, i.e., differ only by the factor $-i$ in their respective matrix entries. Hence, both sub-matrices are represented by the same node, while the differences in the factor is represented by corresponding edge weights ($-i$ in case of the third successor; if no edge weight is annotated, the factor of 1 is assumed). These shared graph structures frequently allow to represent a unitary matrix in a much more efficient fashion than, e.g., by a complete 2-dimensional array. In fact, cases have been investigated where this yields a polynomial or even linear rather than an exponential representation of quantum computations.

Such data-structures eventually can help with several design tasks in quantum computation. First applications confirming that include, e.g., synthesis [21, 23, 41], verification [36, 22], and simulation [40]. In particular for simulation, this recently yielded impressive speed-ups, where a solution based on decision diagrams was able to simulate certain benchmarks in minutes, while other simulation approaches required hours or even days[1]. Moreover, the data-structure can often be used in a "black box" fashion, i.e., the details of the representation can be ignored and only the actually desired functionality has to be instantiated through proper interfaces (e.g., by providing a quantum circuit in an established quantum assembly language). Corresponding implementations of such a data-structure and its interface can be found at http://iic.jku.at/eda/research/quantum_dd.

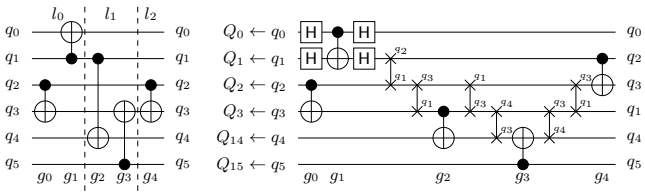
2.2 Methods for the Mapping to Quantum Architectures

One frequently occurring task in the field is to map a given quantum functionality (usually given in terms of a quantum circuit diagram and/or provided by corresponding languages such as Scaffold [4], Quipper [12], or OpenQASM [7]) to a corresponding quantum architecture. In the following, this is illustrated by means of IBM QX architectures. Those have

[1]However, please note that the performance of the data-structure often depends on the characteristics of the benchmark and the applied method. Developing an understanding what kind of representation best works for what kind of quantum computation is still an open research task.



(a) Coupling map for IBM QX3 architecture



(b) Original circuit (c) Mapped circuit

Figure 2: Mapping of a circuit to the *IBM QX3* architecture

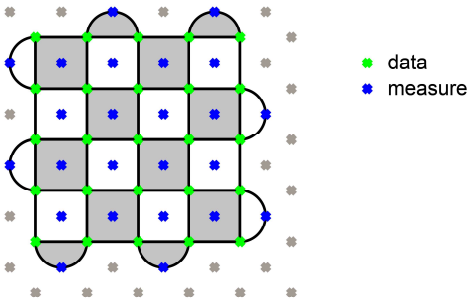


Figure 3: Layout of the 72 qubit "Bristlecone" chip

recently been released by IBM and, by a cloud access, can be utilized publicly. However, those architectures do not allow arbitrary interactions (i.e., operations), but are limited as described by a so-called *coupling graph* shown in Fig. 2a for the 16-qubit architecture IBM QX3. Here, nodes indicate physical qubits (denoted by Q_i) and arrows indicate the allowed interactions. More precisely, two-qubit operations can only be performed on qubits which are adjacent in the graph and where the edge direction goes from the control qubit to the target.

Now, having a quantum functionality over n logical qubits $q_0, q_1, \ldots q_{n-1}$, the problem is how to map this to the m physical qubits $Q_0, Q_1, \ldots Q_{m-1}$ of the architecture. Moreover, since there usually does not exist a mapping solution that satisfies all constraints given by the coupling graph, the mapping between logical qubits and physical qubits might change during the execution. To this end, so-called Hadamard (H) and SWAP gates can be applied to flip the direction of control and target qubits and to change the mapping of the logical qubits, respectively. In other words, these gates can be used to "move" the logical qubits on the actual architecture until the constraints are satisfied. For example, the quantum circuit shown in Fig. 2b[2] cannot directly be mapped to the IBM QX3 architecture, since the corresponding coupling map does not allow, e.g., the interaction between q_1 as a control and q_0 as a target in the second gate as well as the interaction between q_1 and q_4 at all in the third gate. Applying H and SWAP gates as shown in Fig. 2c[3] resolves this problem.

