Characterisation and Benchmarking of Quantum Hardware Simulators and Algorithms through IBM's Qiskit

Project Repository: `https://github.com/SamirFarhat17/quantum-computer-programming-ibm`

1st Samir Farhat Dominguez
Boston University ENG EC713
Boston, Massachusetts
safarhat@bu.edu

***Index Terms*: quantum computing, quantum converters, quantum mappers, quantum solvers, instrumentation, characterization(computing), multi-threading, workloads, benchmarking**

Abstract

The rise of quantum computers has become an increasingly interesting phenomena in the realm of computing. The technology is certainly far from ready for mass adoption, as only the most well funded computing firms and research groups have direct access to these systems.

However, as the technology matures, and adoption increases, those who are looking to fill a demand for high-intensity computing will have to begin considering quantum hardware. As a result, developing tools to instrument and benchmark quantum hardware under different workloads could become ingreasingly useful. This work will seek to utilize IBM's Qiskit Quantum framework in order to delve into the realm of benchmarking and characterisation of quantum hardware and applications. This report presents several metrics regarding the runtime behavior of quantum systems under different parameters, noise enviornments, pulse frequencies, and qubit count profiles. Moreover, this work benchmarks the simulation of problem sets that are specifically benefited by quantum systems and qubit superposition of representations.

I. Introduction

Within the field of computing, the notion of timing relative to power and cost is perhaps the most important factor when deliberating whether a solution or innovation is valuable. One of the greatest hurdles when it comes to developing software for modern computing architectures is the consideration for time and space complexity. Whether it is in academia or enterprise environments, software and hardware is often judged on its ability to perform tasks within a certain memory, hardware occupation, and time on the processor.

There is also ample literature on how to utilize data structures and algorithmic techniques to perform tasks whilst minimizing the hardware utilized [10]–[15]. There has also been extensive research on how to wield hardware such as GPU, multi-core, multi-threading, dynamic architecture, approximate computing, etc; to improve the power and time returns on applications and workloads [2]–[5]. There has also been extensive investigations on wielding hardware platforms such as cloud computing, hpc, distributed-computing, containerization, etc; in order to accomplish the same goal [6]–[9].

A. Quantum Computing

Despite these innovations and improvements, there is a constant and unwavering limiting factor, that of classical computing. Classical computing is often thought of as the contemporary computers of now; which are multi-core, multi-threaded, sometimes heterogeneous and with reduced instruction set architectures [16]. However, classical computing encompasses any system where the fundamental data unit takes the form of binary values, whether it be V_{DD} and GND or more commonly $1s$ and $0s$ [17].

Quantum computing encompasses any computing or system architecture that operates with a fundamental data unit known as the qubit. A qubit, or quantum bit, emerges from a phenomena in quantum mechanics known as quantum superposition, where a single entity can have different momentum, energies and positions depending on the factors or lenses through which they are observed. Similarly, quantum bits are characterised by being able to represent different values simultaneously, otherwise known as a superposition of representation [18]. Since humans generally don't interact with quantum phenomena, it can be a very challenging concept to grasp the concept, but suffice to know that a qubit is able to represent factorials more states then a bit, and can do so simultaneously. This nature is depicted in the following figure.

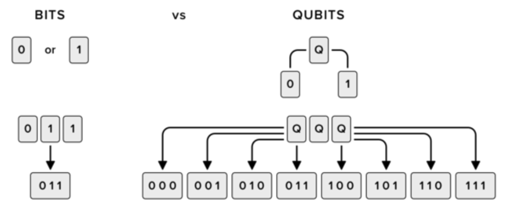

Fig. 1. Potential of a qubits for data representation vs traditional bits

Researchers have been able to produce structures with programmable qubits by bringing circuitry to extremely cold temperatures and manipulating them with magnetic fields in an isolated and controlled enviornment. This means that it can be prohibitively expensive in terms of cost of materials and power to operate a quantum machine. Consequently, physical quantum hardware is generally only available to research groups within academia and commercial computing firms [18].

B. Instrumentation and Characterisation

As mentioned, of the downsides of quantum computing, at least with the current technology available, is that it is very expensive to maintain their operation [18]. Additionally, quantum computers are more efficient only for certain types of workloads, often ones that have high permutation counts or problems that deal with a high dimensionality of features [19]. As a result, as quantum computers develop, parties focused on efficiency and maximizing computing within budgetary constraints will need to perform cost-benefit analyses on the adoption of quantum hardware (against say adding GPUs, faster memory, processors, etc).

