DEGREE PROJECT IN COMPUTER SCIENCE,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2018

4-qubit Grover's algorithm implemented for the ibmqx5 architecture

VERA BLOMKVIST KARLSSON

PHILIP STRÖMBERG

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Page intentionally left blank.

4-qubit Grover's algorithm implemented for the ibmqx5 architecture

PHILIP STRÖMBERG

VERA BLOMKVIST KARLSSON

Degree project in computer science
Date: June 6, 2018
Supervisor: Stefano Markidis
Examiner: Örjan Ekeberg
School of Electrical Engineering and Computer Science
Swedish title: 4-qubit Grovers algoritm implementerad för ibmqx5

Abstract

Quantum computing is a quickly growing field of research thanks to recent hardware advances. The quantum mechanical properties of quantum computers allow them to solve certain families of problems faster than classical computers. A quantum algorithm solving such a problem is Grover's algorithm, which finds an element in an unordered set faster than any classical search algorithm. In this paper an implementation of a 4-qubit Grover's algorithm for the IBM Q computer ibmqx5 is presented. Executing the implementation on an ibmqx5 simulator yield results in line with the theoretically optimal results. The accuracy of the ibmqx5 simulation results compared to the ibmqx5 execution results suggests that current hardware is not yet suitable,due to required complexity of the circuits for a 4-qubit Grover implementation.

Sammanfattning

Kvantteknik är ett snabbt växande forskningsområde tack vare att nödvändig hårdvara förbättrats i rask takt. De kvantmekaniska egenskaperna hos kvantdatorer tillåter vissa familjer av problem att lösas snabbare på dessa än på klassiska datorer. En algoritm för kvantdatorer som löser ett sådant problem är Grover's algoritm, vilken hittar ett element i en oordnad mängd snabbare än vad någon sökalgoritm för klassiska datorer gör. I denna rapport presenteras en implementation av en 4-qubit Grover's algoritm för IBM Q-datorn ibmqx5. Vid exekvering av implementationen på en ibmqx5-simulator erhålls resultat som är i linje med teoretiskt optimala resultat. En jämförelse av korrektheten hos resultaten från ibmqx5-simulatorn och resultaten från ibmqx5-datorn tyder på att nuvarande hårdvara inte är lämplig för kretsar av komplexiteten nödvändig för en 4-qubit Grover implementation.

Contents

Chapter 1

Introduction

Quantum computing was first theorized in the 1980s and is today a quickly growing field thanks to recent hardware advances. Where a classical computer uses binary bits to store its information, a quantum computer uses quantum bits, or qubits. The unique quantum characteristics of qubits allow quantum computers to solve certain families of problems faster than classical computers, such as searching a database [1] and prime factorization [2]

An algorithm that solves such a problem is Grover's algorithm. Grover's algorithm is a quantum algorithm that finds an element in an unordered set of size N in $\mathcal{O}(\sqrt{N})$ time. Finding an element in an unordered set on a classical computer would take on average $\frac{N}{2}$ time, i.e. the sought element would be found in $\mathcal{O}(N)$ time [1].

IBM have several quantum computers available to the public through their cloud service IBM Q. The ibmqx5 is a superconductivity-based 16-qubit quantum computer available through a Python-based programming interface called QISKit. The interface also provides access to an ibmqx5 simulator where simulation runs can be performed.

1.1 Problem statement

Various implementations of Grover's algorithm exist, such as a 2-qubit implementation on the ibmqx2 architecture [3] and a 3-qubit implementation on a trapped-ion architecture [4]. Currently no implementation of a 4-qubit Grover's algorithm has been published for the ibmqx5 architecture.

1.2 Purpose and scope

The purpose of this report is to provide an implementation of Grover's algorithm for a search space of 4 qubits for the ibmqx5 architecture. The implementation does not make use of ancilla bits and uses only single solution oracles. The algorithm is implemented using the QISKit programming interface.

Chapter 2

Background

This chapter provides an introduction to the quantum computing subject and explanations of the related concepts necessary for understanding the work presented in the report. A brief description of the ibmqx5 architecture and the QISKit programming interface is given. Finally Grover's algorithm and its stages are explained.

2.1 Qubit

The unit of information in a quantum computer is called a quantum bit or *qubit*. In contrast to classical bits, or *cbits*, the value of qubits are not as easily defined as those of cbits. Where a cbit either has the value of 0 or 1, a qubit can exist in a superposition of the two values. Consequently a qubit can't be said to have an actual value, but rather a probability to be found in a certain state when measured. The measuring of qubits is performed as the final step of a computation to produce an output in cbits. This destroys the quantum state.

A qubit can be described as a unit vector in a 2D complex vector space $\mathbb{C}^2$. In this paper the ket notation will be used for describing the qubit vectors. A qubit in the zero state is written as $|0\rangle$ and a qubit in the one state as $|1\rangle$. $|0\rangle$ and $|1\rangle$ are basis vectors in the 2D complex vector space $\mathbb{C}^2$.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad\qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

A Bloch sphere, seen in figure 2.1, can be used as a geometrical representation of the qubit. The Bloch sphere is the space of all rays in $\mathbb{C}^2$ [5]. The north pole of the sphere represents the $|0\rangle$ state and the south pole represents the $|1\rangle$ state. A state $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ is represented as the point (θ, ϕ)

on the sphere, where $\alpha_0 = \cos\frac{\theta}{2}$ and $\alpha_1 = e^{i\phi}\sin\frac{\theta}{2}$ [6]. The Bloch sphere can be useful for visualizing the state of a qubit and the effect of transformations performed on the qubit.

