Benchmarking ZX-Calculus Circuit Optimization Against Qiskit Transpilation

Lia Yeh[¶†], Emma (Maitreyee) Dasgupta[+†], Erick Winston[†]
[¶]University of California, Santa Barbara; [+]The University of Chicago; [†]IBM Q

Abstract

Quantum hardware and algorithms researchers alike have expressed the critical need for and shortage of expertise in quantum programming languages, compilation, and computer architecture. We design, implement, and benchmark quantum compilation and visualization in PyZX[1], a Python tool for the ZX-calculus, applied to Qiskit[2], the most widely-used quantum programming language. We demonstrate correct circuit optimization by our Qiskit transpiler pass on 1000 two qubit Randomized Benchmarking circuits, reducing gate count of 29% of the circuits further than Qiskit's default transpiler.

Section I: Problem and Motivation

Quantum computers can efficiently simulate quantum systems, phenomena impossible to efficiently simulate using classical computers[3]. The computational capabilities of experimentally realized quantum computers, across many quantum hardware platforms, has scaled at an exponential rate in recent years. Following quantum supremacy[4] (the demonstration and confirmation of correctness of a quantum algorithm execution vastly outperforming the best-known runnable classical algorithm), quantum computing's next challenge is to run useful quantum algorithms on near-term quantum devices, as in Figure 1. The vast applications for realizable quantum computation include material science, quantum chemistry, quantum gravity, machine learning, cryptography, drug discovery, optimization, and quantum sensing[5].

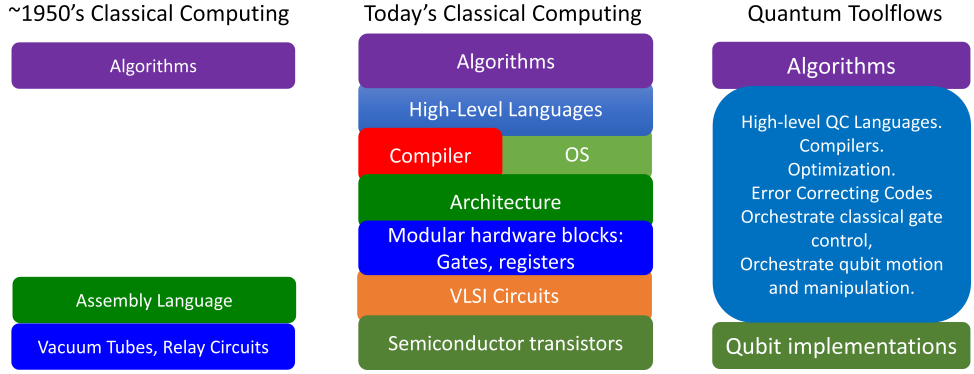

Figure 1: *Reprinted from[6]. What would the organization of quantum toolflows look like? As a technology decades from its full theorized potential, the challenge of full-stack quantum computation is in inventing layers upon layers of abstraction bridging quantum algorithms and scalable quantum hardware.*

Any quantum algorithm, no matter how theoretically sound, requires efficient and correct implementation to be useful. In Section III, we explain why the ZX-calculus helps reason about quantum information more effectively than in the quantum circuit representation, exemplified in Figure 2.

This work realizes the first implementation of the ZX-calculus with the power to express classical registers and measurement operations, critically enabling program execution on quantum hardware.