To perform these analyses, it will be imperative to have the tools and data to observe how instruction counts, power caps, and speed characteristics of quantum systems with distinct workloads and applications. While the full profiling of all these aspects escapes the scope of this work, exploring the viability of instrumentation, characterisation, and benchmarking on current available quantum systems may be immensely valuable for future work to be developed.

II. Background

From the perspective of hobbyists or students, the current crop of tools and frameworks to interact with quantum computing are fairly limited. This is sure to become less of an issue as the technology maturates, but as it stands the most comprehensive framework/library to interact quantum systems is IBM's Qiskit.

A. Qiskit

There are several modules that encompass the Qiskit space, all of which have different purposes and offerings to those seeking interactions with quantum computing. [19]

1) Qiskit OpenQASM: OpenQASM can be thought of as an assembly or hardware description language for quantum circuits. In fact, all modules that pull quantum hardware to implement quantum solutions simply abstract away the implementation of circuits by calling OpenQASM modules.

As a consequence, OpenQASM, or tools that interact with OpenQASM, give us the the finest resolution to instrument quantum hardware and gain a better understanding of the qubit states at that level. Qiskit utilizes these circuits to implement converters and more complex computing hardware to solve several different types of problem sets. These problem sets often benefit from quantum computing, meaning that the time and space complexity is such that classical computing is slow or inefficient at dealing with these [20].

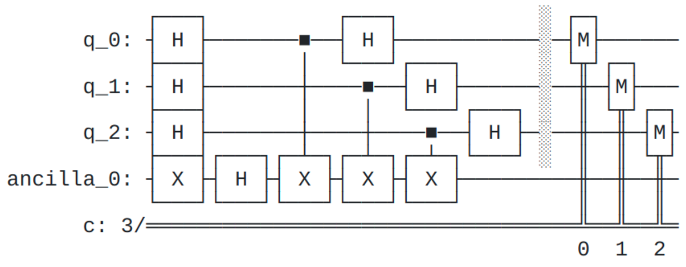

Fig. 2. 4-qubit Bernstein-Vazirani circuit as implemented on OpenQASM

2) Qiskit Aer: The Aer module can be thought of as a middleware API between the high level Python implementations of algorithms and OpenQASM converters and circuits. As such, it will be one of the most valuable libraries in terms of bridging the gap between quantum hardware and algorithms.

3) Qiskit Finance: Qiskit finance presents a catalogue of modules and converters specialized for finance algorithms. These include bull spread calculation, portfolio optimization and candle amplitude estimation.

4) Qiskit Machine Learning: The machine learning Qiskit library presents quantum mappers to connect machine learning techniques to quantum converters, including neural network regressors and classifiers.

5) Qiskit Nature: Qiskit's nature module is particularly interesting in that it presents the potential of quantum computing for research in the natural sciences. The library covers several converters and algorithms for ground state solvers of different compounds, protein folding of different peptides, and vibrational-electronic structure Hamiltonian solvers.

6) Qiskit Optimization: The optimization module implements several quantum converters and algorithms, these can be utilized in several ways to solve problems related to graphs, queues, lists, maps and other operations on data-structures with high combinatorial counts [21].

It's important to mention that the finance, nature, machine learning, and optimization modules all have some overlap when it comes to quantum converters and methodologies. There are several other frameworks available under the Qiskit umbrella, but for the scope of this work the above libraries are most pertinent.

B. Airspeed velocity

As the majority of Qiskit is implemented on python, the airspeed velocity tool is a good way to establish benchmarks for the different algorithms and hardware executions. ASV is a state of the art application profiling tool specifically designated for Python 3 packages.

In the scope of this work, ASV gives us a way to tool and select what information to gather from a running program. As a result, ASV can be used to report the time that the simulated hardware takes to run algorithms or quantum solvers.

III. Experimental Methodology

The breadth of experimentation for this work is fairly extensive. It covers quantum characterisation and benchmarking from the circuit level to the algorithm/application level, this is detailed below.

The experimental section of this work will be subdivided into four stages. The benchmarking of simulated quantum circuits and converters/solvers at ideal noise conditions, benchmarking of particular converters/solvers with different noise conditions, benchmarking of different converters/solvers at varying pulse sampling frequencies, and finally a thorough benchmarking of quantum algorithms at varying parameters relevant to the algorithm.

A. Circuit Benchmarks at ideal noise

This stage of experimentation is a simple benchmarking of quantum circuits simulated on classical hardware [1]. The hardware being simulated varies severely in complexity, ranging from simple logic circuits all the way to problem mappers and solvers which in themselves can solve complex algorithms. Their behavior is charted at 5, 15, and 25 qubits as Qiskit notes that these are representative data points of real variation [1].