However, inserting the additional gates to satisfy the constraints imposed by the coupling graph drastically increases the number of gates – which in turn significantly increases the probability of errors during the computation. Hence, minimizing the number of H and SWAP gates is a primary objective. This constitutes a typical combinatorial task which is very similar to what frequently has to be addressed in the design of conventional circuits and systems. Accordingly, the CAD community has started to develop solutions which tackle this problem by exploiting efficient reasoning methods established in conventional circuit design. Initially, this has been done for so-called nearest neighbor architectures (which impose similar constraints and where also SWAP gates have been applied to move qubits together in order to satisfy physical constraints), e.g., in [38, 28, 37, 29, 13]. Recently, also constraints for IBM architectures have been considered [39][4]. They already can outperform solutions from the quantum community (such as provided by IBM's own SDK QISKIT [3]) which rely on solutions based on enumerations or random searches [3] rather than sophisticated methods such as A*, reasoning, or constraint-based engines regularly employed by the CAD community.

3. BUILDING A RELIABLE LARGE-SCALE QUANTUM COMPUTER

Researchers are currently working to build reliable large-scale quantum computers. In this section, we briefly describe the current status of Google's project and the associated challenges. The 72 qubit "Bristlecone" chip is geometrically laid out as shown in Fig. 3. This chip has only nearest-neighbor interactions but is large enough to handle, one day, a distance 5 surface code [14] in two different positions – enabling alternation between these two positions to facilitate the suppression of leakage [10, 9].

However, many aspects of the system need to be improved to run the surface code. For example:

- First, it is still common for a small number of wires on each chip to not work. The wiring yield needs to be raised to 100% at least part of the time.
- Second, the tunable-frequency qubits we use have frequency-dependent decoherence times, and unwanted two-level systems (TLSs) resulting from, we believe, contaminants on the chip surface that can severely reduce the decoherence times over hundreds of MHz of continuous frequency range. Such frequency ranges must be avoided when bringing neighboring qubits closer in frequency to execute 2-qubit gates. TLSs are not static in time, drifting both in and out of existence and to different frequencies, so finding operating frequency ranges for all pairs of qubits is extremely challenging.
- Third, the lines delivering microwave signals to drive gates from room temperature electronics racks to the chip at 10mK at the bottom of a dilution refrigerator are not 100% reliable, and with over 200 lines for the 72 qubits, it is not uncommon for a wire to be broken.

[2]The control and target qubit of the gates are represented by • and ⊕, respectively.

[3]A Hadamard gate is represented by a box labeled H, while each of the two target qubits of a swap gate is represented by ×.

[4]Recently, IBM even conducted a Developer Challenge to this topic [2]. A description of the winning approach is available in [42].

- Fourth, the final set of electronics to generate and shape these microwaves, at time of writing, is only just now being calibrated and connected to the fridge.
- Finally, high-fidelity multi-qubit simultaneous readout using low amounts of hardware remains an outstanding challenge. Currently, most testing and calibration is done reading out just a single qubit at a time. This aspect of the system needs to be significantly improved, although it should be noted that multi-qubit readout has been achieved by our group in the past.

All of the above issues are being tackled by capable teams of people, and we are optimistic that all of them can be solved in the short to medium term. Besides that, we also hope that accomplishments and expertise from the CAD community may contribute towards the efficient solution of these issues.

4. VERIFYING QUANTUM COMPUTERS

Verification of the quantum computer [18] is an important part of building the machine. It has a lot of commonalities with the verification of conventional computers, but also has additional aspects which are unique to the nature of the quantum computer. Generally speaking, we can categorize the challenges arising from verification of quantum computers into three:

- *Logic verification*, i.e., whether the hardware design is logically equivalent to its specification.
- *Quantum verification*, i.e., whether the hardware indeed performs quantum computation logic.
- *Physical verification*, i.e., whether the hardware implementation performs as intended.

