Reliability Testing, Noise and Error Correction of Real Quantum Computing Devices

Ilias P. Galanis, Ilias K. Savvas, Andrey V. Chernov, and Maria A. Butakova

***Abstract* — From Pharmacology to Cryptography and from Geology to Astronomy are some of the scientific fields in which Quantum Computing potentially will take off and fly high. Big Quantum Computing vendors invest a large amount of money in improving the hardware and they claim that soon enough a quantum program will be hundreds of thousands of times faster than a typical one we know nowadays. But still the reliability of such systems is the main obstacle. In this work, the reliability of real quantum devices is tested and techniques of noise and error correction are presented while measurement error mitigation is explored. In addition, a well-known string matching algorithm (Bernstein–Vazirani) was applied to the real quantum computing device in order to measure its accuracy and reliability. Simulated environments were also used in order to evaluate the results. The results obtained, even if these were not 100% accurate, are very promising which proves that even these days a quantum computer working side by side with a typical one is reliable and especially when error mitigation techniques are applied.**

***Keywords* — Qiskit, Quantum programming, quantum noise, quantum error, Error mitigation, string matching.**

I. Introduction

Quantum computing has been improved on both hardware and software in the last years. Real Quantum Computing Devices (QCD) are already available to researchers in order to test their algorithms and programs. It is a fact now that when Quantum Devices with a large number of qubits will be reliable our world will be shaken. From cryptography to chemistry, from pharmacology to machine learning, almost all scientific fields will change our life dramatically. Quantum computing principles have been applied to a large variety of applications up to now.

Paper received April 05, 2021; accepted June 05, 2021. Date of publication July 31, 2021. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Miroslav Lutovac.

This paper is a revised and expanded version of the paper presented at the 28th Telecommunications Forum TELFOR 2020 [24].

The work was financially supported by Russian Foundation for Basic Research (projects 19-01-00246-a, 19-07-00329-a).

I. P. Galanis is with the Department of Digital Systems, University of Thessaly, Larissa, Greece (e mail: ilgalanis@uth.gr).

Corresponding Author: I. K. Savvas is with the Department of Digital Systems, University of Thessaly, Larissa, Greece (e mail: isavvas@uth.gr).

A. V. Chernov is now with Smart Materials Research Institute, Southern Federal University, Rostov-on-Don, Russia (e-mail: cherno@sfedu.ru).

M. A. Butakova is now with Smart Materials Research Institute, Southern Federal University, Rostov-on-Don, Russia (e-mail: mbutakova@sfedu.ru).

For example, in signal processing parts of the physical layer in wireless communications. More specifically, since 2009 in [1] the authors have presented the theory of quantum serial turbo error correction codes, where they describe their iterative decoding algorithm and finally they study their performances numerically on a depolarization channel. Furthermore, in [2] the authors investigate the use of non-binary classical error-correcting codes in facilitating reliable, entanglement-assisted communication of classical information over quantum depolarizing channels. Finally, quantum stabilizer codes suffer from a low quantum coding rate since they have to recover the quantum bits in the face of both bit-flip and phase-flip errors. To overcome this drawback, in [3] the authors propose a low-complexity concatenated quantum turbo error correction code design exhibiting a high quantum coding rate.

On the other hand, drug discovery and development is another example of the potential use of quantum computing where quantum simulation could speed up characterizations of molecular systems and combined with quantum machine learning could be again a useful tool in early phases if drug discovery [4].

But here stands the main obstacle, the reliability of quantum systems. Nowadays many big vendors like IBM, Google, Microsoft, Rigetti, Honeywell and many others invest a large amount of money in improving their quantum devices [5-9]. Already many Python packages have been developed for quantum programming like Qiskit [10], PyQuil [11], and Cirq [12] while Microsoft developed <Q#> [13] and even new packages for Julia programming language are proposed [14].

In this work, an investigation of the reliability of quantum devices has been made, and especially IBM Quantum Device (QCD) has been tested in string prediction (string matching). To achieve that, quantum noise and errors' correction hve been studied while measurements on errors mitigation to calibrate quantum circuits have also been explored.

String matching, which actually is a bit-sequence matching especially for large texts, is a very time-consuming procedure while a QCD with the appropriate number of qubits could predict in just one shot. So, a well-known quantum algorithm, introduced by Bernstein–Vazirani [15], for string searching was tested and comparatively studied regarding its reliability to two real quantum devices which differ in the number of available qubits. The results (probabilities) of matching the whole string in just one shot and when a mitigation technique is applied to the QCDs the possibilities of exact prediction/matching are further increased.