- Arithmetic Circuits
  - Integer Comparator
  - Weighted Adder
  - Quadratic Form
  - Draper Quantum Fourier Transform Adder
  - CDKM Ripple-carry Adder
  - VBE Ripple-carry Adder
  - HRS Cumulative Multiplier
  - RGQFT Multiplier
- Quantum Fourier Transform Circuit
- N-Local Circuits
  - Real Amplitudes Circuit
  - Linear Real Amplitudes Circuit
  - Efficient SU2 Circuit
  - Linear Efficient SU2 Circuit
  - Excitation Preserving Circuit
  - Linear Excitation Preserving Circuit
- Particular Circuits
  - Fourier Checking Circuit
  - Graph State Circuit
  - Hidden Linear Function Circuit
  - Instantaneous quantum Polynomial circuit
  - Quantum Volume Circuit
  - Phase Estimation Circuit
- Probability Distribution Circuits
  - Uniform Distribution Circuit
  - Normal Distribution Circuit
  - Log Normal Distribution Circuit
- Standard Gates
  - Barrier
  - MC-Phase
- Pauli Rotation Circuits
  - Linear Pauli Rotation
  - Polynomial Pauli Rotation
  - Piecewise Linear Pauli Rotation
  - Piecewise Polynomial Pauli Rotation
- Generalized Gates
  - Multi-controlled multi-target gate (MCMT)
  - MCMT CCX-chain implemented gate
  - Permutation gate
  - Global R-gate
  - General Mølmer–Sørensen Gate (GMS)
  - GMS X-axis Gate
  - GMS Y-axis Gate
  - GMS Z-axis Gate

B. Particular Converters/Solvers at varying noise

Aer allows for the running of hardware under different noise conditions. This is particularly important as the way by which qubits are programmed means noise is a constant consideration. As was previously discussed, the extremely precise materials and devices needed to perform operations at the quantum level can be affected by noise due to several factors, most of which are basically invisible or require specialized equipment to measure [25].

The types of noise tested are those due to damping error and depolarizing error, which Aer is able to model. This work looks at the noise profiles for those circuits that are particularly useful for quantum algorithms [1]. As such, the following circuits were selected for noise profiling.

- Fourier Checking Circuit
- Graph State Circuit
- Hidden Linear Function Circuit
- IQP Circuit
- QV Circuit
- Phase Estimation Circuit

C. Particular Converters/Solvers at different pulse frequencies

In order to ensure that qubits are performing correctly, they need to be tuned every once in a while, the rate at which they are tuned can have a significant effect on performance should conditions be non-ideal, which is nearly impossible outside the vacuum of space [25].

Therefore, it is useful to run benchmarks of quantum hardware when the tuning pulses occur at different frequencies. Namely, 1, 10, 100, 1000 and 10,000; as Qiskit refers these as good thresholds that current quantum computers may run [10]. These pulses have a relatively high overhead, and therefore is important in a cost-benefit analysis of the strengths of adopting quantum nodes. The circuits selected for this benchmarking are the same as above, as these are high complexity converters and solvers that are integrated in several algorithms under the finance, machine learning, nature and optimization modules of Qiskit [1].

D. Algorithm Benchmarking

The algorithm benchmarks are of particular importance, as benchmarks for these will allow potential quantum computing adopters to understand which workloads and applications benefit from quantum hardware. The algorithms are enumerated as follows:

- Machine Learning
  - Quantum Neural Network Classifier
  - Op-flow QNN Classifier
  - Variational Quantum Classifier
- Nature
  - Protein Folding
  - Ground State Solver
  - Vibrations State Solver
  - Hamiltonian Solver(VQE)
- Optimization
  - Clique Problem
  - Docplex Problem
  - Knapsack Problem
  - Vertex Cover Problem

IV. Implementation

All of the experimental stages have quite similar workflow. This section details and depicts the workflow pipeline and its implementation.

