COMPARISON OF CONTAINER-BASED PLATFORMS FOR QUANTUM COMPUTING SIMULATION

Ginés Carrascal

IBM

Sta. Hortensia 26-28,
Madrid, Spain
gines_carrascal@es.ibm.com

Guillermo Botella
Alberto A. Del Barrio

Dept. of Computer Architecture and Automation
Universidad Complutense de Madrid

Prof. José García Santesmases 9,
Madrid, Spain
{gbotella, abarriog }@ucm.es

ABSTRACT

Quantum Computing allows for substantial speed-ups e.g. for integer factorization or database search, compared to conventional computation. However, since real quantum computers are an emerging technology, a significant amount of research in this domain still relies on simulations of quantum computations on conventional machines. This paper proposes a containerized environment to simulate quantum circuits, making this environment quick and easy to share within research teams. Also test the environment implementing an arithmetic circuit (Vedral's adder) and compares the performance of two of the current more used environments: IBM QISKit and Google Cirq.

Keywords: Quantum Computing, Container, performance analysis.

1 INTRODUCTION

Computer engineers are already interacting with quantum computers. These new and modern technologies present challenges to computer engineering students and practitioners (ACM / IEEE, 2016). Quantum computing is already a reality. The arrival of the first commercial quantum computer, the IBM Q System One, has shown us that this discipline has reached a maturity that was difficult to imagine no more than five years ago (IBM, 2019).

2020 is being a very busy year in terms of quantum computing, and we have reason to anticipate that very important achievements in this discipline will come in the coming months thanks, above all, to the efforts of companies such as IBM, Google or Microsoft, and to a lesser extent Amazon and China's Alibaba. Governments, particularly those of the USA, the European Union and China, are funding work in the area with the concern that quantum computers may give the country that gets there first a major advantage. For example, the 2019 U.S. National Quantum Initiative Act authorized $1.2 billion funding over the next 5-10 years (Smith, 2018).

Quantum computation has theoretically been proven to be superior to conventional computation for important applications. For example, quantum algorithms for integer factorization - Shor's algorithm (Shor, 1994) or database search - Grover's Search (Grover, 1996) have been proposed that lead to significant, sometimes even exponential, speedups compared to conventional computations.

Real quantum computing hardware has been in the recent years. The first publicly available quantum processor has been made accessible by IBM through their project IBM Quantum Experience (IBM 2020). Via IBM's cloud infrastructure, the community can access a quantum processor with 5 qubits (launched in March 2017) and 16 qubits (launched in June 2017), respectively, to conduct experiments. IBM further plans to increase the number of available qubits to 53 – similar to Google's plans to provide a quantum chip with 53 qubits that demonstrates quantum supremacy (Pednault et al., 2019).

However, thus far, real quantum computers remains an emerging technology. This requires, besides others, that respective developments have to be conducted while still relying on conventional technologies. In particular, this is an issue when it comes to simulating quantum computations or corresponding quantum algorithms. Although these quantum computations describe approaches to solve several problems significantly faster than a conventional technology, they still have to be simulated on conventional machines thus far. Furthermore, simulation plays an important role in the verification of existing and future quantum computers.

This paper reports on a prototype docker container environment that can be installed on any container platform, from simple local Docker to more advanced cloud container platforms using Kubernetes. Section 2 describes the experimental frameworks used to evaluate the performance of containerized quantum platforms. Section 3 introduces metrics used throughout this paper and addresses the implementation of the testbenches. Sections 4 describes the proposed containerized environment. Section 5 presents the results from the tests. Section 6 presents our conclusions.

2 CONSIDERED ENVIRONMENTS

There are many alternatives for programming quantum computers, including Qiskit (*Qiskit*, n.d.), Cirq (Cirq, n.d.), pyQuil (*PyQuil documentation*, n.d.), Q# (*Quantum Development Kit | Microsoft*, n.d.), and ProjectQ (damian_projectq, n.d.). Qiskit, Cirq or pyQuil are better suited for using real quantum devices. These three frameworks have been developed by companies that have achieved real quantum computers and have been tested and evolved around the real needs of the interaction with the hardware.