The rest of the paper is organized as follows. A brief description of quantum programming is given in Section II. In Section III, techniques for quantum noise and error correction are presented while in Section IV measurement error mitigation is discussed. A prediction of a sequence of bits is implemented. Predicting a bit-sequence and the string matching technique is presented and discussed in Section V, and finally, Section VI concludes this work and introduces some suggestions for future work.

II. Quantum Programming with Qiskit

Basic information in a Quantum world is a qubit which is analogous to a classical bit but instead of being either 1 or 0, it can be |0>, |1> or both. A qubit is actually a vector of the 2-dim Hilbert space as follows:

$$|0\rangle:\begin{pmatrix}1\\0\end{pmatrix}, |1\rangle:\begin{pmatrix}0\\1\end{pmatrix}$$

The interesting thing here is that the qubits could be in a super-position according to the following equation which means that at any time a qubit can be either 1 or 0 with equal probabilities.

$$|\psi\rangle = a|0\rangle + b|1\rangle = \begin{pmatrix}a\\b\end{pmatrix}, |a|^2 + |b|^2 = 1$$

where a and b are complex numbers ($\pm 1/\sqrt{2}$).

Combining more than one qubit, for example 2 qubits, they produce a 4-dim space applying the Kronecker (tensor) product:

$$|00\rangle:\begin{pmatrix}1\\0\end{pmatrix}*\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1*\begin{pmatrix}1\\0\end{pmatrix}\\0*\begin{pmatrix}1\\0\end{pmatrix}\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$$

This means that by combining 3 qubits they produce an 8-dim space, 1,000 qubits approximately 10^{30}-dim space(!), and so on.

A graphical representation of qubits can be shown by using a Bloch Sphere as in Fig. 1.

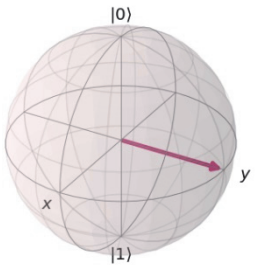

Fig. 1. Bloch sphere.

Qiskit [7] is a python package from IBM which supports quantum programming on both simulated and real device environments. A built-in function in Qiskit can apply all the kinds of transformations on qubits and create quantum registers and circuits. In addition, connecting with IBM real devices one can execute quantum programs in the real world. It is very important that always a quantum device operates with a classical computer. When a quantum circuit is executed in order to store the qubit states, the corresponding number of bits are needed to store them as follows:

```
from qiskit import *
Q = Quantum_Register(N)
C = Classical_Register(N)
circuit = Quantum_Circuit(Q,C)
circuit.measure(Q,C)
```

In this work, for all the experiments Python programming language using the Qiskit package with Jupyter Notebook (Anaconda 3) [16] was used.

The most used quantum gate is probably the Hadamard gate. It operates on a single qubit and puts it on a hyper-position being both 0 and 1 at the same time. The Hadamard gate can be represented as the matrix:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$$

For example, applying the Hadamard gate to qubit |0> the result is as follows:

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

Using the property of Hadamard gate, a very interesting and useful example is to produce random numbers which play an important role in telecommunications [17] and in many other applications, like games, cryptography, simulations and so on. The existing techniques which produce random numbers actually are producing a pseudo-sequence of them. Quantum computing can offer a real random number generator just because of its nature. Just put a qubit in a hyper-position and measure its state. For example, to produce a sequence of random numbers in the range [a,b) we just need a quantum register of $\log_2$(b-a) qubits and put them in a hyper-position (Hadamard Gate) as shown in Fig. 2 and Listing 1 (the example stands for a=0, b=8).

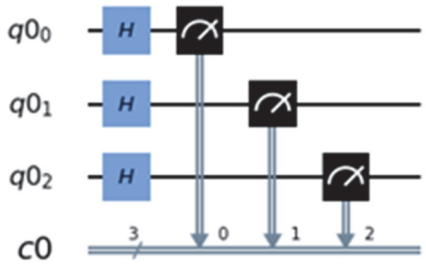

Fig. 2. Quantum circuit to produce a random number between a=0 and b=7.

And that's it. A totally random number has been produced and if we repeat the above procedure a sequence of totally random numbers will appear. And this is that simple as the nature of a qubit in a hyper-position where the possibility to be 0 or 1 is 50%. But in a real device things are slightly different (Listing 2).