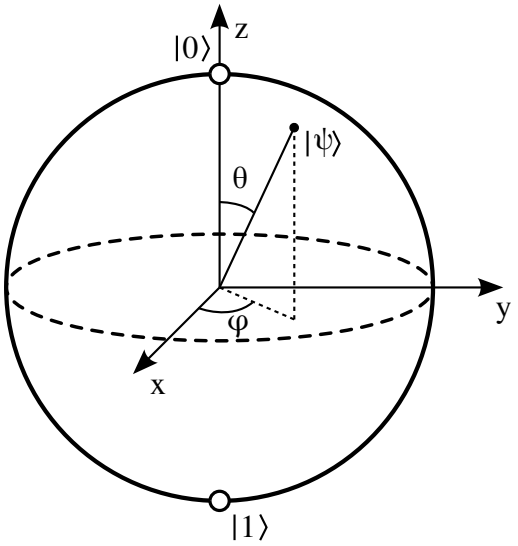


Figure 2.1: Bloch sphere [7]

2.2 Probability amplitude

A qubit can be described by the 2D vector $\begin{pmatrix}\alpha_0\\ \alpha_1\end{pmatrix}$ where α_0 and α_1 are complex numbers [8]. The first value α_0 represents the *probability amplitude*, or *amplitude*, of the $|0\rangle$ state. The second value α_1 represents the amplitude of the $|1\rangle$ state. Classical probability can be calculated from amplitude with the Born rule [9]:

$$p(x) = |\alpha_x|^2$$

where $p(x)$ is the classical probability for state x occurring. This also gives rise to the normalization condition:

$$|\alpha_0|^2 + |\alpha_1|^2 = 1$$

Quantum systems can be written as a linear combination of the basis vectors to illustrate the superposition property [8]. For systems consisting of multiple qubits this can be a clearer way of expressing the quantum state:

$$\alpha_{00} * |00\rangle + \alpha_{01} * |01\rangle + \alpha_{10} * |10\rangle + \alpha_{11} * |11\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix}$$

The fact that α_x is a complex number means it is possible to model the effects of interference and superposition on qubits. Where addition of real probabilities always give rise to an increased probability, addition of amplitudes can instead lead to a smaller total probability. This corresponds to the quantum mechanical property of interference [8].

2.3 Quantum gates

Quantum gates can be represented as matrices describing transformations on qubits. The normalization condition $|\alpha_0|^2 + |\alpha_1|^2 = 1$ must hold for all states, so all actions performed on qubits must give rise to unitary vectors [6]. The matrices describing such transformations are unitary matrices, i.e. the matrix conjugate transpose is its inverse [2]. The conjugate transpose of a Matrix $\mathbf{M}$ is denoted as $\mathbf{M}^\dagger$, pronounced "M dagger".

Since unitary matrices always have an inverse all transformations described by them are reversible. Consequently all quantum gates must be reversible (except the measurement gate which destroys the quantum state), meaning that the circuit must be able to be run in reverse.

Quantum circuit notation

A quantum circuit is represented by a diagram in which each line represents the timeline of a qubit, read from left to right. A gate acting on a qubit is denoted by the symbol of the gate placed on the qubit it is acting on. When describing quantum gates below, the corresponding circuit representation will be shown together with the matrix representation.

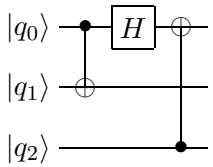


Figure 2.2: Example 3-qubit quantum circuit

Pauli gates

There are three types of Pauli gates, the **X**-, **Y**- and **Z**-gate. The gate rotates the qubit around the named axis in the Bloch sphere by π. The **X**-gate is called bit-flip and the **Z**-gate phase-flip. The **Y**-gate is both a phase- and bit-flip and satisfies $\mathbf{Y} = i\mathbf{XZ}$. [9]

$$\mathbf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad |q_0\rangle - \boxed{X} -$$

$$\mathbf{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad |q_0\rangle - \boxed{Y} -$$

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad |q_0\rangle - \boxed{Z} -$$

T-gate

The **T**-gate is a phase shift gate related to the Pauli **Z**-gate. $\mathbf{T}^4 = \mathbf{Z}$, meaning that performing a **T**-gate four times will yield the same result as applying a **Z**-gate once [10]. The **T**-gate corresponds to a rotation of $\frac{\pi}{4}$ around the Z-axis in the Bloch sphere.

$$\mathbf{T} = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \qquad |q_0\rangle - \boxed{T} -$$

Hadamard gate

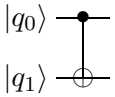
The Hadamard gate performs a rotation π about the X-axis and $\frac{\pi}{2}$ about the Y-axis in the Bloch sphere. This transformation takes X to Z and Z to X. The gate is among other things used to put the target qubit into a superposition state, having an equal chance of being measured as 0 or 1. [9, 10]

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad |q_0\rangle - \boxed{H} -$$

Controlled gate

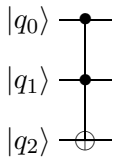
A qubit can be added as a control bit to any gate, so that its operation will only be executed on the target qubit if the control bit is a one. To indicate a controlled gate, a **c** is added to the gate's name. Common examples of controlled gates are the controlled not (**cNOT**) and controlled swap (**cSWAP**). A **cNOT** gate is equal to a **cX**-gate. In the diagram below $|q0\rangle$ would be the control qubit and $|q1\rangle$ the target qubit.

$$\textbf{cNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$



Toffoli gate

A double controlled not gate is called a Toffoli gate. The Toffoli, **ccNOT**, and **ccX** gates are equivalent. In the diagram $|q2\rangle$ is the target qubit.



U-gate

The **U**-gate is a gate for general rotations along the three axes, taking one to three arguments. In this paper the **U1**-gate will be used, whose only argument denotes rotation around the Z-axis in the Bloch sphere.

Measurement gate

A measurement gate is used to project a qubit's state onto the basis vectors $|0\rangle$ and $|1\rangle$. This step is necessary for extracting a result from the quantum computation and is the only non-reversible quantum gate. Once a qubit is measured its quantum state is destroyed.