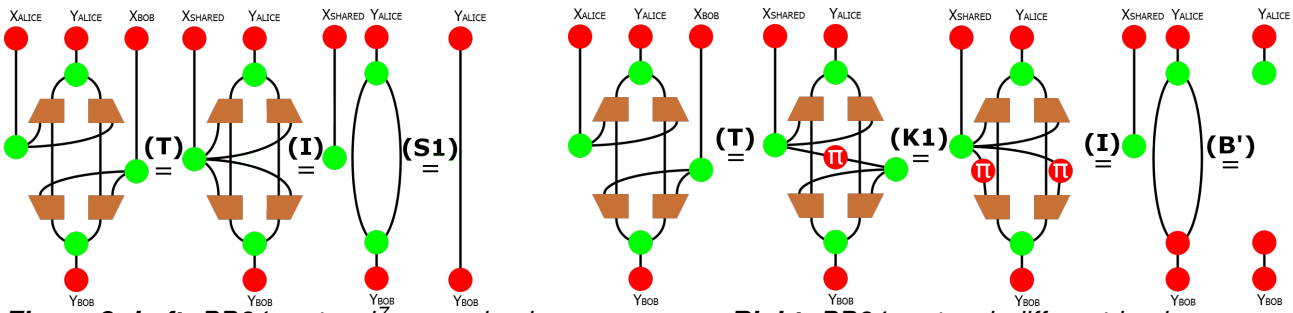

Figure 2: Left: *BB84 protocol[7], same-basis case.* ***Right:*** *BB84 protocol, different-basis case. The above diagrams by Yeh and Burke[8] show succinct simplification of the most widely studied quantum cryptography protocol using only the rules of the ZX-calculus[9], as opposed to using matrix multiplication.*

Section II: Background and Related Work

This work enhances quantum circuit optimization in the quantum programming language Qiskit[2] and the quantum assembly language openQASM[10]. With the monthly number of IBM Q users in the millions, this exponential growth in demand drives need for open-source quantum frameworks and toolchains[11].

The ZX-calculus

The ZX-calculus is a graphical language with monoidal category theory foundations. It represents quantum circuits as graphs transformable via topological rules and extractable to quantum circuit form. It is also called the red-green calculus because its red and green dots represent the Z and X bases, respectively. Coecke and Duncan invented the ZX-calculus[12], while Kissinger and van de Wetering are continuously innovating on PyZX, a Python implementation of the ZX-calculus[1].

The ZX-calculus is one of few circuit optimization tools developed to date for which advantages to both today's Noisy Intermediate-Scale Quantum era[1] and a hypothesized future Fault-Tolerant Quantum Computing era[13] compilation have been demonstrated. Given the insurmountable task of characterizing and suppressing all possible error sources (i.e. coherence errors, gate errors, state preparation and measurement errors)[14], it is highly desirable to deploy software facilitating transition between these two eras.

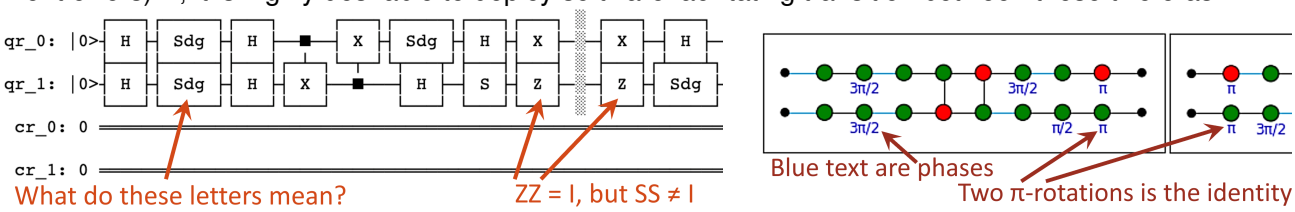

Figure 3: Left: *Quantum circuit drawn in Qiskit.* ***Right:*** *ZX-calculus graph drawn in PyZX. The default representation of quantum circuits today (left) hasn't changed from in quantum information textbooks written decades before programmable quantum computers, riddled with absurd jumbles of letters equating circuits like Z = HXH, nonsensical to anyone who hasn't memorized the matrix for each letter.*

Section III: Uniqueness of the Approach

Present-day quantum programming languages offer little in the way of abstraction. For instance, programs in IBM's Qiskit, the most widely-used quantum PL to date, create a Qiskit QuantumCircuit at the same level in the abstraction hierarchy as openQASM, Qiskit's quantum assembly language specification. IBM Q has recently released a new Qiskit feature, Pulse, allowing user control closer to the hardware layer[15]. Transpiler passes within Qiskit optimize quantum circuits in directed acyclic graph (DAG) representation. However, this restriction to only one to three PL layers, often semantically degenerate, available to the programmer inherently limits quantum compilation and circuit optimization in today's quantum PLs.