LISTING 1: PRODUCING A RANDOM NUMBER

```
n = log(b-a, 2)
qr = QuantumRegister(n)
cr = ClassicalRegister(n)
circuit = QuantumCircuit(qr,cr)
circuit.h(range(n)) # Hadamard gate
circuit.measure(qr,cr) # measurement
simulator = Aer.get_backend('qasm_simulator')
execute(circuit, backend = simulator)
result = execute(circuit, backend = simulator, shots=1).result()
random_number=result.get_counts(circuit)
print(random_number)
```

LISTING 2: PRODUCING A RANDOM NUMBER (IBM REAL QUANTUM DEVICE)

```
------
IBMQ.load_account()
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne')
job = execute(circuit, backend=qcomp, shots=1000)
result = job.result()
-----
```

The output is that all possible combinations could appear and their possibilities are as in Fig. 3. Of course, the possibilities differ each time the program runs; therefore one can always pick the number with the highest or lowest probability and that's it! In this case, the reliability of a QCD is not an issue. On the other hand, when a QCD tries to predict a string, things change. Then the reliability is the main issue so the quantum noise has to be removed and the error correction is now crucial.

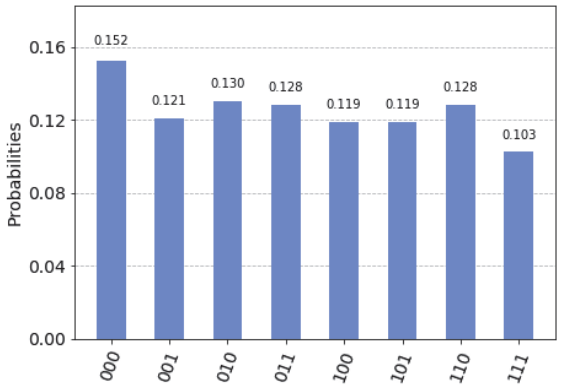

Fig. 3. Possibilities of random number production in a real quantum device.

III. QUANTUM NOISE AND QUANTUM ERROR CORRECTION

Error correction is a major problem for all information processing systems and whenever possible we are building systems that avoid noise completely [18]. Error correction is not really an issue for classical systems since the two states of a classical bit (0 and 1) are so fundamentally different that the probability of happening something like this is negligible [19]. Unfortunately, that is not the case for quantum systems. It is unrealistic to think that it is possible to prepare a qubit precisely in a particular state or apply a certain gate to it that will exactly transform its state into a certain outcome. There are enough reasons that errors occur in quantum computers. The first one is decoherence. Since the quantum system is not perfectly isolated from its environment (assuming that it was, we could neither manipulate nor measure it), over time, information from the system is lost towards the environment. A simple example in quantum computing terms is that if we initialize a qubit in $|1\rangle$ it tends to changing into $|0\rangle$ over time. Sources of noise are random driving forces from the environment (Brownian motion), interactions that produce entanglement between the system and the environment and statistical imprecision in the experimental controls on the system (e.g., timing errors and frequency fluctuations) [20]. Therefore, it is useful to make a distinction between logical qubits (perfect qubits that behave ideally) and physical qubits (qubits that are vulnerable to external sources of noise). From the above, we can conclude that quantum computers require Quantum Error Correction (QEC) in order to be functional. The main idea behind QEC is redundancy. By adding some redundant information to a qubit, even if noise alters its state, we could "decode" that information to gain qubit's pre-altered state. The implementation of redundancy faces obstacles due to the quantum nature of the system:

- Checking for errors is problematic since we cannot measure a qubit without making it collapse and therefore destroy its state.
- No cloning theorem: It is impossible to create an identical copy of an arbitrary unknown quantum state.
- Errors are continuous: A continuum of different errors may occur on a single qubit. Determining which error occurred in order to correct it would appear to require infinite precision, and therefore infinite resources. [18]
- Bit flips are not the only errors. Phase errors (e.g. $|0\rangle + |1\rangle$ to $|0\rangle - |1\rangle$) are just as damaging [19].