2.4 ibmqx5

The ibmqx5 is a 16-qubit quantum computer from IBM available to the public as a part of the IBM Q cloud service. It is accessible through a programming interface called QISKit [11].

The computer uses a superconductivity transmon qubit implementation [12, 13] which has consequences for algorithm design. The possible interactions between qubits are governed by the superconducting bus connections between them [14], and can be described by the ibmqx5 *coupling map*. In the coupling map in figure 2.3 an arrow from qubit A to qubit B represents that a **cNOT** gate can be created with A as the control bit and B as target.

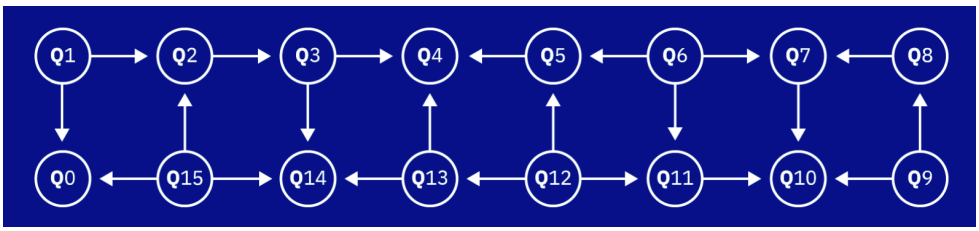


Figure 2.3: Coupling map of ibmqx5 [15]

An alternative representation of the coupling map is

$$\{1 : [0, 2], 2 : [3], 3 : [4, 14], 5 : [4], 6 : [5, 7, 11], 7 : [10], 8 : [7],$$
$$9 : [8, 10], 11 : [10], 12 : [5, 11, 13], 13 : [4, 14], 15 : [0, 2, 14]\}$$

where {A: [B]} have the relationship previously described.

2.5 QISKit

QISKit is a Python-based programming interface for programming quantum computers [11]. The interface provides access to the quantum computers available though IBM Q and also provides access to a local simulator for making simulation runs. The simulator can be configured to have a coupling map equal to that of ibmqx5.

The QISKit interface provides a built in mapper for mapping a qubit in the code to a hardware qubit [16]. Because of this the qubits can be arbitrarily named in the QISKit code. The mapper works provided that the requested implementation can be made to fit the coupling map.

The gates relevant to this report that are available through the QISKit programming interface are the Pauli, Hadamard, **cNOT**, Toffoli, **T**, and **U1** gates. Controlled versions of the Pauli and **U1** gates also exist [16]. To use gates other than those described above they need to be constructed from the available gates.

QISKit has an upper limit on the amount of circuits that can be included in an implementation. The current limit is 75 circuits.

2.6 Grover's algorithm

Grover's algorithm is a quantum search algorithm that runs in $\mathcal{O}(\sqrt{N})$ time over N unsorted elements. Besides searching for an element, Grover's algorithm can be used for inverting a function [17] and solving the collision problem [18]. Given an unordered set of $N = 2^n$ states $X = \{x_1, x_2, ..., x_N\}$ and a binary function $f : \{0, 1\}$ the algorithm will find a state x' such that $f(x') = 1$.

2.6.1 The stages of the algorithm

The algorithm is divided into four stages [1].

Initialization

All qubits are set to be in superposition. All states have the amplitude $\frac{1}{\sqrt{N}}$.

Oracle

The oracle function marks the state x' that satisfies the condition $f(x') = 1$ by performing a phase flip. All other states are left unaltered. The operation has the effect of inverting the state's amplitude and it runs in constant time.

Amplification

The amplification stage phase flips the amplitudes around the average amplitude. As the target state's amplitude was inverted while the other states kept their original amplitudes, the flip causes the target state's amplitude to increase and the others to decrease.

Measurement

The qubits are read and output given.

2.6.2 General result probabilities

The probability of finding the t results among N elements in one iteration can be calculated from equation 2.1 [19]. Iterations of Grover's algorithm refers to the number of times that the Oracle and Amplification stages are performed. Each iteration of the algorithm increases the amplitude of the sought state by $\mathcal{O}(\frac{1}{\sqrt{N}})$ [1].

$$t * \left(\frac{\frac{N-2t}{N} + 2\frac{N-t}{N}}{\sqrt{N}} \right)^2 \tag{2.1}$$

On a 4 qubit computer $4^2 = 16$ elements are possible, giving us $N = 16$:

Single solution ($t = 1$) $\quad 1 * \left(\frac{\frac{16-2}{16} + 2\frac{16-1}{16}}{\sqrt{16}} \right)^2 = \frac{121}{256} \approx 47.27\%$

2.7 Unreliability

2.7.1 Gate error

Each quantum gate introduces a small error to the quantum state when it is applied. Because of the hardware implementation and layout of the coupling map each qubit interacts differently with gates and other qubits. This creates varying error sizes for the different qubits. On the ibmqx5 the multi-qubit gate and readout error is always larger than the single-qubit gate error [15].

2.7.2 Quantum decoherence

Quantum decoherence means a loss of quantum mechanical properties in a system due to its interaction with the environment [20]. The effect increases with the size of the system, i.e. the number of qubits [21]. Decoherence is described by the two phenomena *energy relaxation* and *dephasing* [22].

Energy relaxation refers to a qubit decaying from the high-energy state $|1\rangle$ to the low energy state $|0\rangle$ over time [22]. Dephasing is the effect where the phase relation between $|0\rangle$ and $|1\rangle$ in a superpostition state degenerates over time [22].

Chapter 3

Methods

The execution of the algorithm was performed in two stages. The algorithm was first executed on the QISKit local simulator. The simulator was set up to have a coupling map equal to that of the ibmqx5 backend. The simulated results indicated that the algorithm had been correctly implemented and the execution process was repeated using the ibmqx5 backend. In both stages 16 runs were performed, one run per element. For each run the code was edited to include the oracle corresponding to the sought element. Each run consisted of executing the implementation 8192 times.