The two undergraduate authors proposed, designed, implemented, and benchmarked a Qiskit transpiler pass using PyZX, enabling use of the ZX-calculus to its full potential as a higher level abstraction layer.

In addition to circuit optimization, this work enables step-by-step visualization of quantum circuit optimization techniques side-by-side with its PyZX circuit representation, and vice versa, as in Figure 3. This may be used to give insight into quantum information theory justifications for circuit optimization techniques, as well as help develop better quantum circuit optimization techniques. From a quantum information perspective, the ZX-calculus has several advantages over the quantum circuit representation: 1) As a tensor network capable of expressing quantum logic, the ZX-calculus is well-positioned to leverage centuries of research on graph theory optimization techniques. 2) Edges in ZX-calculus graphs are undirected, and thus more natural for quantum computation (which must be reversible) than the DAG form (i.e. the form used during transpilation) of Qiskit QuantumCircuits. 3) From an information theory standpoint, the colored generators and phase labels in the ZX-calculus and its topological rules are more elegant, expressive, and powerful than the clumsy letters requiring memorization of the quantum circuit representation. In pushing this frontier where quantum information and programming languages intersect, the development of ZX-calculus related tooling enhances capabilities to intuit about cutting-edge quantum compiler optimizations.

Completeness

PyZX is a framework for the ZX-calculus in Python, a Turing-complete programming language. However, it does not natively support several operations critical to a fully-programmable quantum computer. Figure 4 delineates this work's main contributions to expanding PyZX's capacity to express such operations.

The ZX-calculus is complete for stabilizer quantum theory, i.e. restricted to the Clifford gate set, which is efficiently simulatable by a classical computer according to the Gottesman-Knill theorem[16]. There exist several more involved variants of the ZX-calculus, some of which have been proven complete for universal quantum computation, achievable using a Clifford + T gate set[17].

Section IV: Results and Contributions

Implementation

This work's functionalities are delineated in Figure 4. Figure 5 illustrates example circuit transformations.

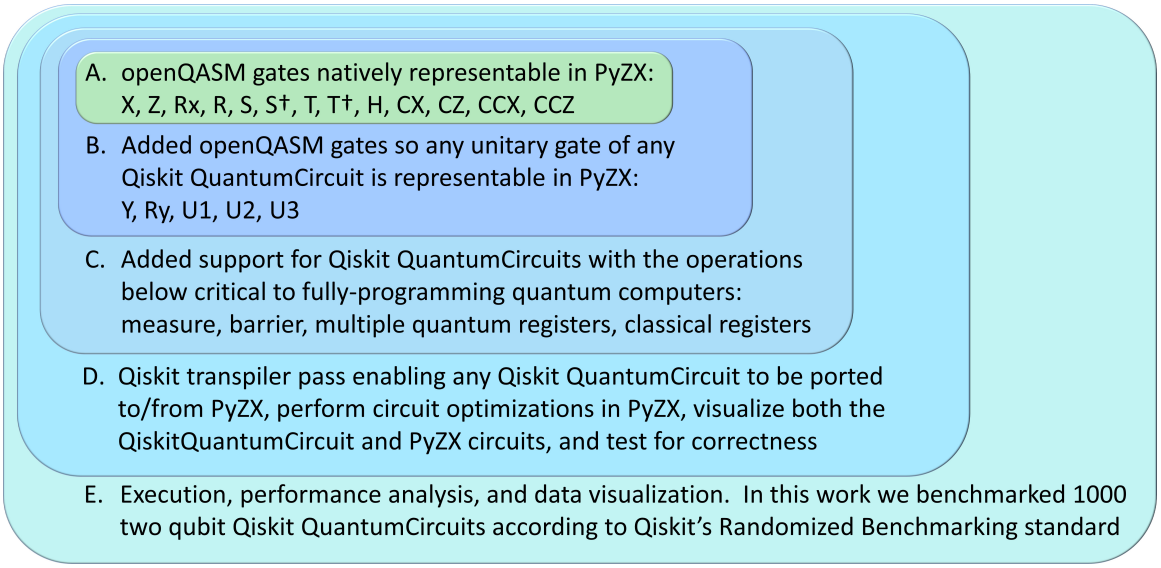

Figure 4: *A. lists openQASM gates supported by PyZX. This work implements features B., C., D., and E., smoothly integrated into our forks of PyZX and Qiskit Terra. Our transpiler pass can be applied to any Qiskit QuantumCircuit with only two lines of code: one to import our pass from PyZX, and one to apply it.*