In the following sections, those issues are briefly covered.

4.1 Logic Verification

Logic verification does not pose a great challenge for the near future, as near-term machines are likely to have no more than a few hundreds of qubits, and no complex design structures are being planned for this time period. Hence, logic verification generally amounts to inspection of the design, and simple checks that all intended connectivities as, e.g., in the architecture of Fig. 2(a), are in place.

4.2 Quantum Verification

Verifying the Process is Quantum

Quantum verification is an inherent quantum computation challenge which does not have a conventional analog. Here, we are interested whether the result we get from running our machine utilizes in any real way the quantum mechanical aspects of computation. Naively, one could question why this should be interesting. After all, if we get the result we wanted, why should we care how we got it. However, deeper consideration shows that this indeed is a fundamental question to answer for at least two reasons. From a theoretical viewpoint, it is crucial to show that the added computational power promised by quantum computation can indeed be realized by a physical system. In its deepest sense, this is a test of our understanding of quantum mechanics. A preliminary to pass this test is that we can verify that the computation was quantum, and not an ingeniously fast implementation of a conventional (digital or analog) computer. Second, from a practical point of view, building a quantum computer requires tremendous amounts of efforts. If at the end, the result we get cannot be verified to be of quantum nature, than those efforts have mostly been wasted, as we could likely have built the same conventional computer which gave us the results at much lesser efforts.

Verifying the Result is Correct

Related to quantum verification is the question of validating the output of a presumably quantum (and presumably correct) computer. This is again an open challenge with deep theoretical roots. The question amounts to how do we verify that any given result of an algorithm run on a quantum computer is correct. For some classes of problems the answer is trivial. For example, for NP-hard problems, as well as for the Shor's and Grover's algorithms, the correctness of the results can be verified in polynomial runtime on a conventional computer. Similarly, results of optimization problems [8] can be verified for correctness (i.e., that they satisfy all input constraints) in polynomial time on a conventional computer, and if in addition the value of the optimization function obtained from the computer is lower than for any previously obtained solution, then we have reached our goal of finding a globally better solution. Thirdly, there may be problems that can be verified in the physical laboratory. For example, if the quantum computer, when solving a quantum chemistry problem [15], suggests a particular design for a molecule, we may go ahead and verify in the laboratory that the molecule can be synthesized as suggested, and that its parameters conform with the computation.

However, for a whole set of problems such verification cannot be readily done and in fact remains a serious open challenge. Such problems include sets of the hardest problems known. For example, sampling problems such as Boson sampling [32] are apparent candidates for realizing quantum advantage for the first time. That is because they are particularly hard to solve on conventional computers, even for relatively small problem sizes, but adjust well to quantum sampling algorithms. However, these same virtues make their verification extremely challenging. Suppose a 60-qubit quantum machine came up with some answer to the problem. How do we verify that the answer is correct? After all, the best one can do to solve an n-qubit quantum sampling problem on a conventional machine is to simulate the quantum algorithm. Hence, with today's conventional simulation limit of 56-qubits [26] (and this also on the largest supercomputers available), one cannot know whether the result obtained from a quantum machine solving a 60-qubit sampling problem is in fact correct.

Solution Approaches

There are a variety of theoretical approaches for solving the above problems [18], but none is satisfactory in practice yet. However, thinking of this from a conventional design automation point of view, the problem is not fundamentally different from a conventional verification problem. For example, when the next generation of a conventional computer becomes available (either actual hardware, or at the design stage), there is typically no computer available which can reliably verify each result that is obtained by the new design. This is because of two reasons. First, the new design (e.g., an IBM POWER 9 machine, or the Summit supercomputer), is much stronger and faster than any predecessor. Second, the shear results space is exponentially large and one cannot just verify any result which is obtained by the newly available computer. The practical way to solve this inherent issue is twofold. First, invest as much as possible already in

verification at the design stage, covering as much of the logic space as possible – in contrast to attempting to cover the results space. Second, use the most powerful machines from the last predecessor generation to run for a very long time, covering huge areas of this space. This twofold approach has proved itself time and again, and is what allows us to continue developing modern microprocessors of unprecedented complexity [31].

