Benchmarks Summary

Medina Bandic, Nikiforos Paraskevopoulos

July 2020

1 Introduction

We present **qbench**, a categorized and as of now the most comprehensive set of quantum algorithms (benchmarks) from various sources and platforms and in different quantum programming languages. Most of the currently existing and used quantum algorithms, synthetically generated and application-based circuits are included in this collection and classified based on different criteria. This benchmark collection has benchmarks from various sources cataloged in folders based on how they are implemented (e.g., based on real algorithm, random, application-based), the language they are written in, and their size. The set also contains various scripts for translating circuits from one language to another and circuit interaction graphs. We hope that this collection will be useful for testing new quantum processors, updated regularly by the research community to keep up with the new technologies, compilers, programming languages and most importantly applications, and eliminate the over-the-top amount of benchmark sources. We are also hoping that this circuits set will be used for benchmarking quantum computing systems as well as parts of it, such as compilation techniques, and to encourage the development of new better benchmarks and their success metrics.

Quantum algorithms (hardware-agnostic circuits) used as benchmarks for analyzing the performance of the mappers are not representative as they are in most cases reversible circuits that will not provide any computational advantage compared to their classical counterparts. In addition, they are usually only profiled in terms of a number of gates, circuit depth, percentage of two-qubit gates and number of interactions between qubits pairs (this latter is used for deriving an optimal placement of qubits). By having a more in-depth profiling of the quantum algorithms in which characteristics of the interaction graphs (i.e. how many times each pair of qubits interact and how those interactions are distributed among qubits and in time) and of the quantum instruction dependency graph (i.e. identifying clusters of operations) can be beneficial for obtaining optimal mapping solutions. These variables derived from the algorithm profiling will also be essential for developing application-specific quantum systems.

2 Success criteria for circuits

As it could be seen from examples above there are 2 types of success criteria: per circuit and after many repetitions. The latter one is usually the average of single-circuit ones, but that is not always the case. The "test" for the benchmark is sometimes considered successful if all circuits were run successfully, sometimes if over 90% of them and sometimes if average quantitative success is achieved.

Success of some circuit can be defined as running circuit on processor with the same result as on the perfect processor with no errors. This definition is however too strict and almost impossible, which is why we need some other criteria. Some of the success criteria include:

- The probability of correct outcome of circuit is greater than some threshold with high statistical confidence. Applies to definite-outcome circuits.
- Heavy output probability is similar to previous one, but it also applies to some indefinite-outcome circuits. Used for QV circuits. Tests decomposition ability and compilation strategy. Output is considered 'heavy' if its probability of success is higher than median. It is tested if there are over 2/3 of heavy outputs after performing many repetitions of circuits with high statistical confidence.
- Distribution of outcome of the circuit within a specific distance from ideal distribution with high stat. confidence. Applies to all circuits.
- The coarse-grained outcome distribution is close to ideal with respect to some specific coarse-graining. It includes heavy output probability.
- Probability of output that is within specific Hamming distance of correct outcomes is greater than some threshold.
- Empirical cross-entropy between outcome frequencies and predicted probabilities is lower than some threshold. Used for Google's proposal for quantum supremacy experiment.
- $L1$ - norm distance: Only for small number of qubits.The most exact and correct method which compares full ideal and noisy output distributions. High memory requirement.

3 Real Benchmarks

According to Nielsen and Chuang there are 3 groups of currently used real benchmarks that can make some advantage for quantum in comparison to classical devices:

- Algorithms based on Fourier Transform: Schor's, QFT, Deutsch-Jozsa, ripple adders ...
- Search algorithms: Grover's...
- Algorithms simulating real-life applications: Quantum chemistry, quantum mechanics and quantum machine learning algorithms...