Benchmarking

The goal of a transpiler pass is to, with non-neglible probability and reasonable runtime, optimize a quantum circuit without violating correctness up to a global phase. Dozens of transpiler passes in Qiskit feature many cutting-edge circuit optimization strategies in the literature, implemented by estimably dozens of IBM Q software engineers and hundreds of open-source contributors. These passes can be consecutively applied, but doing so increases probability of bugs and does not necessarily result in further optimization. As it is easy to apply multiple passes and choose the most optimal of their outcomes, the results in Figure 6 effectively mean our integration of PyZX with Qiskit reduces gate count for 29% of the quantum circuits we randomly generated. This strongly implies that there exists a sizable percentage of all quantum circuits of interest, for which PyZX has significant advantage in optimizing over competing methods.

Correctness

Although we successfully validate correctness (up to a global phase) of our transpiler pass's optimizations for each of the thousands of two qubit circuits executed and described in this paper, this naive approach of verification utilizing matrix multiplication is untenable. To name one obstacle to scaling, the size of these matrices scales exponentially with the number of qubits in the circuit. This lack of methodologies and tools to verify correctness of quantum programs is concerning[18]. Not only is programmer error common due to the esoteric nature of quantum programming, but quantum programming languages are rife with issues. This is not solely due to programmer error — in carrying out this work and another work awarded by the ACM Student Research Competition, the two undergraduate authors of both works discovered around a dozen bugs and mistakes in Qiskit's and PyZX's codebase, example algorithm implementations, and published specifications[19]. In particular, one positive auxiliary outcome of this work was that Qiskit's maintainers addressed our report of Qiskit's openQASM parser implementation lacking the functionality to parse custom unitarity gates generated by its own circuits.

The PyZX compiler verifies its circuit rewrites utilizing translation validation. Grounded in the equational theory of the ZX-calculus, this work has the added functionality of making any quantum circuit in Qiskit simultaneously more robust against programming errors and more easily visualized and reasoned about.