For this work we have chosen Qiskit and Cirq, because they provide both a 100% python based local quantum simulator (PyQuil is a python library that calls an independent C quantum virtual machine).

We briefly describe Qiskit and Cirq the selected quantum environments for this comparison, addressing their respective strengths.

2.1 IBM's Quantum Experience

IBM, through its *IBM Quantum Experience initiative* (*IBM Quantum Experience*, n.d.), has been providing any user with remote access to multiple quantum computers based on superconducting technology since 2016. It is the first platform that allowed access to its computers and on which more experiments have been run to date.

Three computers, two 5 *qubits (Yorktown and Tenerife),* as well as a 16 qubit third *(Rueschlikon)*, are available free of charge. The service can be accessed with the same software for using commercial computers of 20 and 53 qubits.

Qiskit is an Open Source project developed in Python. It includes several separate modules:

- Terra (*Qiskit Terra | A solid foundation for quantum computing*, n.d.): The basic module, which imports all translation functionalities to OpenQASM[16] for interaction with quantum computers.
- Ignis (*Qiskit Ignis | Understanding and mitigating noise in quantum systems.*, n.d.): Provides tools for noise characterization, hardware parameterization, etc.
- Aer (*Qiskit Aer | A high performance simulator framework for quantum circuits*, n.d.): Implements a simulator with noise modeling.
- *Ibmq Provider* (*IBM Q Account | Access to world-leading quantum systems and simulators.*, n.d.)*: Allows* access to IBM remote quantum computers and dimulators.
- Aqua (*Qiskit Aqua | Algorithms for quantum computing applications*, n.d.): Provides a layer of abstraction of quantum gates, allowing the application of already imported quantum algorithms to higher level applications.

Qubits are declared in "quantum registers" and classical bits in what they have called "classical registers". Such classical registers are used to accommodate the result of applying a measure on a quantum register.

All quantum operations to be applied on one or more quantum records are encompassed in a "quantum circuit". It is useful to have circuits as independent functional units, because you can operate them separately, thus being able to add them and build more complex circuits, etc.

2.2 Google's Cirq

Currently in the alpha phase, Cirq is the open source system designed by Google for the design and simulation of quantum circuits. It is a Python library specifically designed for NISQ (Noisy Intermediate-Scale Quantum) systems. As stated in the introduction to such a platform, Cirq attempts to expose the details of the hardware to the programmer rather than creating an abstraction layer, claiming that the control of the details ultimately determines whether a circuit is feasible to work with.

The simulator, which is capable of simulating up to 25 qubits, is integrated into the software and runs locally. Google also has two real quantum computers available, FoxTail and Bristlecone, which are only accessible upon invitation from the platform.

The operations implemented by Cirq are aimed at programming functions that design circuits, which will be introduced as an input argument to the simulator, quantum computer or other algorithm analysis system.

3 TESTING THE SELECTED SIMULATORS

3.1 The Quantum Circuit

A comparison of the most suitable platforms was carried out through a quantitative performance analysis. We perform the simulation of an arithmetic circuit, specifically the adder circuit proposed in Vedral, Barenco & Ekert (Vedral et al., 1996), and applied successively to add numbers from 1 to 8 bits.
This is a linear-depth ripple-carry quantum addition circuit. Previous addition circuits required many ancillary qubits linearly; the aforementioned adder uses only a single ancillary qubit. Also, it has less depth and fewer gates than previous ripple-carry adders. The objective was to simulate the same circuit on each platform, extracting metrics from the circuit created and the results obtained, in order to be able, through the same parameters extracted from different platforms, to compare objective values.
We have selected this kind of circuit for the comparison because of its O(n) complexity in the number of gates with the size in bits of the numbers added, so been able to create small enough circuits to perform this tests in the local simulator.
The implementation of the adder circuit begins by first implementing the carry operation, going on to calculate and store the most significant digit of the sum. The carry operation is then undone, applying the same gates in reverse order on each qubit, in order to retrieve the initial state, allowing us to implement the sum operation on the initial values. Finally, the sum is implemented, applying it bitwise from the least significant to the most significant qubit. As shown in Figure 1(Qiskit implementation for 4 bits), qubits from q0_0 to q0_3 are used for coding the first summing, q1_0 to q1_4 are used both for the second summing and the result. The rest are the ancillas and the classical registers to read the result.