3.1 Gate construction

Some of the gates required for implementing Grover's algorithm are not available through the QISKit environment and must be constructed. The implementations of these gates are shown below.

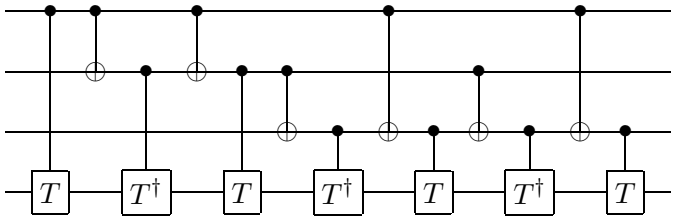
Controlled T-gate

A **T**-gate describes a rotation of $\frac{\pi}{4}$ around the Z-axis, its conjugate transpose $\mathbf{T}^\dagger$ describes a rotation of $-\frac{\pi}{4}$ around the same axis. A general controlled **U1**-gate is available, so the sought gate and its conjugate transpose can be created as described below.

$$\mathbf{cT} = \mathbf{cU1}(\frac{\pi}{4}) \qquad\qquad \mathbf{cT}^\dagger = \mathbf{cU1}(\tfrac{-\pi}{4})$$

Triple controlled Pauli Z-gate

A triple controlled **Z**-gate is needed to implement Grover's algorithm for a search space of 4 qubits. According to Mc Gettrick and Murphy [23] an arbitrary triple controlled **U**-gate can be created from a series of cNOT and controlled **V**-gates, where $\mathbf{U} = \mathbf{V}^4$. Using the design proposed by Mc Gettrick and Murphy [23] and the fact that $\mathbf{Z} = \mathbf{T}^4$ a **cccZ**-gate can be created as shown below. QISKit code for the implementation can be found in Appendix A.1.



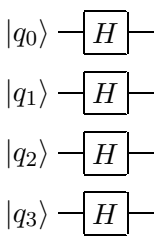
3.2 4-qubit Grover implementation

The implemented algorithm is a single iteration Grover's algorithm that does not use any ancilla bits. See Appendix A.2 for the QISKit code of the implementation and Appendix B.2 for the corresponding circuit diagram.

3.2.1 The implemented stages of the algorithm

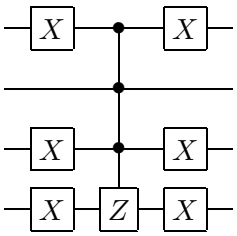
Initialization

In the first stage of the algorithm all qubits are set to be in superposition. This is done by applying the Hadamard gate to each qubit. After this operation the amplitude of each state is $\frac{1}{\sqrt{16}} = \frac{1}{4} = 0.25$.



Oracle

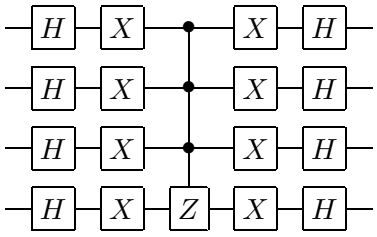
The oracle function performs a phase flip on the marked state. The phase flip inverts the amplitude α_{0010} of the state, making it $-\frac{1}{4} = -0.25$. Below the oracle for the state $|0010\rangle$ is shown. The corresponding QISKit code can be found in Appendix A.2, and a list of all oracles in Appendix B.1.



Amplification

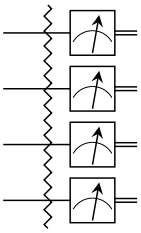
The amplification stage performs an inversion about the average of the amplitudes. It can be implemented as **HRH**, where **H** is the Hadamard transform and **R** a phase shift transform [1]. Due to the qubits being initialized to the $|0\rangle$ state **X**-gates are included to produce the correct behaviour.

The average amplitude is $\alpha_{avg} = \frac{15*0.25+(-0.25)}{16} = 0.21875$. α_{0010} differs from the average with $\delta_{0010} = \alpha_{avg} - (-0.25) = 0.46875$. Inversion around the average amplitude gives $\alpha_{0010} = \alpha_{avg} + \delta_{0010} = 0.6875$. Using the same method the amplitudes of the other states is found to be 0.1875.



Measurement

Finally the qubits are measured. A barrier, denoted by a zigzag line, is included to prevent backend optimization that could possibly reorder the gates [24].



Chapter 4

Results

In this chapter the results from the simulation runs and the executions on the ibmqx5 backend will be presented and compared. The sought state will be referred to as the *marked* state and the output value as the *measured* state.

4.1 Execution results

The average amount of times the marked state is measured on the simulator is 3882. The probability of finding the marked state $|m\rangle$ on the simulator is $\frac{3882}{8192} \approx 47.39\%$.

The average amount of times the marked state is measured on the ibmqx5 is $\frac{8675}{16} \approx 542.18$. The probability of finding the marked state $|m\rangle$ on the ibmqx5 is $\frac{8675}{16*8192} \approx 6.62\%$.

Figure 4.1 and 4.2 show data from simulation runs and executions on the ibmqx5 respectively. Each row in the heat map represents a simulation run or ibmqx5 execution. The x-axis shows what state was marked by the oracle, the y-axis shows what state was actually measured.