The conventional approach described above should be adopted in the quantum regime as well. The first part, verifying the new processor already at the design stage, falls back to logic verification. While we are not there yet (see Section 4.1), we should start building methodologies and tools to be ready to the time where logical design structures for quantum chips start becoming complex. The second part, verifying top-end chips with the newest hardware available, is a much more challenging task which still requires a significant amount of theoretical and practical research. The point is that the latest predecessor of a new quantum chip is also a quantum computer. We are currently still in the very first stages of understanding how to simulate a quantum computer by using another quantum computer [5]. Nevertheless, as this is going to be a necessity, we expect this line of research to grow rapidly, resulting in robust algorithms and subsequent methodologies for performing such quantum-by-quantum verification. As the field evolves, we expect it to lead to ways to reliably verify both aspects – of the quantumness of the process, and the correctness of results.

4.3 Physical Verification

Physical verification lies between the two extremes of quantum verification and logic verification. The main need for physical verification is the ubiquitous presence of noise in the entire lifetime of the quantum circuit. Noise events may appear at the initialization stage, at the application of any type of gate, at readout, and anywhere in between. Noise is crucial to analyze for two different reasons. First, we need to understand the overall behavior of our hardware. In particular, at the design phase, we need to use this understanding to decide between design alternatives. Second, near-term usages of quantum machines will be noise-limited and actual applications would be of so-called *Noisy Intermediate-Scale Quantum* (NISQ) type [27]. This means that any near-term application would need to know the noise behavior of the underlying hardware in order to choose the exact type and parameters of the algorithms it will run [8, 15]. In addition to those pressing issues, and looking further into the future, fault-tolerant quantum computation would be based on huge on- and off-chip infrastructure for error correction. Designing the error correction schemes would again strongly rely on a very detailed understanding of the noise processes in the hardware.

In the following, we discuss directions towards addressing those issues.

Simulation and Emulation

Two ingredients are needed in order to understand the physical hardware. First, is a set of physical noise models at the right level of abstraction. Here, we have a plethora of models, ranging from very specific, hardware-dependent noise processes, up to abstract, mathematical models specified at the same language of the quantum program [20]. Second, we need algorithms which, given the noise models and possibly a quantum circuit, can simulate or emulate the runtime behavior of the system in order to give us the insights we need.

The exact description of the hardware behavior by, e.g., state vector simulators, is lower-bounded by single-exponential complexity in the number of qubits. If built naively, runtime can easily become double exponential when describing noise in the density matrix formalism [20]. In order to manage this runtime complexity, specific algorithmic schemes can be deployed. The density-matrix formalism can be abandoned in favor of quantum trajectories formalisms [6], which reduces the double exponential runtime to a single exponential while keeping the exact functionality. The cost here is the need to write and maintain a much more complex algorithm. Further, using sophisticated data structures as presented in Section 2.1, while still conserving the single-exponential runtime, may result in huge performance gains on specific sets of problems. Lastly, making use of modern computational physics approaches such as tensor networks [25], may again drastically reduce runtime by taking hold of hidden patterns in the inputs to the simulator. All those advancements require complex software building blocks and are a major new playground where expertise in building tools for exponential problems can be a strong entry point. The conventional verification community has long dealt with exponential problems and solutions (satisfiability solvers, constraint satisfaction problems – CSP's, and binary decision diagrams to name a few), and can be of substantial help in this new domain.