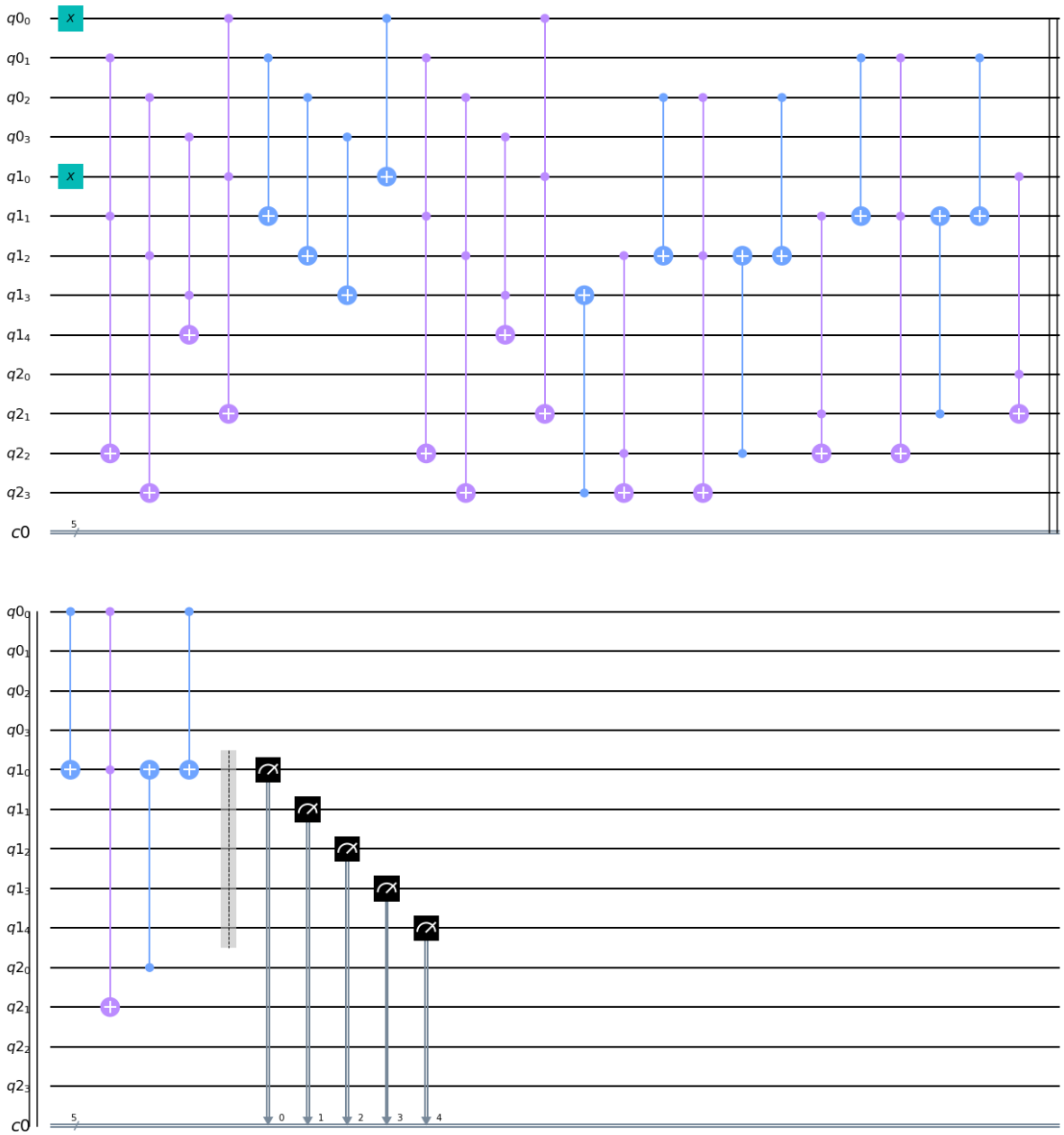

Figure 1: Adder circuit for 4 bit numbers implemented in Qiskit. Source: own compilation

Figure 2 shows the same circuit using Cirq. The picture is more compact than the Qiskit one but is more difficult to follow by non-experts because of the lack of graphical capabilities of the library.

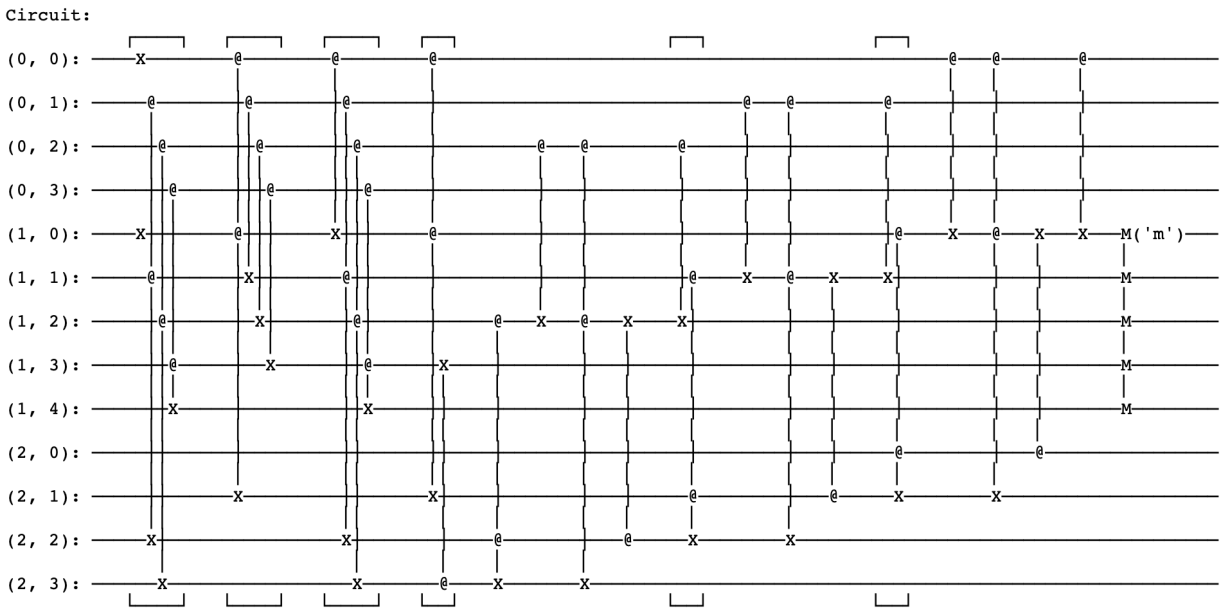

Figure 2: Adder Circuit for 4 bit numbers implemented in Cirq. Source: own compilation

3.2 Performance Test

A python test program was developed to test both environments in the same circumstances.

Firs, two functions specific to each library were defined, one to create the circuit (see Appendix A) and other to perform a simulation.