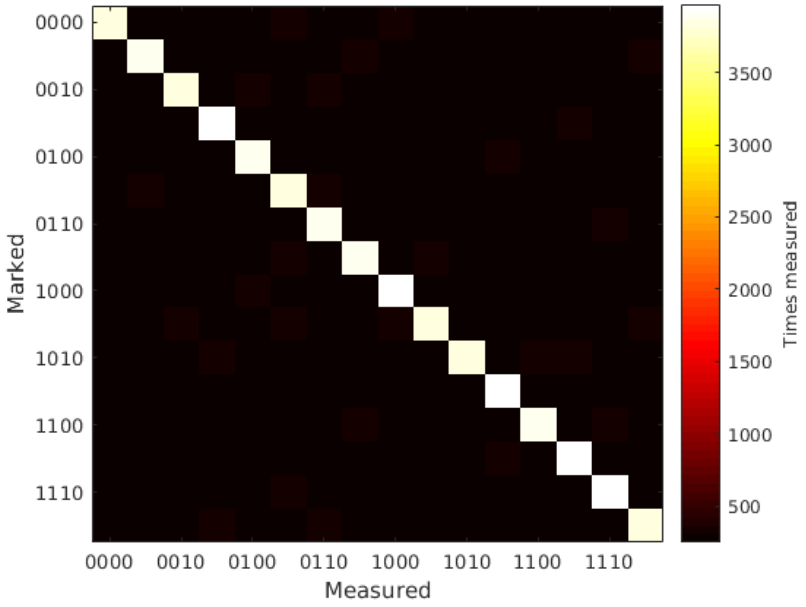


Figure 4.1: Simulator results

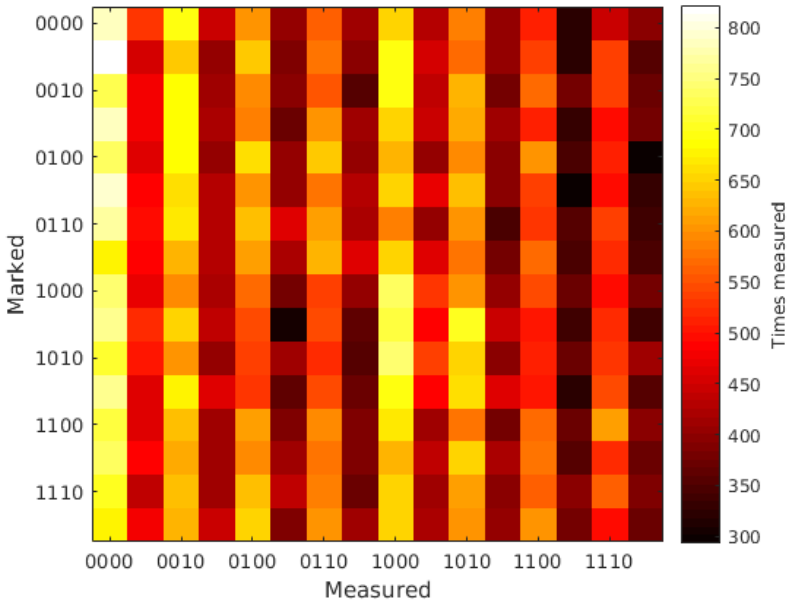


Figure 4.2: ibmqx5 results

4.2 Difference of execution results and average execution results

Figure 4.3, 4.4 and 4.5 depict the difference between the measured data and the average amount of actual measurements for each state. Each element in the heat map represents the difference in amount of measurements. In figure 4.5 the color scale is adjusted to emphasize high occurrences.