Whatever the technology for dealing with quantum simulation, at the end, the problem is exponential and sooner or later we will reach the limit of applicability of any exact approach (at least if run on a conventional computer). Hence, in parallel to simulation we should start thinking about ways to gain required knowledge on the system (the combined hardware and intended applications running on it), while giving up exact and complete knowledge. This leads us to the field of quantum emulation – and in particular methods of simulating the full system at the right level of abstraction. Emulation relies heavily on our ability to understand and model the entire system at any given abstraction. This again is a skill, coupled with tools and methodologies, developed and tuned over decades in the field of hardware verification. Thus, another natural and significant conventional point of entry into the quantum verification regime.

Randomized Benchmarking and Quantum Tomography

One cannot discuss the area of physical verification without mentioning two additional technologies widely used. Quantum tomography [20] is used to verify in detail the very nature of the quantum physical system. It is practical only for very small systems. In contrast, random benchmarking [17] is a highly scalable way of characterizing and calibrating the overall behavior of a circuit – gates and qubits combined. Here, long random sequences of gates are applied in a way which leads to conventionally predictable results. Comparing these expected results with the actual outcome from the hardware provides a means for verifying the hardware and calibrating it as necessary. A main challenge here is how to choose the best set of sequences such that a particular verification task is achieved, while maintaining relatively small sets of sequences that need to be run. This challenge closely resembles the challenges behind automatic test program generation for conventional hardware verification. It appears CSP, the technology of choice for modeling and solving this task [19], can charmingly fit the quantum version as well.

5. CONCLUSIONS

We have briefly presented the exciting new field of quantum design and verification through the prism of the great sea of knowledge and methodologies existing with the conventional CAD community. We hope this overview helps bridging the gap between this and the quantum computation community. We believe there is so much to gain from close collaboration and mutual learning between the two communities and, hence, warmly advocate any steps taken towards that goal.

6. ACKNOWLEDGMENTS

We thank Gadi Aleksandrowicz, Yael Ben-Haim, Alexandru Paler, and Alwin Zulehner for their contributions to this work. We also thank all researchers and collaborators which, in the past years, worked with us on the development of the approaches which have been reviewed in this paper. We acknowledge use of IBM Q for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q team. This work has partially been supported by the European Union through the COST Action IC1405 and the Google Research Award Program.

7. REFERENCES

[1] IBM Q. https://www.research.ibm.com/ibm-q/. Accessed: 2018-08-08.

[2] QISKit Developer Challenge. https://qx-awards.mybluemix.net/#qiskitDeveloperChallengeAward. Accessed: 2018-08-08.

[3] QISKIT SDK. https://qiskit.org/. Accessed: 2018-08-08.

[4] A. J. Abhari, A. Faruque, M. J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, and F. Chong. Scaffold: Quantum programming language. Technical report, 2012.

[5] S. Bravyi, G. Smith, and J. A. Smolin. Trading classical and quantum computational resources. *Physical Review X*, 6(2):021043, 2016.

[6] T. A. Brun. A simple model of quantum trajectories. *American Journal of Physics*, 70(7):719–737, 2002.

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

[8] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[9] J. Ghosh and A. G. Fowler. A leakage-resilient approach to fault-tolerant quantum computing with superconducting elements. *Phys. Rev. A*, 91:020302(R), 2015. arXiv:1406.2404.

[10] J. Ghosh, A. G. Fowler, J. M. Martinis, and M. R. Geller. Leakage and paralysis in ancilla-assisted qubit measurement: Consequences for topological error correction in superconducting architectures. *arXiv:1306.0925*, 2013.

[11] B. Giles and P. Selinger. Exact synthesis of multiqubit Clifford+T circuits. *Phys. Rev. A*, 87(3):032332, Mar. 2013.

[12] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: a scalable quantum programming language. In *Conf. on Programming Language Design and Implementation*, pages 333–342, 2013.

[13] Y. Hirata, M. Nakanishi, S. Yamashita, and Y. Nakashima. An efficient conversion of quantum circuits to a linear nearest neighbor architecture. *Quantum Information & Computation*, 11(1&2):142–166, 2011.

[14] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter. Surface code quantum computing by lattice surgery. *New J. Phys.*, 14:123011, 2012. arXiv:1111.4022.