The following test sequence was used, from 1 to 8 bit numbers: Create the circuit, measure the execution time of 100 simulations adding 1+1:

```
import timeit

times=[]

for i in range(1,9):

    s = f"""\

from __main__ import get_adder

from __main__ import simulate

circuit = get_adder({i},"{"0"*i}1","{"0"*i}1")

"""

    times.append(timeit.timeit('simulate(circuit)', setup=s, number=100))

    print(i,times[-1])


print(times)
```

8 bit numbers is our limit because it is necessary to include 3n+1 = 25 qubits to implement the adder circuit.

4 CONTAINERIZED ENVIRONENT

Container technologies offer the possibility of packaging application codes and all their dependencies and then agilely launching many instances with different parameters and variations of that application. Containers are portable, scalable and are standardized so that they can easily and rapidly be deployed in a cloud-based environment and have the additional advantage of quicker processing speed when compared to VMs (Varghese et al., 2016).

Containers also have the feature that they can run on the same machine and share the OS kernel with other containers, each running as isolated process in the user space. Containers can also offer the advantage that they can isolate the software from its environment in a way that ensures that the container incorporates portability and performs uniformly across many different platforms.

Docker was selected as container software for building this prototype (Boettiger, 2015). This software has been thoroughly tested and is also a stable production level container technology (Rad et al., 2017). The fundamental building block of this Docker technology is the container image. The Docker image creates a docker container. The image is a lightweight, stand alone, executable package that includes the code, runtime, system tools, system libraries and settings needed to run an application.

The selected base image was Jupyter Notebook (*Project Jupyter*, n.d.). The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Two Docker images where created, one for Qiskit and other for Cirq, including all the needed dependencies and ready to run.

Once instantiated, each container provides a URL to access the environment from any web browser. This point is key to enable both the local installation and also a cloud deployment where a research team can share their programs and computational power in a centralized way.

5 RESULTS

The described test was performed in a MacBook Pro (16-inch, 2019), Processor: 2,4 GHz 8-Core Intel Core i9, Memory: 32 GB 2667 MHz DDR4.

Each container was configured with 8 CPUs, 2 GB of memory and 1GB of swap in the Docker environment.

Table 1 shows the execution time of 100 simulations of the adder circuit implemented for different amount of qubits

Table 1: Time measure for increasing number of qubits

Summing bits	Total qubits	Qiskit Time	Cirq Time
1	4	2.794548899983056	0.159240199951455
2	7	2.8395655000349507	0.2790897999657318
3	10	2.76701419998426	0.4256545000243932

4	13	2.7808956000953913	0.605262600001879
5	16	2.6760485999984667	1.0336101999273524
6	19	2.6706397000234574	4.1191501000430435
7	22	2.69583350000903	29.681731199962087
8	25	2.693196399952285	310.2610318000661

To facilitate the comparison, the results are plot in Figure 3, in logarithmic scale.