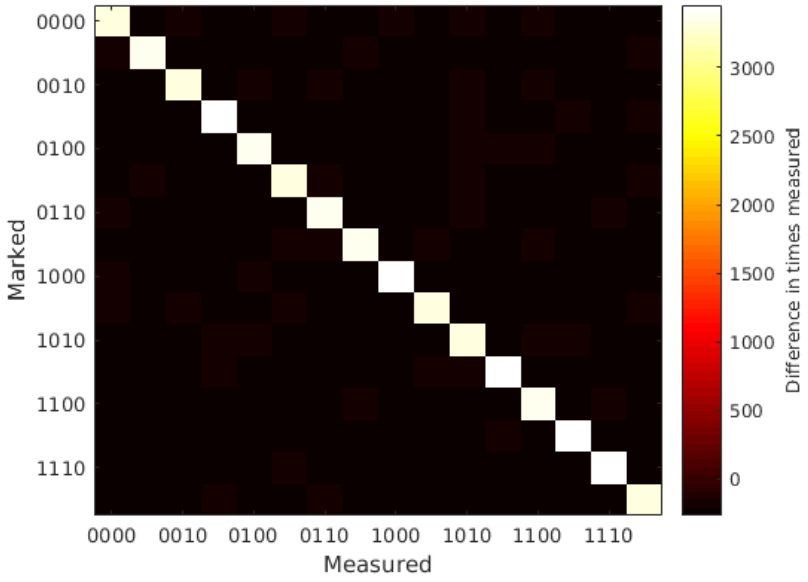


Figure 4.3: Difference of simulation results and average simulation results

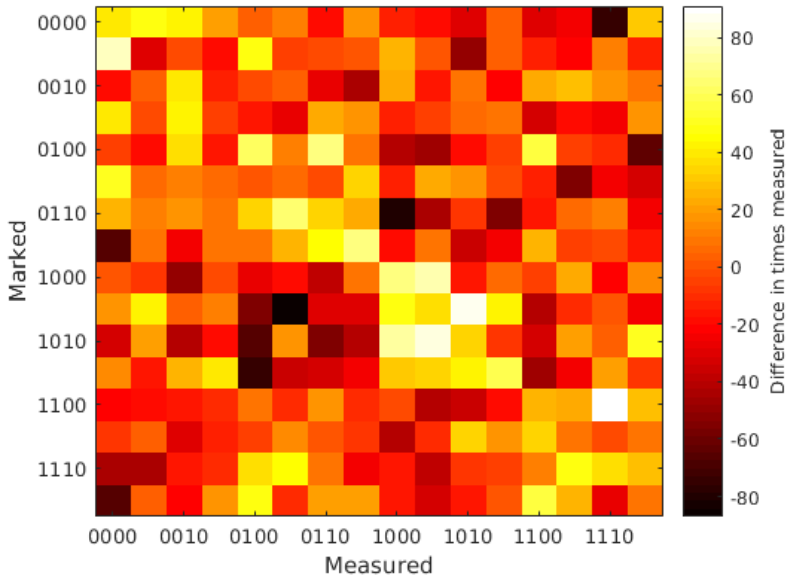


Figure 4.4: Difference of ibmqx5 results and average ibmqx5 results

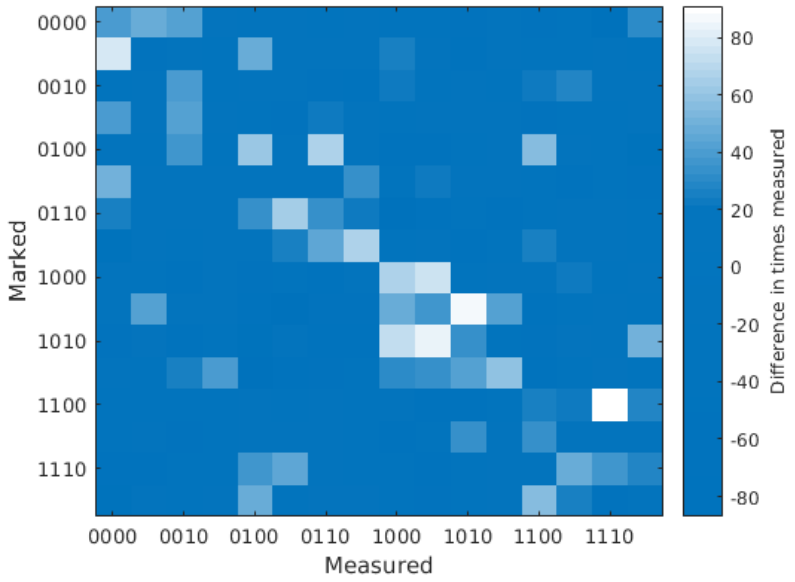


Figure 4.5: Difference of ibmqx5 results and average ibmqx5 results, emphasized

4.3 Difference of execution results and theoretically optimal results

Figure 4.6 and 4.7 depict the difference between the measured data and the statistically optimal amount of measurements for each state based on which state was marked. Each element in the heat map represents the difference between the actual measurements and the optimal amount of measurements for that state.

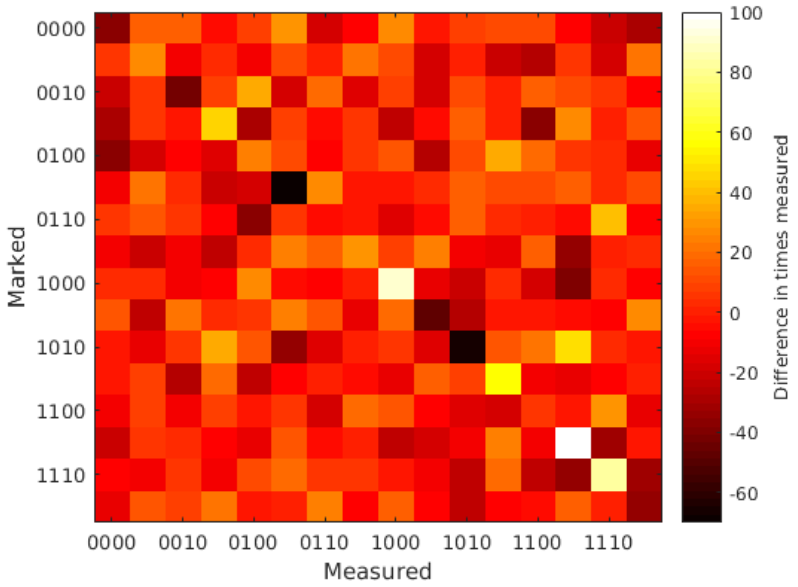


Figure 4.6: Difference of simulation results and theoretically optimal results

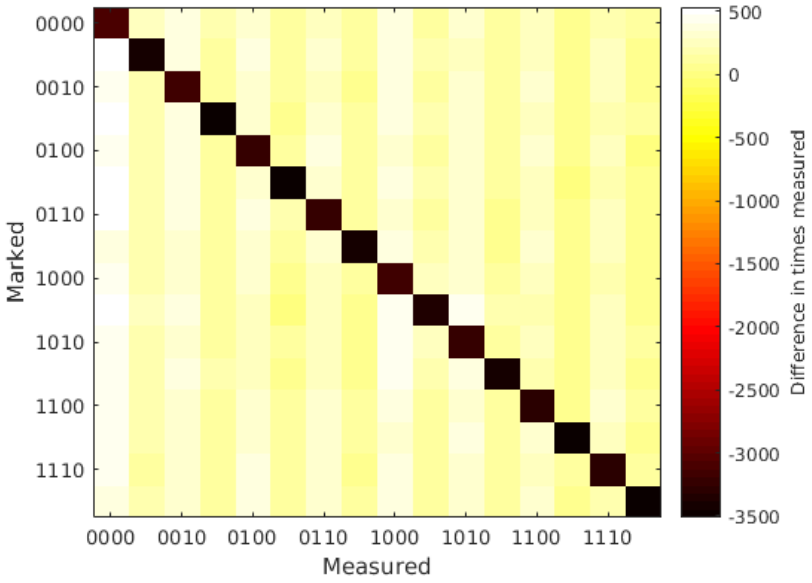


Figure 4.7: Difference of ibmqx5 results and theoretically optimal results

4.4 Comparison of measured marked gates

Figure 4.8 depicts a comparison of how many times the marked state was measured between the simulator and ibmqx5.