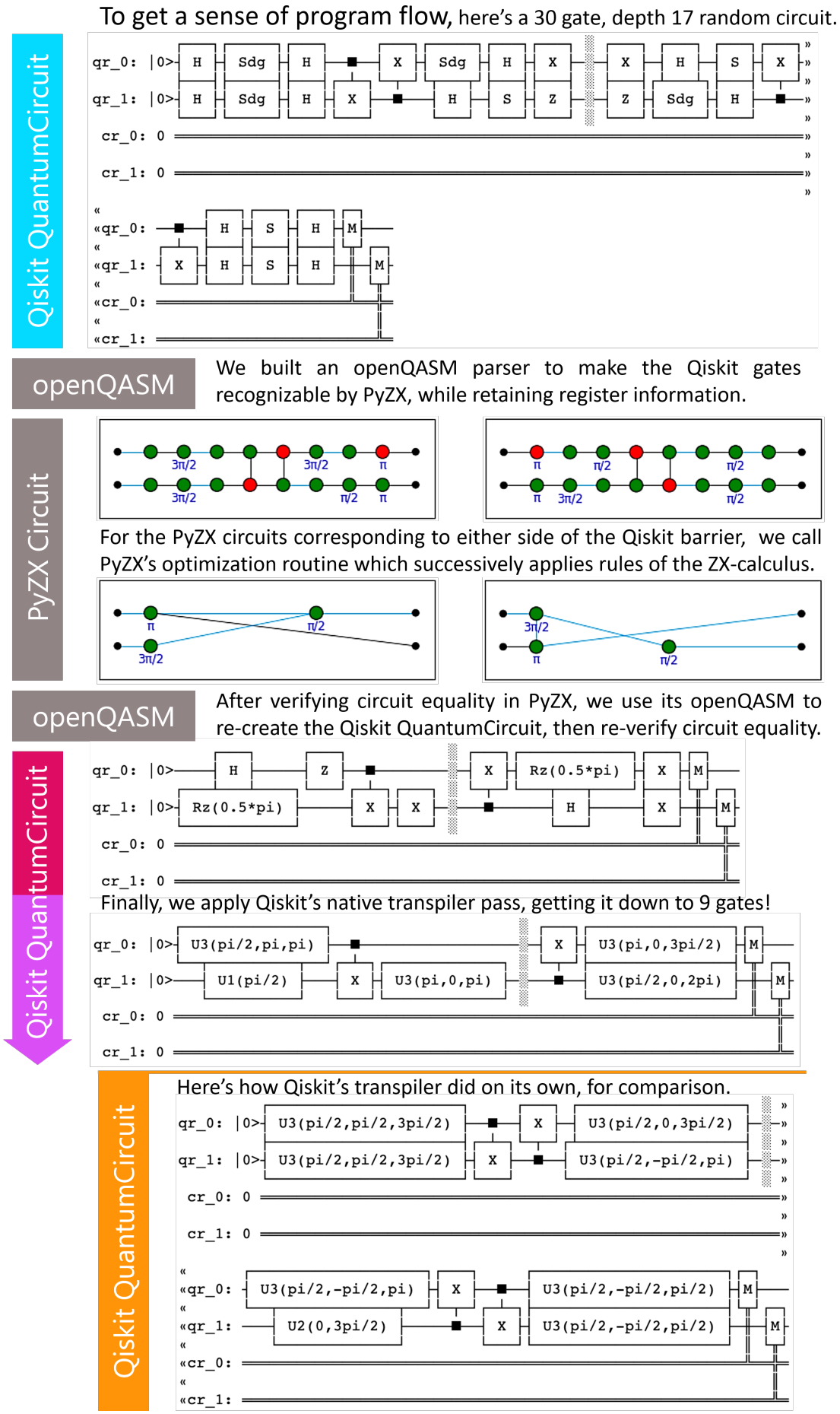

Figure 5: *Our transpilation of a Randomized Benchmarking Qiskit QuantumCircuit, converted to openQASM, PyZX circuit, openQASM, and back. Here PyZX outperformed the native Qiskit transpiler; furthermore, PyZX is compatible with the native Qiskit transpiler, resulting in additional optimization.*