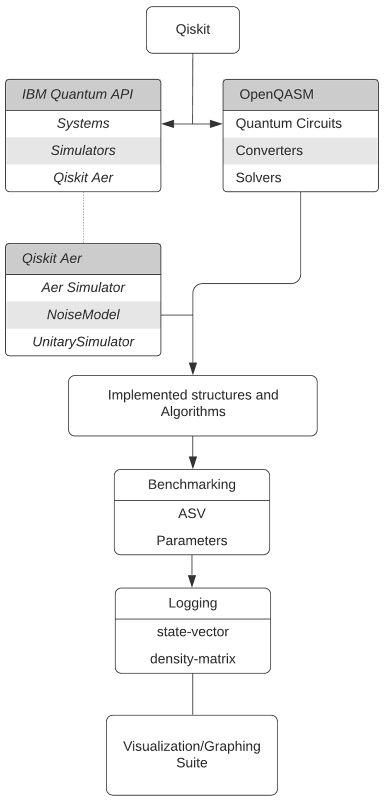

Fig. 3. Pipeline of experimental stage

A. Qiskit Integration

The first few layers of the pipeline involve direct interaction with Qiskit sub-modules. First, one needs to establish credentials with the IBM Quantum API to gain access to backend hardware and simulators. These can be called upon through the Qiskit Aer libraries using the api key. Aer also gives one access to noise modulators and The OpenQASM modules are all separately available on the Qiskit github. Specific to this work, the Aer and unitary simulators as well as noise models are called from the Aer backend.

B. Native Project Level

Following this, the different circuits, converters and solvers to be benchmarked are implemented. These modules can be then pointed to by keeping them under the same directory and referring to this directory in the .asv configuration file. Here noise models and pulse models can be retrieved from Terra, a sub-module of Aer. Functions defining the initialization and running of circuits and algorithm are defined, and the running of these is orchestrated under the .asv specified directory.

ASV runs at default settings on these retrieving the completion time, and these results are sent in Json format under the directory specified by asv, and finally logged as per asv convention and the state vector or matrix product state format. Finally having collected the data, it is parsed through and plotted using Python numpy and matplotlib libraries.

C. Contributions

Several parts of the implementation had already been implemented as examples under qiskit Aer. However data collection for these had never transpired. This work made the following contributions from the building blocks of Aer.

1) Trial runs for several algorithm types under machine learning, nature, finance and optimization
2) New ASV enviornment coniguration specific to repository under the main branch
3) All data from benchmarking circuits, converters, solvers and algorithms
4) Circuits, Converters and Solvers
   - All ripple-carry adder and multipliers
   - Excitation preserving circuits
   - All probability disttribution circuits
   - All standard gates
   - All Pauli Rotation Circuits
   - All Generalized Gates
5) Noise Models for the following
   - Fourier Checking Converter
   - Graph State Converter
   - Hidden Linear Function Solver
   - Phase Estimation Circuit
   - Instantaneous Quantum Polynomial
6) Pulse frequency modelling for the above
7) Optimization Algorithms
   a) Knapsack Problem
   b) Clique Problem

   c) Max-cut
   d) Vertex Cover Problem
8) All data processing and graphing

V. Results

As the breadth of experimentation was tremendously wide, only results of interest are reported, as some data may have been from previously implemented benchmarks, or the data collected simply doesn't contain results of interest for the scope of this work.

A. Circuits, Converters and Solvers

1) Arithmetic Circuits: Observing the data below, we see there seems to be a linear relationship between the qubit count and benchmarking time for the weighted adder and quadratic solver circuits. The rest exhibit a fast and constant time no matter the qubit count.

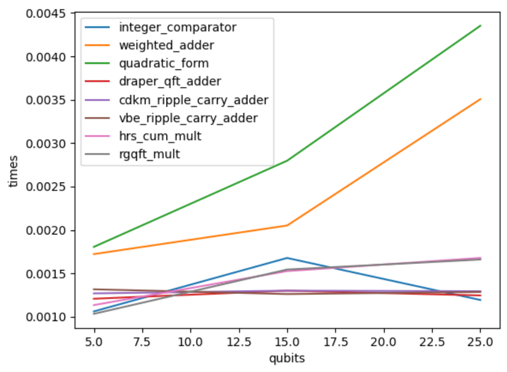

Fig. 4. Benchmarks of Q-Arithmetic circuits under different qubit counts

2) Generalized Gates: By analyzing the figure below, we see a linear relationship between qubits and runtime for mcmt, mcmtv ccx-chain and gms. Contrastingly, the global-R gates show constant runtimes for increasing qubit complexity.

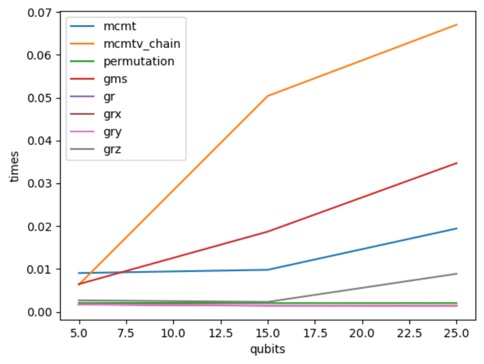

Fig. 5. Benchmarks of Q-Generalized circuits under different qubit counts

3) Pauli Rotational Circuits: Looking at the pauli-rotational circuits, we observe an inverse relationship between qubit complexity and time running. This makes sense, as the additional bits give a higher degree of tools for the solver to perform calculations and represent rotation matirces.