Table 1: Real Algorithms with Speed-up

Name	Speed-up	References	Code Ref.	Language	Scalable
Grover's Search Algorithm	quadratic	[40, 39, 1, 2, 31]	[19]	Qiskit Jupiter	Yes
			[55]	C++	Yes
			[45]	Openql Python	Yes
			[43]	CQASM	No
			[14]	Cirq Python	Yes
			[47]	OpenQl Python	No

Continued on next page

Table 1 – continued from previous page

Name	Speed-up	References	Code Ref.	Language	Scalable
			[51, 9]	Riggeti Python	Yes
			[34]	Q# / Jupyter	Yes
Square Root	exponential	[23, 39, 60, 25, 61]	[42]	OpenQl Python	No
			[13]	.scaffold	Yes
QFT	exponential	[38, 1, 37, 40, 2, 23, 60, 57, 4]	[55]	C++	Yes
			[44]	CQASM	No
			[14]	Cirq Python	No
			[17]	OpenQASM	No
			[13]	.scaffold	Yes
			[51, 9]	Riggeti Python	Yes
			[34]	Q# / Jupyter	Yes
Shor's Factoring Algorithm	exponential	[23, 2, 25]	[19]	Qiskit Jupiter	Yes
			[14]	Cirq Python	Yes
			[13]	.scaffold	Yes
			[42]	OpenQl Python	No
Deutsch-Jozsa	exponential		[19]	Qiskit Jupiter	Yes
			[14]	Cirq Python	No
			[47]	OpenQl Python	No
			[51, 9]	Riggeti Python / Jupyter	yes
			[34]	Q# / Jupyter	Yes

Continued on next page

Table 1 – continued from previous page

Name	Speed-up	References	Code Ref.	Language	Scalable
Jordan Gradient	exponential	[24]	[51, 9]	Riggeti Python / Jupyter	Yes
QAOA	Still negotiable	[15, 12]	[14]	Python	Yes
			[19]	Python	Yes
			[51, 9]	Riggeti Python	Yes
			[48]	cQASM	No
Hidden Shift Algorithm (uses QFT)	exponential	[8, 40, 38, 7]	[14]	Cirq Python	Yes
Application-based circuits		[49, 5, 11]	[49, 47]	OpenQl Python	No
			[6]	OpenQASM	No
			[18]	Qiskit Python	No
			[33]	Q#	No

Table 2: Real Algorithms with NO Speed-up

Name	Parametrizable	References	Code Ref.	Language
Bernstein-Vazirani	YES	[38, 40, 39]	[19]	Qiskit Jupiter
			[14]	Python
			[47]	OpenQl Python
			[17]	OpenQASM
			[51, 9]	Riggeti Python
Supremacy Circuits	YES	[40]	[14]	Python
Peres Gate	PROBABLY NO	[40, 38]	[46]	.pla
Adder	YES	[40, 38, 56, 25]	[17]	OpenQASM
Ising model	YES	[39, 38, 61]	[13]	.scaffold
			[51, 9]	Riggeti Jupyter
Fredkin (controlled-swap)	NO	[40, 32]	[46]	.pla
Toffoli (Gate)	YES	[32, 40, 38, 31]	[46]	.pla
VQE	YES	[3, 23]	[19]	Qiskit Jupiter
			[13]	.scaffold
			[51, 9]	Riggeti Python
			[17]	openQASM

RevLib benchmarks are within the domain of reversible and quantum circuit design. References: [25, 61, 62, 59, 22, 28, 29, 58, 60, 57]. RevLib benchmarks: [46].

Qlib Benchmarks: [42, 30]

Most common reversible circuits code in OpenQl Python can be found in [42].

OpenQASM version of all benchmarks classified in 3 groups: large-, medium- and small-scale can be found in [26, 27].

All the codes written in C++ can be directly run through Quantum Inspire, Rigetti Forest, IBM Q Experience or OpenQl by using LibKet tool[36].

4 Synthetic Benchmarks

Synthetic benchmarks represent the group of randomly generated circuits, which provide is bigger variety in terms of their parameters (e.g. number of qubits, gates, two-qubit gate ratio). Some examples include volumetric benchmarks already described in previous section of this document. The random circuits however, can be generated in various ways. Summary of different random circuits used so far in previous works can be found in the table below:

Table 3: Synthetic circuits

Name	**Method for generating circuits**	**References**	**Code Ref**	**Language**
Quantum volume model circuit	Random square circuits with the same width and depth. Trying to use up all qubits per layer. Using any two-qubit gate from SU(4).	[10]	[21] [20] [14]	OpenQASM Python Python / Jupyter
Volumetric benchmarks	Family of random circuits that includes not just QV circuit but also circuits with different width and depth, as well as periodic circuits, shallow complex circuits and application-based circuits.	[5, 35]	—-	—-
'Realistic' random circuits	Generated random circuits in a way that they see more realistic, with assigning different probabilities on edges for interactions between qubits.	[2]	[14]	Python
Random circuits 1 from reinforcement learning paper	One layer random circuit which is engaging all the qubits into two-qubit gates. (max. circ. density)E.g. if there are 16 qubits there will be one layer of 8 two-qubit gates performed on different pairs of qubits.	[16, 41]	—	—

Continued on next page

Table 3 – Continued from previous page

Name	Method for generating circuits	References	Code Ref	Language
Random circuits 2 from reinforcement learning paper	More realistic version of random circuit where they choose qubits at random as well (they do not force engaging all qubits per layer into two-qubit gates), what therefore leads to higher circuit depth. They played with circuit density parameter and compared results instead of using the max. one like in previous case.	[16, 41]	—	—
Uniformly generated random circuits	Random circuits are generated uniformly random (considering used qubits and and single- and two-qubit gates). Two-qubit gates are chosen from already predefined set.	[50]	[54] [6]	OpenQl Python OpenQASM MCHECK
QUEKO	Depth-optimal generated circuits based on generic input quantum circuit and device graph.	[52]	[53]	OpenQASM

5 Next steps

5.1 Plan for converting the rest of the q. algorithms

The goal is to have all algorithms either in CQASM or in OpenQl. To simplify the process we should convert the algorithms we don't have in one of those 2 languages in the next way:

1) In case we have OpenQASM version of code we can convert it directly to CQASM by using our OpenQASM->CQASM translator. 2) In case we have scalable Python (from any framework) version of code we should either find the way to convert it to OpenQl Python version. 3) In case we don't have the scalable version of code we should just convert the non-scalable one in the same way as described in previous 2 steps.

All converted files are available at [42].

6 Metrics for interaction graph characterisation

In this paragraph we listed all the graph metrics relevant for characterizing interaction graphs relevant to mapping problem.

1)Standard metrics:

- # of qubits
- # of gates
- percentage of two-qubit gates

2)Hopcount related metrics:

- Avg. hopcount (shortest path) in graph - global, the smaller the better
- Closeness (avg. distance from each node to other nodes in hopcounts) - local, the smaller the better
- Diameter (longest shortest path in graph, sparsity) - global, the bigger the better

3)Degree related metrics:

- Degree (num. of nodes to which some node is connected) - local
- Avg. degree - global, the smaller the better
- Min. and max. degree - global, the smaller the better, we should compare max. min degree values to see uniformity of graph
- Degree distribution

4)Clustering related metrics:

- Clique/n-clique (subset of nodes such that they are all connected / connected in n hops) - global
- Clustering coefficient (Measuring cliquishness of neighbourhood. (betw. 0 and 1, 1 means fully-connected graph - local or global, the lower the better)

5)Adjacency matrix related metrics:

- Weight distribution
- Min. and max. weight
- Std. deviation
- Variance

After we characterize interaction graphs based on this metric the plan is to characterize coupling graphs (connectivity of physical qubits in device), compare and use these both information for improvements of the mapper. One of the mapping methods we are considering is involving graph partitioning

Here we have some metrics to be used later:

- Persistence (smallest of links whose removal increases diameter or disconnects the graph) - for graph partitioning
- Central point of dominance (Max betweenness of any point in the graph, 0 for complete and 1 for star graphs (with one central node). Betweennes is how many shortest paths go through some node or link) - for improving mapping method -> choosing the best initial placement
- Distortion (after presenting graph as a spanning tree check how many extra hops we need between nodes that were connected in original graph) ?
- Clique,clustering coefficient - for graph partitioning
- Vertex/edge connectivity and other reliability-related metrics(respective smallest number of nodes and links whose removal disconnects graph) - for graph partitioning

- Giant component (maximal subgraph of directed graph such that for every pair of nodes there is directed path PA->B and PB->A for hopcount k) - for directed coupling graph analysis.

References

[1] Understanding quantum control processor capabilities and limitations through circuit characterization, 2020.