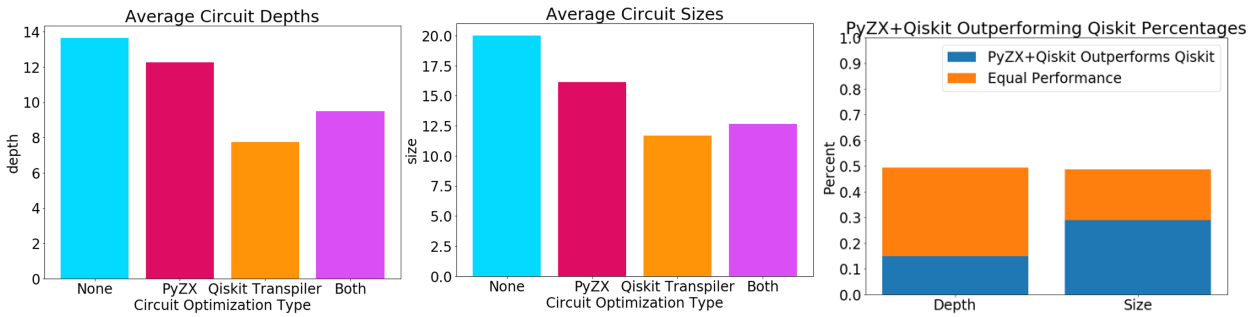

Figure 6: Left and Center: *Comparison of 1000 Randomized Benchmarking circuits under 4 conditions:* ***1:*** *No optimizations,* ***2:*** *Optimize via the ZX-calculus,* ***3:*** *Optimize via Qiskit's transpiler, and* ***4:*** *Both conditions 2. and 3. For 29% of the circuits, our PyZX transpiler pass back-to-back with Qiskit's transpiler outperformed Qiskit's transpiler at reducing circuit size. This is a non-trivial result because it is not necessarily the case that chaining two transpiler passes yields a better result than either alone.*
Right: *For the average randomized Qiskit QuantumCircuit, Qiskit transpilation outperforms PyZX. However, this discrepancy in size and depth is largely explained by the fact that unlike Qiskit, PyZX does not presently support combining one or more successive gates into a single-qubit custom unitary.*

Runtime

The objective of quantum circuit optimization is to reduce the number of quantum gates necessary to implement the desired quantum algorithm. Accordingly, the primary criterion for success is how effectively the gate count is reduced, as minimizing gate count will directly reduce the accumulation of gate-based errors during a quantum computation. Therefore, the (classical) runtime of the transpiler pass is considered to be of far less importance compared to the (quantum) runtime of the quantum computation. We note that runtime did not impede our circuit generation, transpiler pass, and benchmarking on a timescale of seconds, on a single core of a 2.40 GHz processor on the corresponding author's rather typical personal laptop. In fact, the runtime of generating the random circuits dominated that of our transpiler pass. Furthermore, we were able to apply our transpiler pass to other nontrivial quantum circuits standardized for benchmarking, up to on the order of ten thousand gates, without runtime being an issue.

Section V: Ongoing Work

The corresponding author of this work is investigating foreseeable applications of the ZX-calculus across the quantum computing stack:

1: Noisy Intermediate-Scale Quantum- and Fault-Tolerant Quantum Computing-era circuit optimization
2: Quantum hardware agility: circuit routing and gateset conversion
3: As an alternate representation of quantum computation to the quantum circuit model
4: Quantum verification and quantum cryptography protocols

Section VI: Conclusion

Today, quantum computers have prohibitively high error rates, and quantum programming languages have countless bugs. To run useful quantum programs theorized to solve many problems unsolvable by classical computers, it is crucial to optimally compile quantum circuits. Not only does reducing circuit size improve runtime of quantum programs, but it can make the difference between whether quantum program results are too error-riddled to interpret.

The significance of this work is the two undergraduate authors' proposal, design, implementation, and benchmarking of our open-source transpiler pass in Qiskit, the most widely-used quantum programming language, achieving the end of reporting significant probability (29%) of outperforming Qiskit's native transpiler on Randomized Benchmarking circuits. Correctness up to a global phase was verified for 100% of the thousands of two qubit circuits executed and described in this work. This verification was performed in both PyZX (on each subcircuit) and in Qiskit (on the circuit in its entirety), utilizing built-in methods to compare tensors.

On balance, a more comprehensive analysis of the ZX-calculus applied to simplifying circuits informs ongoing endeavors to develop a more optimal axiomatization of universal quantum computation. To facilitate the transition from Noisy Intermediate-Scale Quantum to fault-tolerant quantum computation, we conclude the ZX-calculus is a powerful tool for diagrammatic reasoning and complexity reduction of quantum circuits.

References

[1] A. Kissinger and J. van de Wetering, "Pyzx: Large scale automated diagrammatic reasoning," in *QPL 2019*, Jun 2019.