Fortunately, there are ways to overcome all these problems. Quantum error Correction codes have been developed over the years. Some of these codes are:

- *Bit flip code (three-qubit bit flip code)* [21]: Encodes a single qubit state $a|0\rangle + b|1\rangle$ into $a|000\rangle + b|111\rangle$. Through measurements we can determine what kind of error occurred and correct them.
- *Sign flip code*: solves the phase errors occurring in quantum circuits. Encodes logical qubit 0 into $|++\rangle$ and logical qubit 1 into $|--\rangle$
- *Shor code* [22]*:* A combination of the bit flip code and sign flip code. At first, physical qubit 0 is encoded into $|++\rangle$ and physical qubit 1 into $|--\rangle$. Then each of these physical qubits are encoded as follows: $|+\rangle$ into $(|000\rangle + |111\rangle)\sqrt{2}$ and $|-\rangle$ into $(|000\rangle - |111\rangle)\sqrt{2}$. In conclusion: Logical qubit 0 = $((|000\rangle+|111\rangle)\ (|000\rangle+|111\rangle)\ (|000\rangle+|111\rangle))/(2\sqrt{2})$

Logical qubit 1 =
((|000⟩-|111⟩) (|000⟩-|111⟩) (|000⟩-|111⟩)) / (2√2)

IV. Measurement Error Mitigation in Qiskit

Qiskit's measurement error mitigation focuses, as its name suggests, on errors occurring in measurements. During the measurement of a circuit of the size of n qubits the possible outcomes are 2^n bit strings. By preparing all the possible 2^n basis states and measuring them, we can determine the probability for each outcome [23]. Then we can store these outcomes in a matrix and use them in order to calibrate our circuit. The overall procedure adopts a linear algebraic approach in its core in order to mitigate the errors. In order to explain this procedure, we define it as:

1. $\mathbf{C_{noisy}}$: A matrix containing the counts of each outcome pre-calibration
2. $\mathbf{M}$: The calibration matrix
3. $\mathbf{C_{ideal}}$: A matrix containing the ideal outcomes for the circuit given that no errors occurred when measured.

From the above, we can conclude that:

$$\mathbf{C_{noisy}} = \mathbf{M}\ \mathbf{C_{ideal}}$$

The unknown factor in this given problem is C_{ideal}. By solving the above equation, the matrix of ideal outcomes is as follows:

$$\mathbf{C_{ideal}} = \mathbf{M}^{-1}\ \mathbf{C_{noisy}}$$

where $\mathbf{M}^{-1}$ is the inverse calibration matrix $\mathbf{M}$. So, the describing algorithm could be:

1. Write the counts dictionaries as column vectors.
2. Run a series of calibration circuits on real devices in order to define what is noise and what is not.
3. Use the data from the calibration circuits to filter out the errors from the circuit we want to run.

As an example, a 4 qubit circuit was created, as shown in Fig. 4 in which the measurement error mitigation technique is applied.

```
nqubits = 4
circuit = QuantumCircuit(nqubits, nqubits)
circuit.h(0)
circuit.cx(0,1)
circuit.cx(1,2)
circuit.cx(2,3)
circuit.barrier()
circuit.measure([0,1,2,3], [0,1,2,3])
```

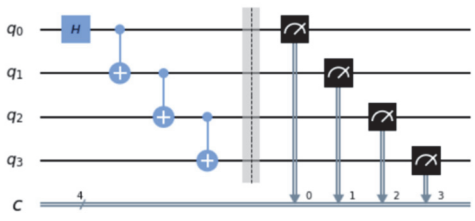

Fig. 4. Noise example circuit.

Running the example circuit on a simulator, the outcome is as shown in Fig. 5.

The same circuit when run on the IBM ibmqx2 quantum device, returns each possible bit string as a result though it is clear that the correct answers are those with the higher probability (Fig. 6).

In addition, from Fig. 7 it is clear that though there is still a small amount of noise, most of the measurement errors are eliminated completely and that the probability of measuring the correct answers, which are the bit's sequences 0000 and 1111 changed from 0.393 to 0.494 and from 0.316 to 0.442 respectively, approaching the ideal results for the example circuit as shown in Fig. 5.

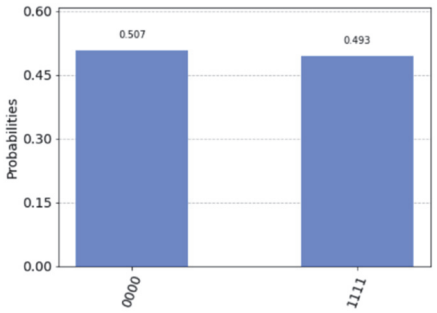

Fig. 5 Possible outcomes for the example circuit (simulator).