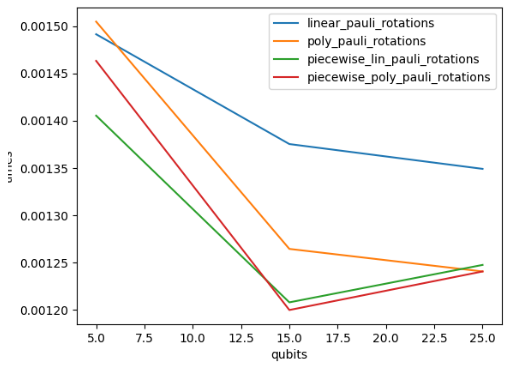

Fig. 6. Benchmarks of Pauli Rotation circuits under different qubit counts

4) Probability Distribution circuits: The probability distribution circuits show that there is a linear relationship between qubit counts and runtime. This demonstrates that the modelling of probability distributions has a linear space-time complexity relationship. This is significant, as the representation of distributions on classical computers tend to have a polynomial relationship when it comes to time and space [26].

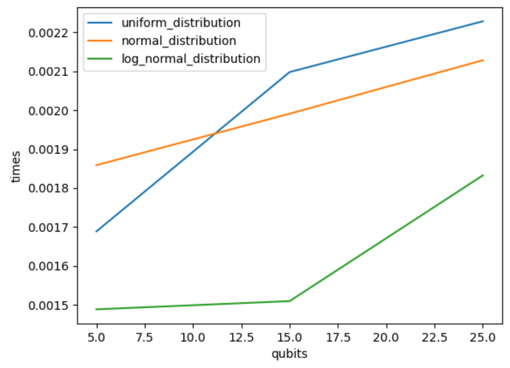

Fig. 7. Benchmarks of Pauli Rotation circuits under different qubit counts

B. Noise Modelling

1) Damping Models: It seems for noise emerging from damping error, the solvers become erratic and vary significantly for runtime, regardless of qubit quantities. This indicates using depolarizing programming channels may not be the best ideas if the equipment and materials required to ensure low error aren't available, or if runtime isn't a massive concern.

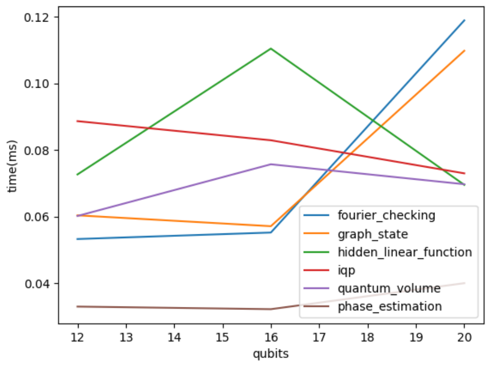

Fig. 8. Benchmarking of Particular Circuits for Damping Noise

2) Depolarizing Models: The behavior of solvers under depolarizing noise greatly contrasts damping error. Only fourier checking sees significant change, and only for 15 qubits.

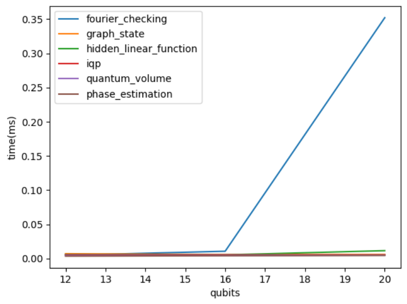

Fig. 9. Benchmarking of Particular Circuits for Depolarizing Noise

C. Pulse Modelling

1) Hidden Linear Function: For the HLF circuit we observe erratic runtimes for different frequencies and qubits, particularly for 5 qubit. Ergo, we may claim that HLF is one of those circuits where runtime may always be inconsistent.

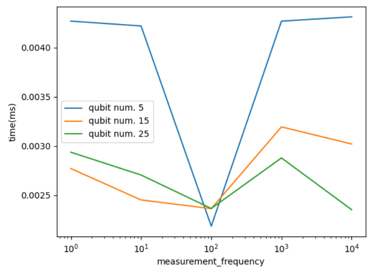

Fig. 10. Benchmarking of Hidden Linear Function at Different Frequencies

2) Phase Estimation: The phase estimation circuit shows a complete independence of tuning frequency, and that runtime is very consistent when the number of qubits is known. This may be valuable asset to those looking to offer a constant runtime rather then those pushing for highest speeds by having the most finely grained tuners.