[2] Jonathan M Baker, Casey Duckering, Alexander Hoover, and Frederic T Chong. Time-sliced quantum circuit partitioning for modular architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 98–107, 2020.

[3] Panagiotis Kl Barkoutsos, Jerome F Gonthier, Igor Sokolov, Nikolaj Moll, Gian Salis, Andreas Fuhrer, Marc Ganzhorn, Daniel J Egger, Matthias Troyer, Antonio Mezzacapo, et al. Quantum algorithms for electronic structure calculations: Particle-hole hamiltonian and optimized wave-function expansions. *Physical Review A*, 98(2):022322, 2018.

[4] Debjyoti Bhattacharjee, Abdullah Ash Saki, Mahabubul Alam, Anupam Chattopadhyay, and Swaroop Ghosh. Muqut: Multi-constraint quantum circuit mapping on noisy intermediate-scale quantum computers. *arXiv preprint arXiv:1911.08559*, 2019.

[5] Robin Blume-Kohout and Kevin C Young. A volumetric framework for quantum computer benchmarks. *arXiv preprint arXiv:1904.05546*, 2019.

[6] Cambridge. tket benchmaking. https://github.com/CQCL/tket_benchmarking, 2020.

[7] Andrew M Childs and Wim Van Dam. Quantum algorithm for a generalized hidden shift problem. *arXiv preprint quant-ph/0507190*, 2005.

[8] Andrew M. Childs and Wim van Dam. Quantum algorithm for a generalized hidden shift problem. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1225–1232, USA, 2007. Society for Industrial and Applied Mathematics.

[9] Rigetti Computing. Rigetti - quantum algorithms built using pyquil. https://github.com/rigetti/grove, 2019.

[10] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. Validating quantum computers using randomized model circuits. *arXiv:1811.12926*, 2018.

[11] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018.

[12] Minh Do, Zhihui Wang, Bryan O'Gorman, Davide Venturelli, Eleanor Rieffel, and Jeremy Frank. Planning for compilation of a quantum algorithm for graph coloring. *arXiv preprint arXiv:2002.10917*, 2020.

[13] EPiQC. Scaffcc benchmarks. https://github.com/epiqc/ScaffCC/tree/master/Algorithms, 2020.

[14] Google. Cirq benchmarks. https://github.com/quantumlib/Cirq/tree/master/examples, 2018.

[15] Gian Giacomo Guerreschi and Jongsoo Park. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology*, 3(4):045003, 2018.

[16] Steven Herbert and Akash Sengupta. Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers. *arXiv preprint arXiv:1812.11619*, 2018.

[17] IBM. Openqasm benchmarks. https://github.com/Qiskit/openqasm/tree/master/examples, 2017.

[18] IBM. Application-based qiskit benchmarks. https://github.com/Qiskit/qiskit-aqua/tree/master/qiskit, 2020.

[19] IBM. Python qiskit benchmarks. https://github.com/Qiskit/qiskit-tutorials/tree/master/tutorials/algorithms, 2020.

[20] IBM. Python qiskit quantum volume model circuit. https://github.com/Qiskit/qiskit-terra/blob/master/qiskit/circuit/library/quantum_volume.py, 2020.

[21] IBM. Python qiskit quantum volume model circuit in openqasm. https://qiskit.org/documentation/tutorials/noise/5_quantum_volume.html, commit = , 2020.

[22] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, and Atsushi Matsuo. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*, 70:43–50, 2020.

[23] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. Scaffcc: Scalable compilation and analysis of quantum programs. *Parallel Computing*, 45:2–17, 2015.

[24] Stephen P Jordan. Fast quantum algorithm for numerical gradient estimation. *Physical review letters*, 95(5):050501, 2005.

[25] Lingling Lao, Bas van Wee, Imran Ashraf, J van Someren, Nader Khammassi, Koen Bertels, and Carmen G Almudever. Mapping of lattice surgery-based quantum circuits on surface code architectures. *Quantum Science and Technology*, 4(1):015005, 2018.

[26] Ang Li. Openqasm benchmarks collection. https://github.com/uuudown/QASMBench, 2019.

[27] Ang Li and Sriram Krishnamoorthy. Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation. *arXiv preprint arXiv:2005.13018*, 2020.