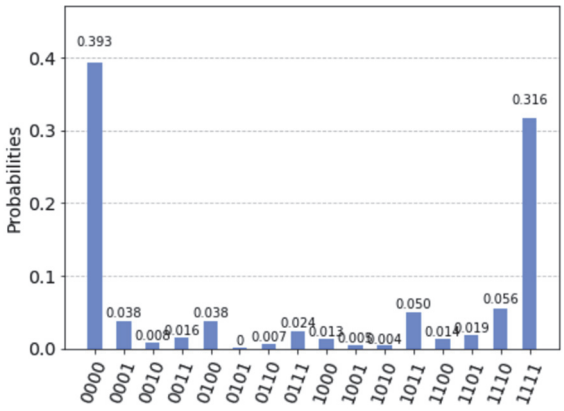

Fig. 6. Possibilities for the example circuit (real quantum device ibmqx2).

Since we implement a 4-qubit circuit, there must be 2^4=16 different calibration circuits, one for each different basis state. Using the outcomes of the calibration circuits, noise is filtered out from our example circuit as shown in Fig. 7.

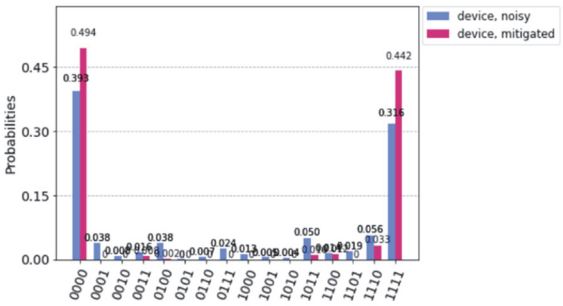

Fig. 7. Comparison of the example 4-qubit circuit before and after calibration.

V. Predicting a Sequence of Bits

Let us assume that a sequence of two bits, s= {1,0,1} is to be predicted by a QCD. In common programming there are 8 states that have to be examined and this gives a worst case of 8 searches. Normally, in just one search a QCD should guess the secret sequence applying the Bernstein - Vazirani algorithm [12] (which is straightforward and relatively simple) as it is presented in Algorithm 1 and implemented in Listing 3.

Algorithm 1: Bernstein-Vazirani Algorithm

1. Put n qubits in state |0>
2. Put these qubits to hyper=position (Hadamard gate/transform)
3. Perform a controlled negation to all of them (Cx gate)
4. Again, put the resulting sates of qubits to hyper-position
5. Measure on the standard basis ({|0>, |1>}) the circuit

Since the controlled negation gate requires an additional qubit, the number of required qubits for a sequence of n bits is n+1. Assuming that only 5 qubits are available (IBM Quantum Device – ibmq_16_melbourn) it is obvious that, in order to be as accurate as possible, only parts of 4 bits will be searched at a time, so in this case, 4 qubits are needed since s={1,0,1}. Algorithm 1 is implemented as in Listing 3, producing the circuit as in Fig. 8.

Listing 3: Bernstein - Vazirani Algorithm

```
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.tools.visualization import plot_histogram
secret = input('Secret (digital) number =');
n = len(secret) #number of digits
circuit = QuantumCircuit(n+1, n)
circuit.h(range(n))
circuit.x(len(secret))
circuit.h(len(secret))
for i, digit in enumerate(reversed(secret)):
   if digit == '1':
      circuit.cx(i, n)
circuit.h(range(n))
circuit.measure(range(n),range(n))
```

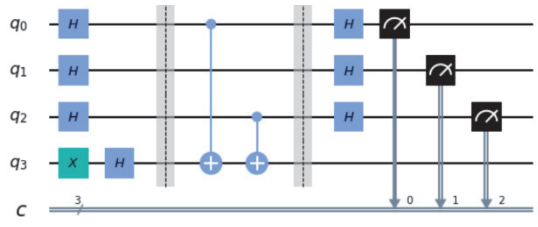

Fig. 8. Circuit for Bernstein - Vazirani Algorithm.

The output again is a sequence of possibilities (Fig. 9), but the good news is that the secret sequence of bits presents the highest possibility 63.6% while the second highest is only 18%. This is a very promising result and led this work further, to try to guess a secret text (a string of characters). Since the same QCD is used (IBM Quantum Device – ibmq_16_melbourn) with 5 available qubits, special treatment has to be made for each 8bit character of the string dividing it into two substrings, each one of them containing 4bits, and after applying the technique, merge them to one to produce the character to be predicted.