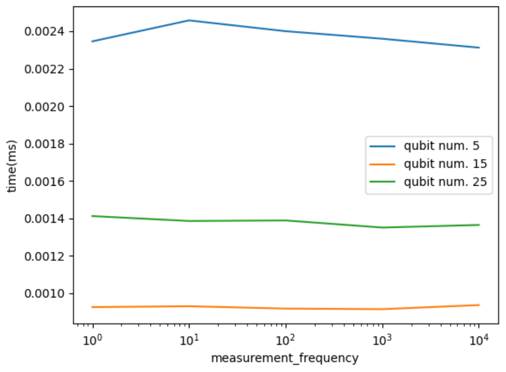

Fig. 11. Benchmarking of Phase Estimation for Different Tuning Frequencies

3) Instantaneous Quantum Polynomial Circuit: The IQP Circuit shows a constant runtime for 15 and 25 qubits, but some eraticism under the 5-qubit setting. This may indicate that to find consistency for this circuit it may be necessary to give it a higher number of programmable units.

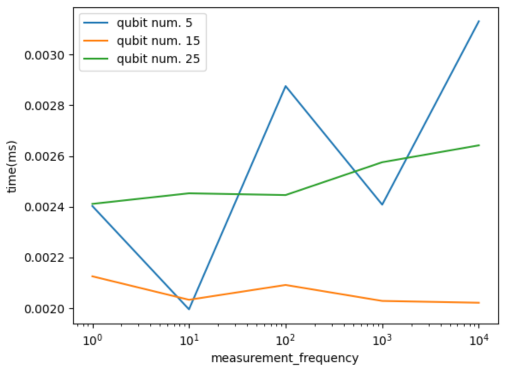

Fig. 12. Benchmarking of Instantaneous Quantum Polynomial for Different Tuning Frequencies

D. Algorithms

1) Clique: The clique problem shows a somewhat exponential resolution time with increased nodes and an inverse coorelation with higher degree counts. This shows that the benefits of quantum computing for the clique problem may not be that beneficial then when compared to classical computing, although the clique problem does have a polynomial big-O time complexity with classical systems [27], so it is definitely still an improvement.

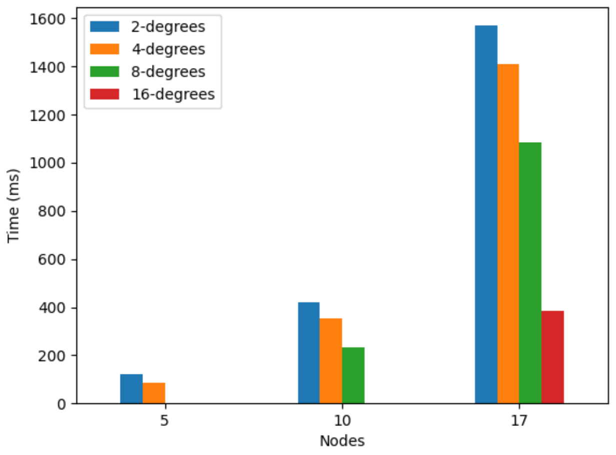

Fig. 13. Benchmarks for Clique Application with varying nodes and degrees

2) Maxcut: The maxcut problem has a very stark linear relation to number of degrees, however the performance of the quadratic unconstrained binary optimization solver (qubo) greatly outperforms variational quantum eigenolver(vqe) and the quantum approximate optimization algorithm(qaoa). Most importantly, grover greatly outperforms even qubo, so much so that grover had to be scaled up by a magnitude of 10.

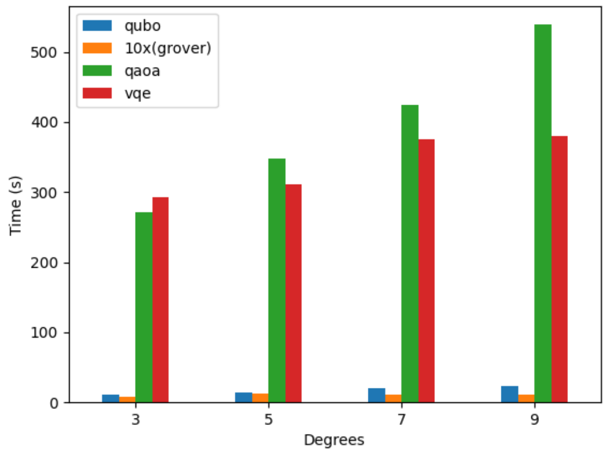

Fig. 14. Benchmarks for Maxcut Application of 16 nodes with varying solvers and degrees with different quantum solvers