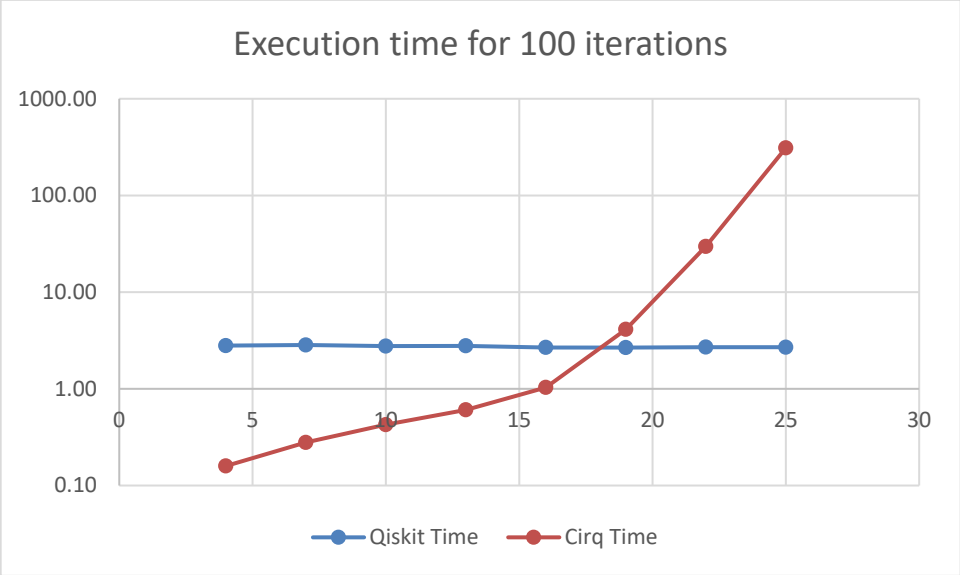

Figure 3: Comparison of execution time of 100 iterations in logarithmic scale

The data in Table 1 and Figure 3 illustrates the importance of testing the performance change related to the number of qubits for Quantum simulators. Due to the nature of quantum computing, simulating it with classical computers, at the end will need exponential resources as the number of qubits grow.

To compare different simulators, more important than the punctual performance is the stability of the performance in the range of qubits that is researchable for the simulator. Indeed, the maximum number of qubits that the simulator can manage with a determined hardware resources may vary substantially depending on this measure.

On a real quantum computer (IBMQ), the execution time is more dependant in queue time than in the quantum processor time. Single qubit gates are approximately 10 ns, and two qubit gates 100 ns.

Here is a rough breakdown of the processing time for a quantum circuit like this:

- Loading the experiment into the instruments that create the pulses (~ 15s)
- 1024 repetitions (shots) of running calibration pulses & circuit (~ 5s)
    - Reset qubits (relaxation) + calibration: ~ 4ms
    - Reset qubits (relaxation) + circuit: ~ 1ms

6 CONCLUSIONS

In this paper we have proposed an easy to use and to share containerized environment for Quantum computing Simulation. This kind of environments

Using this environment and an arithmetic circuit (Vedral adder), equivalent implementations have been created in Qiskit and Cirq. These implementations have been used to test the performance of both frameworks in equal conditions, increasing the amount of qubits from 4 to 25.

We believe that it is interesting using real useful algorithms to test the performance rather than only aleatory circuits for the sake of generality and clarity.

Analyzing the time measures, Qiskit presents a high performance simulator, sustaining a stable time of simulation with the increase of qubits. Although Cirq shows smaller times with few qubits, it presents a clear exponential degradation with the amount of qubits, as expected in a direct implementation, therefore, not suitable for high volume simulations.

In the future we should extend this work in two ways: Firstly, implementing more arithmetic circuits and secondly, creating more containerized environments for other quantum computing simulator frameworks.

ACKNOWLEDGMENTS

This paper has been supported by the CM under grant S2018/TCS-4423,the EU (FEDER) and the Spanish MINECO under grant RTI2018-093684-B-I00 and by Banco Santander under grant PR26/16-20B-1and by PID 2018-2020 (UCM) Innova Docentia, PR 209.

A APPENDICES

Qiskit implementation of the Vedral adder circuit:

```
def get_adder(n, sumando_1, sumando_2):
    a = QuantumRegister(n)
    b = QuantumRegister(n+1)
    c = QuantumRegister(n)
    resultado = ClassicalRegister(n+1)

    qc = QuantumCircuit(a,b,c,resultado)

    for i in range(n):
        if sumando_1[i] == "1":
            qc.x(a[n - (i+1)])
    for i in range(n):
        if sumando_2[i] == "1":
```

```
            qc.x(b[n - (i+1)])

    for i in range(n-1):
        qc.ccx(a[i], b[i], c[i+1])
        qc.cx(a[i], b[i])
        qc.ccx(a[i], b[i], c[i+1])

    qc.ccx(a[n-1], b[n-1], b[n])
    qc.cx(a[n-1], b[n-1])
    qc.ccx(a[n-1], b[n-1], b[n])

    qc.cx(c[n-1], b[n-1])

    for i in range(n-1):
        qc.ccx(c[(n-2)-i], b[(n-2)-i], c[(n-1)-i])
        qc.cx(a[(n-2)-i], b[(n-2)-i])
        qc.ccx(a[(n-2)-i], b[(n-2)-i], c[(n-1)-i])

        qc.cx(c[(n-2)-i], b[(n-2)-i])
        qc.cx(a[(n-2)-i], b[(n-2)-i])

    qc.barrier(b)
    qc.measure(b,resultado)

    return qc



def simulate(circuit):
    my_backend = Aer.get_backend("qasm_simulator")
    job = execute(qc, my_backend, shots=20)
    job_stats = job.result().get_counts()

    return job_stats
```