[28] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.

[29] Gushu Li, Yufei Ding, and Yuan Xie. Towards efficient superconducting quantum processor architecture design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1031–1045, 2020.

[30] Chia-Chun Lin, Amlan Chakrabarti, and Niraj K Jha. Qlib: Quantum module library. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 11(1):1–20, 2014.

[31] Dmitri Maslov. Basic circuit compilation techniques for an ion-trap quantum machine. *New Journal of Physics*, 19(2):023035, 2017.

[32] N David Mermin. *Quantum computer science: an introduction.* Cambridge University Press, 2007.

[33] Microsoft. Q. https://github.com/microsoft/Quantum, 2020.

[34] Microsoft. Tutorials and programming exercises for learning q and quantum computing. https://github.com/microsoft/QuantumKatas, 2020.

[35] Daniel Mills, Seyon Sivarajah, Travis L Scholten, and Ross Duncan. Application-motivated, holistic benchmarking of a full quantum computing stack. *arXiv preprint arXiv:2006.01273*, 2020.

[36] Matthias Moller. Libket framework. https://gitlab.com/mmoelle1/LibKet/-/tree/matthias_branch, 2020.

[37] Alejandro Morais. Quantum algorithms and their implementation on quantum computer simulators, 2019.

[38] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1015–1029, 2019.

[39] Prakash Murali, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. Formal constraint-based compilation for noisy intermediate-scale quantum systems. *Microprocessors and Microsystems*, 66:102–112, 2019.

[40] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 527–540. IEEE, 2019.

[41] Matteo G Pozzi, Steven J Herbert, Akash Sengupta, and Robert D Mullins. Using reinforcement learning to perform qubit routing in quantum compilers. *arXiv preprint arXiv:2007.15957*, 2020.

[42] QE-Lab. Benchmarks collection of qe-lab. https://github.com/QE-Lab/qbench, 2020.

[43] QuTech. Cqasm grover's algorithm. https://www.quantum-inspire.com/kbase/grover-algorithm/.

[44] QuTech. Cqasm qft algorithm. https://www.quantum-inspire.com/kbase/libket/.

[45] QuTech. Python quantum inspire benchmarks. https://github.com/QuTech-Delft/quantuminspire/tree/dev/docs.

[46] RevLib. Revlib. http://revlib.org/functions.php#cat_1, 2020.

[47] Aritra Sarkar. Openql benchmarks. https://github.com/prince-ph0en1x/QAGS/tree/master/Archived, 2018.

[48] Aritra Sarkar. Openqasm qaoa benchmarks. https://github.com/prince-ph0en1x/QuASeR/tree/master/QAOA_DeNovoAsb/test_output, 2020.

[49] Aritra Serkar. Quantum algorithms for pattern-matching in genomic sequences, 2018.

[50] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t| ket>: A retargetable compiler for nisq devices. *Quantum Science and Technology*, 2020.

[51] Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture, 2016.

[52] Bochen Tan and Jason Cong. Optimality study of existing quantum computing layout synthesis tools. *arXiv preprint arXiv:2002.09783*, 2020.

[53] UCLA. Queko benchmark. https://github.com/UCLA-VAST/QUEKO-benchmark, 2020.

[54] Diogo Valada. Openql random circuits. https://github.com/Astlaan/OpenQL/blob/metrics/tools/random_circuit_generator.py, 2020.

[55] Mike van der Lans. Quantum algorithms and their implementation on quantum computer simulators, 2017.

[56] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147, 1996.

[57] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[58] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W Dueck, and Rolf Drechsler. Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pages 220–225. IEEE, 2008.

[59] Robert Wille, Aaron Lye, and Rolf Drechsler. Exact reordering of circuit lines for nearest neighbor quantum architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1818–1831, 2014.

[60] Pengcheng Zhu, Xueyun Cheng, and Zhijin Guan. An exact qubit allocation approach for nisq architectures. *Quantum Information Processing*, 19(11):1–21, 2020.

[61] Pengcheng Zhu, Zhijin Guan, and Xueyun Cheng. A dynamic look-ahead heuristic for the qubit mapping problem of nisq computers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[62] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018.