3) Knapsack: The knapsack problem, even though it is also a graph resolution problem, shows greatly different results then maxcut. Here the solvers runtime increases linear with weights, but all perform similarly except for qubo. Qubo drastically overperforms the rest, so this shows that different quantum hardware may be specialized for different quantum problems. Once again maxcut and knapsack have factorial time complexity for classical systems [27], so linear time complexity shows how beneficial quantum systems are for these sorts of problems.

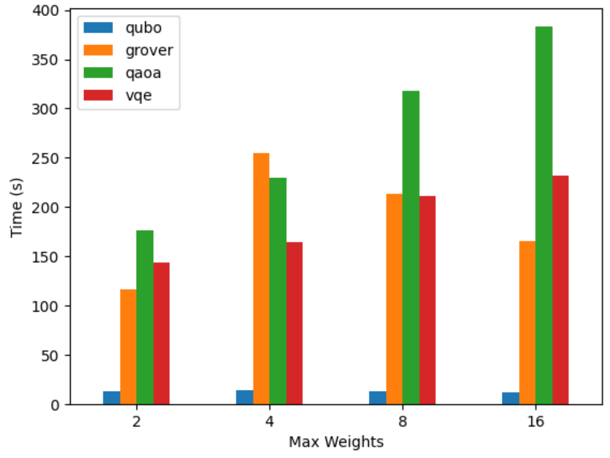

Fig. 15. Benchmarks for Knapsack Application with varying nodes and degrees

4) Vertex Cover: Vertex cover shows very similar results to maxcut. There is a distinct linear time complexity, as benchmarks increase proportional to node degrees. Once again grover and qubo greatly outperform qaoa as well as vqe. And once again grover is so strong that it needs scaling up.

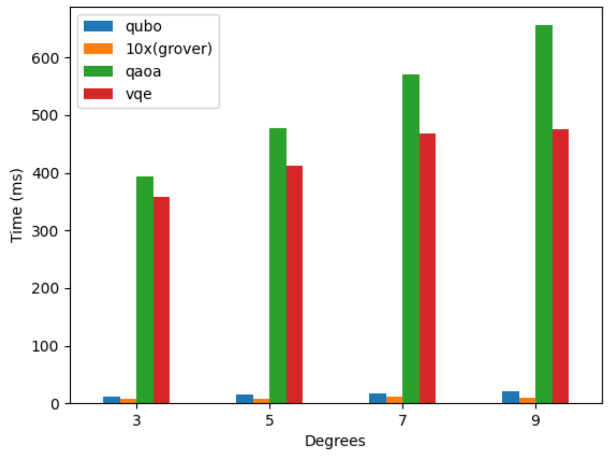

Fig. 16. Benchmarks for Vertex Cover Application with varying nodes and degrees

VI. Conclusion

A. Discussion and Reflection

In this work, we were able to interact with quantum computers to further our comprehension of the sorts of workloads, systems and applications which may benefit from the use of quantum hardware. While the results of this work are far from conclusive, this has been a good step in the direction of truly measuring the benefits of quantum computing, at least in the realm of specific problem types and applications.

Additionally, this work was a great opportunity to develop some familiarity with Qiskit and to interact for the first time with quantum computers. The Qiskit team and community is open to contributions and development, and with some polishing of the code a possible contribution to their repositories may be possible, the steps to do so will be coming in the next few weeks after submission of this work.

B. Limitations and Extensions

Due to the over-ambitious scope and extensive experimentation breadth there are several loose ends and further questions that have emerged. One of the most notable of these is the inherent weakness of simulated hardware. Due to time and resource constraints, all benchmarks had to be ran on simulators, which means they may not necessarily reflect results in real world conditions.

Additionally, it must be noted that this was an investigative nature of the work, and was more of a 'dip our toes' approach to what is a massive undertaking. This problem is that of developing robust quantum characterisation and benchmarking tools, the work was a bit unfocused. A lot of data was collected, and some conclusions could be made of it, but a lot of these are tentative and subject to scrutiny.

This work also leaves a few interesting follow-ups to be explored. One of these is the different problem types which may be benefitted by quantum enviornments; Further optimization problems, approximate computation, cryptography adversary attacks, and SVMs to name a few. Moreover, once experimentation can take place on real quantum hardware, it would be interesting to perform benchmark comparisions with high intensity classical computing nodes, such as HPC, cloud systems, GPUs etc.