Cirq implementation of the Vedral adder circuit:

```
def get_adder(n, sumando_1, sumando_2):
    a = [cirq.GridQubit(0, i) for i in range(n)]
    b = [cirq.GridQubit(1, i) for i in range(n+1)]
    c = [cirq.GridQubit(2, i) for i in range(n)]

    # Create a circuit
    circuit = cirq.Circuit()

    for i in range(n):
        if sumando_1[i] == "1":
            circuit.append(X(a[n - (i+1)]))
    for i in range(n):
        if sumando_2[i] == "1":
            circuit.append(X(b[n - (i+1)]))

    for i in range(n-1):
        circuit.append(CCX(a[i], b[i], c[i+1]))
        circuit.append(CX(a[i], b[i]))
        circuit.append(CCX(a[i], b[i], c[i+1]))

    circuit.append(CCX(a[n-1], b[n-1], b[n]))
    circuit.append(CX(a[n-1], b[n-1]))
    circuit.append(CCX(a[n-1], b[n-1], b[n]))

    circuit.append(CX(c[n-1], b[n-1]))

    for i in range(n-1):
        circuit.append(CCX(c[(n-2)-i], b[(n-2)-i], c[(n-1)-i]))
        circuit.append(CX(a[(n-2)-i], b[(n-2)-i]))
        circuit.append(CCX(a[(n-2)-i], b[(n-2)-i], c[(n-1)-i]))

        circuit.append(CX(c[(n-2)-i], b[(n-2)-i]))
        circuit.append(CX(a[(n-2)-i], b[(n-2)-i]))
```

```
        circuit.append(cirq.measure(*b, key='m'))  # Measurement.

        return circuit


    def simulate(circuit):
        simulator = cirq.Simulator()
        result = simulator.run(circuit, repetitions=20)

        frequencies = result.histogram(key='m', fold_func=bitstring)
        return frequencies
```

REFERENCES

Announcing Cirq: An Open Source Framework for NISQ Algorithms. (n.d.). Google AI Blog. Retrieved January 28, 2020, from http://ai.googleblog.com/2018/07/announcing-cirq-open-source-framework.html

Association for Computing Machinery [ACM] / Institute of Electrical and Electronics Engineers [IEEE]. (2016). Computer Engineering Curricula 2016. In Computer Engineering Curricula 2016. CE2016. Curriculum Guidelnes for Undergraduate Degree Programas in Computer Engineering. IEEE, ACM. https://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf

Boettiger, C. (2015). An introduction to Docker for reproducible research, with examples from the R environment. ACM SIGOPS Operating Systems Review, 49(1), 71–79. https://doi.org/10.1145/2723872.2723882

damian_projectq. (n.d.). ProjectQ. ProjectQ. Retrieved January 28, 2020, from https://projectq.ch/

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. ArXiv:Quant-Ph/9605043. http://arxiv.org/abs/quant-ph/9605043

IBM Q Account | Access to world-leading quantum systems and simulators. (n.d.). Retrieved January 30, 2020, from https://qiskit.org/ibmqaccount

IBM Quantum Experience. (n.d.). IBM Quantum Experience. Retrieved January 28, 2020, from https://quantum-computing.ibm.com