[2] H. Abraham, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, G. Alexandrowics, E. Arbel, A. Asfaw, C. Azaustre, AzizNgoueya, P. Barkoutsos, G. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, L. S. Bishop, S. Bolos, S. Bosch, S. Bravyi, D. Bucher, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, R. Chen, J. M. Chow, C. Claus, C. Clauss, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Dartiailh, DavideFrr, A. R. Davila, D. Ding, J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, A. Frisch, A. Fuhrer, M. GEORGE, J. Gacon, Gadi, B. G. Gago, J. M. Gambetta, A. Gammanpila, L. Garcia, S. Garion, J. Gomez-Mosquera, S. de la Puente González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, M. Haglund, I. Haide, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, A. Javadi, A. Javadi-Abhari, Jessica, K. Johns, T. Kachmann, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, S. King, Knabberjoe, A. Kovyrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, G. Kus, R. LaRose, R. Lambert, J. Latone, S. Lawrence, D. Liu, P. Liu, Y. Maeng, A. Malyshev, J. Marecek, M. Marques, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, D. McPherson, S. Meesala, M. Mevissen, A. Mezzacapo, R. Midha, Z. Minev, A. Mitchell, N. Moll, M. D. Mooring, R. Morales, N. Moran, P. Murali, J. Müggenburg, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, Y. Naveh, P. Neuweiler, P. Niroula, H. Norlen, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, S. Oud, D. Padilha, H. Paik, S. Perriello, A. Phan, F. Piro, M. Pistoia, A. Pozas-iKerstjens, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, D. M. Rodríguez, RohithKarur, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, H. Sargsyan, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, J. Schwarm, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock, Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, SooluThomas, D. Steenken, M. Stypulkoski, J. Suen, K. J. Sung, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, D. Vogt-Lee, C. Vuillot, J. Weaver, R. Wieczorek, J. A. Wildstrom, R. Wille, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, D. Yeralin, R. Young, J. Yu, C. Zachow, L. Zdanski, C. Zoufal, Zoufalc, a matsuo, azulehner, bcamorrison, brandhsn, chlorophyll zz, dan1pal, dime10, drholmie, elfrocampeador, enavarro51, faisaldebouni, fanizzamarco, gadial, gruu, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yelojakit, and yotamvakninibm, "Qiskit: An open-source framework for quantum computing," 2019.

[3] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, p. 467–488, Jun 1982.

[4] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[5] "National strategic overview for quantum information science," Sep 2018.

[6] M. Martonosi and M. Roetteler, "Next steps in quantum computing: Computer science's role," *Computing Research Association CCC Catalyst*, Nov 2018.

[7] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Theoretical Computer Science*, vol. 560, pp. 7 – 11, 2014. Theoretical Aspects of Quantum Cryptography – celebrating 30 years of BB84.

[8] L. Yeh and K. Burke, "A short guide to the zx-calculus," Mar 2018.

[9] B. Coecke, Q. Wang, B. Wang, Y. Wang, and Q. Zhang, "Graphical calculus for quantum key distribution (extended abstract)," *Electronic Notes in Theoretical Computer Science*, vol. 270, no. 2, pp. 231 – 249, 2011. Proceedings of the 6th International Workshop on Quantum Physics and Logic (QPL 2009).

[10] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," 2017.

[11] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, "A case for multi-programming quantum computers," *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, Oct 2019.

[12] B. Coecke and R. Duncan, "Interacting quantum observables: categorical algebra and diagrammatics," *New Journal of Physics*, vol. 13, p. 043016, 2011.

[13] A. Kissinger and J. van de Wetering, "Reducing T-count with the ZX-calculus," *arXiv preprint arXiv:1903.10477*, 2019.

[14] S. S. Tannu and M. Qureshi, "Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes," Jun 2019.

[15] D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Córcoles, D. Egger, S. Filipp, J. Gomez, M. Hush, A. Javadi-Abhari, D. Moreda, P. Nation, B. Paulovicks, E. Winston, C. J. Wood, J. Wootton, and J. M. Gambetta, "Qiskit backend specifications for openqasm and openpulse experiments," 2018.

[16] D. Gottesman, "The heisenberg representation of quantum computers, talk at," in *International Conference on Group Theoretic Methods in Physics*, p. 9807006, 1998.

[17] P. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, "A new universal and fault-tolerant quantum basis," *Information Processing Letters*, vol. 75, no. 3, pp. 101 – 107, 2000.

[18] Y. Huang and M. Martonosi, "Statistical assertions for validating patterns and finding bugs in quantum programs," in *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, (New York, NY, USA), p. 541–553, Association for Computing Machinery, 2019.

[19] M. E. Dasgupta, L. Yeh, Y. Huang, M. Martonosi, and S. Lyon, "Statistical assertions for debugging in qiskit," in *ACM Student Research Competition*, Association for Computing Machinery, Oct 2019.