[15] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242, 2017.

[16] N. Khammassi, I. Ashraf, X. Fu, C. Almudever, and K. Bertels. QX: A high-performance quantum computer simulation platform. In *Design, Automation and Test in Europe*, 2017.

[17] E. Magesan, J. M. Gambetta, and J. Emerson. Scalable and robust randomized benchmarking of quantum processes. *Physical review letters*, 106(18):180504, 2011.

[18] Y. Naveh, E. Kashefi, J. R. Wootton, and K. Bertels. Theoretical and practical aspects of verification of quantum computers. In *Design, Automation and Test in Europe*, pages 721–730, 2018.

[19] Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. Marcus, and G. Shurek. Constraint-based random stimuli generation for hardware verification. *AI magazine*, 28(3):13, 2007.

[20] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[21] P. Niemann, R. Wille, and R. Drechsler. Efficient synthesis of quantum circuits implementing Clifford group operations. In *Asia and South Pacific Design Automation Conf.*, pages 483–488, 2014.

[22] P. Niemann, R. Wille, and R. Drechsler. Equivalence checking in multi-level quantum systems. In *Int'l Conf. of Reversible Computation*, pages 201–215, 2014.

[23] P. Niemann, R. Wille, and R. Drechsler. Improved synthesis of Clifford+T quantum functionality. *Design, Automation and Test in Europe*, 2018.

[24] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler. QMDDs: Efficient quantum function representation and manipulation. *IEEE Trans. on CAD*, 35(1):86–99, 2016.

[25] R. Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[26] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff. Breaking the 49-qubit barrier in the simulation of quantum circuits. *arXiv preprint arXiv:1710.05867*, 2017.

[27] J. Preskill. Quantum computing in the NISQ era and beyond. *arXiv preprint arXiv:1801.00862*, 2018.

[28] M. Saeedi, R. Wille, and R. Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing*, 2010.

[29] A. Shafaei, M. Saeedi, and M. Pedram. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *Design Automation Conf.*, pages 41–46, 2013.

[30] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Foundations of Computer Science*, pages 124–134, 1994.

[31] B. Sinharoy, J. Van Norstrand, R. J. Eickemeyer, H. Q. Le, J. Leenstra, D. Q. Nguyen, B. Konigsburg, K. Ward, M. Brown, J. E. Moreira, et al. IBM POWER8 processor core microarchitecture. *IBM Journal of Research and Development*, 59(1):2–1, 2015.

[32] J. B. Spring, B. J. Metcalf, P. C. Humphreys, W. S. Kolthammer, X.-M. Jin, M. Barbieri, A. Datta, N. Thomas-Peter, N. K. Langford, D. Kundys, et al. Boson sampling on a photonic chip. *Science*, page 1231692, 2012.

[33] D. S. Steiger, T. Häner, and M. Troyer. ProjectQ: an open source software framework for quantum computing. *arXiv preprint arXiv:1612.08091*, 2018.

[34] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003.

[35] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo. An XQDD-based verification method for quantum circuits. *IEICE Transactions*, 91-A(2):584–594, 2008.

[36] R. Wille, D. Große, D. M. Miller, and R. Drechsler. Equivalence checking of reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 324–330, 2009.

[37] R. Wille, O. Keszöcze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler. Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits. In *Asia and South Pacific Design Automation Conf.*, pages 292–297, 2016.

[38] R. Wille, A. Lye, and R. Drechsler. Exact reordering of circuit lines for nearest neighbor quantum architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 33(12):1818–1831, 2014.

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

[40] A. Zulehner and R. Wille. Advanced simulation of quantum computations. *IEEE Trans. on CAD*, 2018. Code available at: http://iic.jku.at/eda/research/quantum_simulation/.

[41] A. Zulehner and R. Wille. One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic. *IEEE Trans. on CAD*, 37(5):996–1008, 2018.

[42] A. Zulehner and R. Wille. Compiling SU(4) quantum circuits to IBM QX architectures. 2019.