The next step is to search a character *secret* string, in this case S='Hi there!'. In order to do that, the same technique as the one described previously is applied, while splitting each character into two 4-bit substrings since a 5 qubit circuit was used (Algorithm 2). Part of the code is presented in Listing 4.

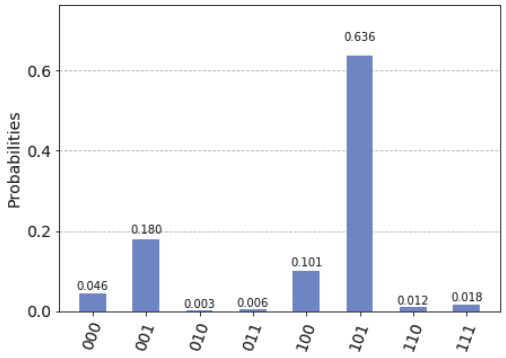

Fig. 9. Probabilities produced trying to guess S={101}.

Algorithm 2. Searching a string

1. Input a string s and split it to characters x
2. For each character x
3. Covert it to number nx
4. Convert it to binary bnx
5. Discover/guess binary number b_isit as described in Algorithm 1
6. Convert it to decimal d_isit
7. Convert it to character is_x

Listing 4: Searching a String

```
# Bernstein - Vazirani algorithm
import binascii
def text_to_bits # converts text to bits
----
def text_from_bits # convert bits to text
----
from qiskit import QuantumCircuit, execute, Aer, IBMQ
%matplotlib inline
from qiskit.tools.visualization import plot_histogram
IBMQ.load_account()
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne')
s = input('Secret string =')
is_s = ''
for x in s:
   bnx = text_to_bits(x)
   n = len(bnx) #number of digits
   temp_n = n//2 # for 5 qubits device
   for i in range(2):
      circuit = QuantumCircuit(temp_n+1, temp_n)
      circuit.h(range(temp_n))
      circuit.x(temp_n)
      circuit.h(temp_n)
      ---- as Listing 3 ---
```

The possibilities obtained are presented in Table 1 (the third column). After that, the same algorithm was applied

after mitigating the noise of the QCD. The results were better (Table 1, the fourth column) and in any case the possibilities obtained for the correct character were always the highest. The results prove that even with the limited reliability of a real quantum device it is rather enough to perform a text-matching technique with success taking under consideration the highest probabilities, which lead to the correct recognition of the "secret" string.

TABLE 1: PREDICTING "HI THERE!" USING REAL IBM QCD

Character	***Bit sequence***	***Probability - (Before calibration)***	***Probability (After calibration)***
H	01001000	0.55859375	0.60775000
i	01101001	0.34863281	0.38523926
<space>	00100000	0,63867188	0.70062305
t	01110100	0.49365234	0.54449854
h	01101000	0.55615234	0.61677295
e	01100101	0.38085938	0.41970703
r	01110010	0.35888672	0.39441650
e	01100101	0.44921875	0.54721875
!	00100001	0.48437500	0.57849219

Another very interesting observation is that the reliability may depend on the complexity of the circuit. The number of ones (1) within a bit sequence determines the number of gates applying to it. So, one can see that the better performance of the algorithm is to characters containing a small number of ones like the space character or 'H' and characters with a larger number of ones like 't' or 'e' present lower possibilities. So, another test was performed predicting a character with many ones versus a character consisting of many zeros. The characters are 'A'={01000001} and 'z'={01111010} and the possibilities obtained were 51% and 37% respectively (using the same IMB real QCD 'ibmq_16_melbourne'. While going to the edges and testing {00000000} and {11111111} the results were 79% and 23% respectively, and all the obtained results lead to the observation that the complexity of the quantum circuit affects its reliability.

VI. CONCLUSION

In this work the reliability of a real quantum device was examined while techniques for noise and error correction were applied to mitigate errors on real quantum computing devices. To test the reliability of a real QCD a string matching algorithm was tested (the Bernstein-Vazirani algorithm) on the IBM Quantum Device – ibmq_16_melbourn. The results obtained were more than promising. The probabilities of guessing the correct character were always the highest, and when error correction and mitigation techniques have been applied, the possibilities of getting the correct character improved.