References

[1] McKay, David C., et al. "Qiskit backend specifications for openqasm and openpulse experiments." arXiv preprint arXiv:1809.03452 (2018).

[2] Cochran, Ryan, et al. "Pack & cap: adaptive dvfs and thread packing under power caps." 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2011.

[3] Eggers, Susan J., et al. "Simultaneous multithreading: A platform for next-generation processors." IEEE micro 17.5 (1997): 12-19.

[4] Boyd-Wickizer, Silas, Robert Tappan Morris, and M. Frans Kaashoek. "Reinventing scheduling for multicore systems." HotOS. 2009.

[5] Venkatesh, Ganesh, et al. "Conservation cores: reducing the energy of mature computations." ACM Sigplan Notices 45.3 (2010): 205-218.

[6] Ferdman, Michael, et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." Acm sigplan notices 47.4 (2012): 37-48.

[7] Cong Xu, Karthick Rajamani, Alexandre Ferreira, and Aditya Akella. Iron: isolating network-based CPU in container environments. In Proceedings of the 15th USENIX Conference on Networked Systems

[8] Choi, Jee, et al. "Algorithmic time, energy, and power on candidate HPC compute building blocks." 2014 IEEE 28th international parallel and distributed processing symposium. IEEE, 2014.

[9] Choi, Jee, et al. "Algorithmic time, energy, and power on candidate HPC compute building blocks." 2014 IEEE 28th international parallel and distributed processing symposium. IEEE, 2014.

[10] Giakkoupis, George, and Philipp Woelfel. "On the time and space complexity of randomized test-and-set." Proceedings of the 2012 ACM symposium on Principles of Distributed Computing. 2012.

[11] Poland, Jan, and Andreas Zell. "Main vector adaptation: A CMA variant with linear time and space complexity." Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation. 2001.

[12] Streeter, Matthew J. "Upper bounds on the time and space complexity of optimizing additively separable functions." Genetic and Evolutionary Computation Conference. Springer, Berlin, Heidelberg, 2004.

[13] Rathod, Pawan, Ankit Vartak, and Neha Kunte. "Optimizing the complexity of matrix multiplication algorithm." 2017 International Conference on Intelligent Computing and Control (I2C2). IEEE, 2017.

[14] Wang, Hao, et al. "Time complexity reduction in efficient global optimization using cluster kriging." Proceedings of the Genetic and Evolutionary Computation Conference. 2017.

[15] Datta, Samir, and Raghav Kulkarni. "Space complexity of optimization problems in planar graphs." International Conference on Theory and Applications of Models of Computation. Springer, Cham, 2014.

[16] Ceruzzi, Paul E., E. Paul, and William Aspray. A history of modern computing. MIT press, 2003.

[17] Manin, Yuri I. "Classical computing, quantum computing, and Shor's factoring algorithm." ASTERISQUE-SOCIETE MATHEMATIQUE DE FRANCE 266 (2000): 375-404.

[18] Fisher, Chris. "What Is Quantum Computing?" IBM Quantum, International Business Machines, 2 Apr. 2009, https://www.ibm.com/quantum-computing/what-is-quantum-computing/.

[19] Cross, Andrew. "The IBM Q experience and QISKit open-source quantum computing software." APS March Meeting Abstracts. Vol. 2018. 2018.

[20] Burgholzer, Lukas, Rudy Raymond, and Robert Wille. "Verifying results of the IBM Qiskit quantum circuit compilation flow." 2020 IEEE International Conference on Quantum Computing and Engineering (QCE). IEEE, 2020.

[21] Zhang, Xin, et al. "An efficient quantum circuits optimizing scheme compared with QISKit." arXiv preprint arXiv:1807.01703 (2018).

[22] Droettboom, Michael. "airspeed velocity Documentation." (2019).

[23] van der Walt, Stéfan J., and K. Jarrod Millman. "The Next Decade of Scientific Python." University of California, Berkeley (2020).

[24] Downey, Allen B. Python for software Design. Cambridge University Press Textbooks, 2009.

[25] Hill, Scott, and William K. Wootters. "Entanglement of a pair of quantum bits." Physical review letters 78.26 (1997): 5022.

[26] D. Bellot, "Approximate discrete probability distribution representation using a multi-resolution binary tree," Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence, 2003, pp. 498-503, doi: 10.1109/TAI.2003.1250231.

[27] Pardalos, Panos M., and Jue Xue. "The maximum clique problem." Journal of global Optimization 4.3 (1994): 301-328.