IBM Unveils World's First Integrated Quantum Computing System for Commercial Use—Jan 8, 2019. (n.d.). Retrieved January 26, 2020, from https://newsroom.ibm.com/2019-01-08-IBM-Unveils-Worlds-First-Integrated-Quantum-Computing-System-for-Commercial-Use

Pednault, E., Gunnels, J., Maslov, D., and Gambetta, J. (2019, October 21). On "Quantum Supremacy." IBM Research Blog. https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/

Project Jupyter. (n.d.). Retrieved March 4, 2020, from https://www.jupyter.org

Qiskit. (n.d.). Retrieved January 28, 2020, from https://qiskit.org/

Qiskit Aer | A high performance simulator framework for quantum circuits. (n.d.). Retrieved January 30, 2020, from https://qiskit.org/aer

Qiskit Aqua | Algorithms for quantum computing applications. (n.d.). Retrieved January 30, 2020, from https://qiskit.org/aqua

Qiskit Ignis | Understanding and mitigating noise in quantum systems. (n.d.). Retrieved January 30, 2020, from https://qiskit.org/ignis

Qiskit Terra | A solid foundation for quantum computing. (n.d.). Retrieved January 30, 2020, from https://qiskit.org/terra/

Quantum Development Kit | Microsoft. (n.d.). Microsoft Quantum - US (English). Retrieved January 28, 2020, from https://www.microsoft.com/en-us/quantum/development-kit

Rad, B. B., Bhatti, H. J., and Ahmadi, M. (2017). An Introduction to Docker and Analysis of its Performance. 8.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. Proceedings 35th Annual Symposium on Foundations of Computer Science, 124–134. https://doi.org/10.1109/SFCS.1994.365700

Smith, L. (2018, December 21). H.R.6227 - 115th Congress (2017-2018): National Quantum Initiative Act [Webpage]. https://www.congress.gov/bill/115th-congress/house-bill/6227

Varghese, B., Subba, L. T., Thai, L., and Barker, A. (2016). Container-Based Cloud Virtual Machine Benchmarking. 2016 IEEE International Conference on Cloud Engineering (IC2E), 192–201. https://doi.org/10.1109/IC2E.2016.28

Vedral, V., Barenco, A., and Ekert, A. (1996). Quantum networks for elementary arithmetic operations. Phys. Rev. A, 54(1), 147–153. https://doi.org/10.1103/PhysRevA.54.147

Welcome to the Docs for the Forest SDK! —PyQuil 2.16.0 documentation. (n.d.). Retrieved January 28, 2020, from http://docs.rigetti.com/en/stable/#

AUTHOR BIOGRAPHIES

GINÉS CARRASCAL received the M.A. Sc. degree in Physics in 1999, from the University of Salamanca, Spain. After post graduate studies joined IBM in 2000 as IT Architect, and currently working as Quantum Ambassador. Since 2017, he has been an Interim Assistant Professor of Computer Science with the Department of Software Systems and Computation, UCM. His research interests include applied artificial intelligence and Quantum Computing. His email address is gines_carrascal@es.ibm.com.

GUILLERMO BOTELLA received the M.A. Sc. degree in Physics in 1998, the M.A.Sc. degree in Electronic Engineering in 2001 and the Ph.D. degree in 2007, all from the University of Granada, Spain. Currently he is a Full Professor at the Department of Computer Architecture and Automation of Complutense University of Madrid, Spain. His current research interests include Digital Signal Processing for VLSI, FPGAs, GPUs, HPC and Vision Algorithms. His email address is gbotella@ucm.es.

ALBERTO A. DEL BARRIO received the Ph.D. degree in Computer Science from the Complutense University of Madrid (UCM), Madrid, Spain, in 2011. Since 2017, he has been an Interim Associate Professor of Computer Science with the Department of Computer Architecture and System Engineering, UCM. His

research interests include Design Automation, Arithmetic as well as Video Coding Optimizations. His email address is abarriog@ucm.es.