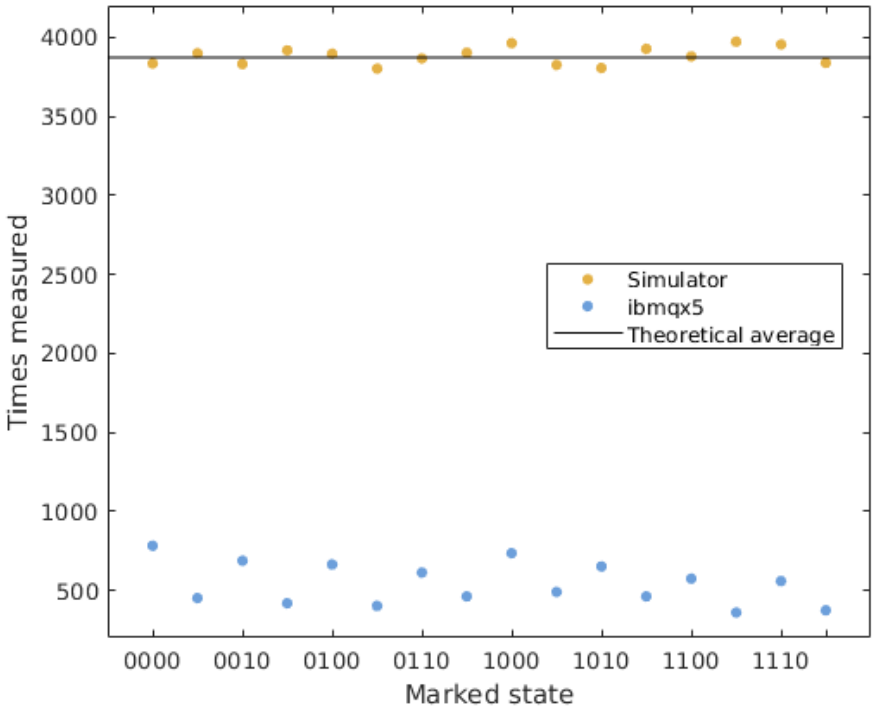


Figure 4.8: Marked states from the simulator and ibmqx5

Figure 4.9 depicts the difference between the amount of times the marked state was measured and the average amount of measurements for that state regardless of which state was marked. The data indicates that the correct state is measured marginally more often than other states.

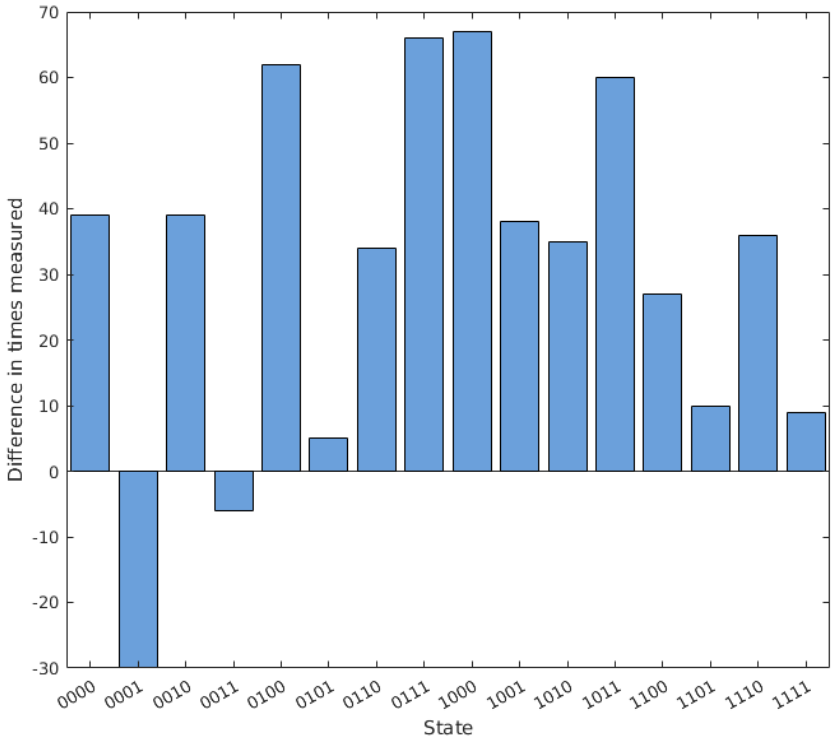


Figure 4.9: Difference in correct state measured to average state measured

4.5 Simulation

The simulator measured the marked state on average in $\frac{3882}{8192} \approx 47.39\%$ of the cases, a difference to the theoretical percentage with

$$\frac{3882}{8192} - \frac{121}{256} = \frac{5}{4096} \approx 0.12\%$$

This means that the simulation gives the correct measure 0.12% more of the time than what is expected.

The heat maps showing simulator data all look similar. Figure 4.1 and 4.3 are completely black with white elements on the diagonal, indicating that the sought elements were correctly measured. In figure 4.6 most of the heat map is red, indicating that the simulator results rarely deviates from the optimal results. Most of the extreme black and white elements are from elements that are the ones being marked, naturally having a higher value and therefore deviation. No deviation is abnormally large, as can be seen in figure 4.8.

4.6 ibmqx5

In the runs performed on the ibmqx5 backend the correct state was measured on average in $\frac{\frac{8675}{16}}{8192} \approx 6.62\%$ of the times. This differs from the theoretical amount with

$$\frac{\frac{8675}{16}}{8192} - \frac{121}{256} \approx -40.65\%.$$

This means that the ibmqx5 executions gives the correct measure 40.65% less of the time than what is expected.

Comparing the figures 4.2 and 4.1 it is obvious that the results from the ibmqx5 backend differs greatly from the simulated results. Instead of a bright diagonal the heat map has a striped appearance. The alternating bright stripes indicates that even states have a higher probability of being measured than uneven states.

In figure 4.4 a faint pattern can be seen, further enhanced in figure 4.5. A pale ribbon can be seen along the diagonal, similar to that of figure 4.3. This shows that the correct values are measured slightly more often than the incorrect values. Figure 4.9 further emphasizes this.

Chapter 5

Discussion

In this chapter the limitations that had an effect on algorithm design are discussed. The results and possible reasons for the vast discrepancy in expected and actual measurements are examined, and the chapter is concluded with suggestions for improvements.