Future work firstly includes testing and comparing the same techniques on other real devices from other vendors like Rigetti and if possible comparing the time needed to achieve it against a classical computer. The second research direction is to estimate if and how the complexity of a quantum circuit affects its reliability. Increasing the number of applied gates seems to decrease the accuracy of the results.

Since the time complexity of a quantum device strongly depends on the number of available qubits making it highly parallel by its nature, if these qubits will be available and reliable in the future, a new era of computing will arise, completely different from nowadays and probably a revolution in all sciences will be achieved. Sciences will change forever and will meet a revolution for the benefit of mankind.

REFERENCES

[1] D. Poulin, J. Tillich and H. Ollivier, "Quantum Serial Turbo Codes," *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2776-2798, June 2009, doi: 10.1109/TIT.2009.2018339.

[2] C. Boyd, R. Pitaval, Ü. Parts and O. Tirkkonen, "Non-binary classical error-correcting codes for quantum communication," *2015 IEEE International Conference on Communications (ICC)*, London, UK, 2015, pp. 4060-4065, doi: 10.1109/ICC.2015.7248959.

[3] D. Chandra, Z. Babar, S. X. Ng and L. Hanzo, "Near-Hashing-Bound Multiple-Rate Quantum Turbo Short-Block Codes," *IEEE Access*, vol. 7, pp. 52712-52730, 2019, doi: 10.1109/ACCESS.2019.2911515.

[4] Y. Cao, J. Romero, and A. Aspuru-Guzik, "Potential of quantum computing for drug discovery," *IBM Journal of Research and Development*, vol. 62, no, 6, pp. 1-20, 2018.

[5] IMB, "Quantum", visited: 2020, url: https://www.ibm.com/quantum-computing/

[6] Google Research, "Quantum Computing", visited: 2020, url: https://research.google/research-areas/quantum-computing/

[7] Rigetti, "Think Quantum", visited: 2020, url: https://research.google/research-areas/quantum-computing/

[8] Microsoft,: Quantum Computing", visited: 2020, url: https://www.microsoft.com/en-us/quantum

[9] Honeywell, "Honeywell Quantum Solutions", visited 2020, url: https://www.honeywell.com/en-us/company/quantum

[10] Qiskit, "Open-Source Quantum Development", visited: 2020, url: https://qiskit.org/

[11] Rigetti, "PyQuil Documentation", visited: 2020, url: https://docs.rigetti.com/en/stable/

[12] Cirq, 2020, url: https://cirq.readthedocs.io/en/stable/

[13] Microsoft, "<Q#>, visited: 2020, url: https://www.microsoft.com/en-us/quantum/development-kit

[14] Julia, "Julia |Quantum>", visited: 2020, url: https://juliaquantum.github.io/projects/

[15] E. Bernstein and U. Vazirani, "Quantum Complexity Theory," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1411–1473, 1997, doi:10.1137/S0097539796300921

[16] Anaconda Documentation, "Using Jupyter Notebook", visited 2020, url: https://docs.anaconda.com/ae-notebooks/user-guide/basic-tasks/apps/jupyter/

[17] H. N. Hung, P. C. Lee and Y. B. Lin, "Random number generation for residual life of mobile phone movement," *IEEE International Conference on Networking, Sensing and Control, 2004*, Taipei, Taiwan, 2004, pp. 30-33 vol. 1, doi: 10.1109/ICNSC.2004.1297404.

[18] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. 10th Anniversary Edition, 10th ed., USA: Cambridge University Press, 2011.

[19] N. D. Mermin, *Quantum computer science: an introduction*. 1 ed., Cambridge University Press, 2007.

[20] Daniel A. Lidar, and Todd A. Brun, *Quantum Error Correction.* Cambridge University Press, 2013.

[21] A. Peres, "Reversible logic and quantum computers," *Phys. Rev. A*, vol. 32, no. 6, pp. 3266–3276, December 1985.

[22] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol. 52, no. 4, pp. R2493–R2496, October 1995.

[23] H. Abraham et al., "*Qiskit: An Open-source Framework for Quantum Computing*", 2019, doi: 10.5281/zenodo.2562110.

[24] I. K. Savvas, A. V. Chernov and M. A. Butakova, "Experiments with IBM Quantum Devices for Random Number Generation and String Matching," *2020 28th Telecommunications Forum (TELFOR)*, 2020, pp. 1-4, doi: 10.1109/TELFOR51502.2020.9306624.