Comparing the results with results from similar projects is difficult, as the hardware and number of circuits for each implementation differ. In the case of a 4-qubit Grover implementation on the ibmqx5 architecture, no previous papers have been published, limiting the amount of suitable comparisons.

5.1 Limitations

5.1.1 QISKit limitations

The QISKit interface does not allow programs to consist of more than 75 circuits. This makes having multiple iterations of the algorithm impossible, as it would require too many gates in its current implementation. It is arguable that having multiple iterations would not increase the occurrences of measuring marked states. Rather, having multiple iterations could decrease the occurrences, as the system would increase in size, increasing decoherence and gate errors.

Double solution oracles have a higher probability of measuring the marked state, from equation 2.1 we can calculate that it is about 78.12%. Making all possible double solutions is, however, both theoretically complicated and potentially impossible with the number of circuits available.

5.1.2 Hardware limitations

Creating a larger than 4-qubit Grover with this implementation is impossible on the ibmqx5, as its coupling map does not allow it. As the amplification step has to include a **cccZ**-gate, the implementation requires that the target bit be controlled by three other bits. The implementation also requires that at least one of the controlling bits can control two of the other bits and that another one can be controlled by two others. While the coupling map for ibmqx5 can handle these requirements, it is impossible to increase the number of qubits in the algorithm. For larger search spaces, more control bits need to be added to the **cccZ**-gate, requiring a coupling map with a higher degree of connectivity.

5.2 Sources of error

The implementation is relatively large and almost hits the circuit capacity of QISKit. As the number of circuits increases so does both execution time and gate errors. Both of these variables affect the accuracy of the algorithm and are likely the reason why patterns are not easily spotted from the data. As the simulator does not suffer from these complications and the results from it are in line with mathematical theory, it further strengthens the idea that decoherence and gate errors are what causes the ibmqx5 results to be as erratic as they are.

The striped appearance visible in figure 4.2 indicates that the likelihood of measuring a state is related to the state of the first qubit, the $|0\rangle$ state increasing the likelihood. A potential explanation to this could be that the first qubit has a lower energy relaxation time than the other qubits, making it decay to the $|0\rangle$ state faster.

5.3 Suggestions for improvements

When the circuit limit is increased, future studies could investigate the possibility of implementing additional iterations of Grover's algorithm. As the amplitude of the desired state is increased with each iteration, multiple iterations should improve the results. However, as discussed in section 5.1.1, this effect is arguable.

The 20-qubit IBM quantum computer QS1_1 [15] offers more qubits and a coupling map that would allow an implementation of Grover's algorithm with 5 or more qubits. However, at the time of writing it is not available to the public.

5.4 Conclusions

In this paper an implementation of a 4-qubit Grover's algorithm for the ibmqx5 architecture is presented. It is the largest published implementation of Grover's algorithm, as well as the largest currently possible implementation, on the ibmqx5 architecture. Executing the implementation on an ibmqx5 simulator yield results in line with the theoretically optimal results. The accuracy of the ibmqx5 simulation results compared to the ibmqx5 execution results suggests that current hardware is not yet suitable for circuits of the complexity required for a 4-qubit Grover implementation.

Bibliography

[1] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM. 1996, pp. 212–219.

[2] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[3] Kamal Gurnani, Bikash K Behera, and Prasanta K Panigrahi. "Demonstration of Optimal Fixed-Point Quantum Search Algorithm in IBM Quantum Computer". In: *arXiv preprint arXiv:1712.10231* (2017).

[4] C Figgatt et al. "Complete 3-qubit Grover search on a programmable quantum computer". In: *Nature communications* 8.1 (2017), p. 1918.

[5] Ingemar Bengtsson and Karol Życzkowski. *Geometry of quantum states: an introduction to quantum entanglement*. Cambridge University Press, 2017.

[6] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. 2000.

[7] Smite-Meister. *Bloch sphere, a geometrical representation of a two-level quantum system.* Online; accessed 23 Apr 2018. 2009.

[8] Noson S Yanofsky. "An introduction to quantum computing". In: *arXiv preprint arXiv:0708.0261* (2007).

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

[10] IBM. *Quantum computer composer*. Accessed 20 April 2018. `https://quantumexperience.ng.bluemix.net/qx/editor`. 2018.

[11] *QISKit - Open Source Quantum Information Software Kit*. Accessed 29 April 2018. `https://qiskit.org`. 2018.

[12] IBM Research and the IBM QX team. *Frequently Asked Questions*. Accessed 5 May 2018. https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=full-user-guide&page=000-FAQ~2F000-Frequently_Asked_Questions. 2018.

[13] Jens Koch et al. "Charge-insensitive qubit design derived from the Cooper pair box". In: *Physical Review A* 76.4 (2007), p. 042319.

[14] Baleegh Adbo et al. *IBM QX5: Albatross*. Accessed 4 May 2018. https://github.com/QISKit/ibmqx-backend-information/tree/master/backends/ibmqx5. 2017.

[15] IBM. *Quantum computer composer*. Accessed 20 April 2018. https://quantumexperience.ng.bluemix.net/qx/devices. 2018.

[16] IBM Research. *QISKit SDK 0.5.3 documentation*. Online; accessed 2 May 2018. https://qiskit.org/documentation/_autodoc/qiskit.html. 2018.

[17] Gennaro Auletta, Mauro Fortunato, and Giorgio Parisi. *Quantum Mechanics*. Cambridge University Press, 2009.

[18] Gilles Brassard, Peter Hoyer, and Alain Tapp. "Quantum algorithm for the collision problem". In: *arXiv preprint quant-ph/9705002* (1997).

[19] Michel Boyer et al. "Tight bounds on quantum searching". In: *Fortschritte der Physik: Progress of Physics* 46.4-5 (1998), pp. 493–505.

[20] Wojciech Hubert Zurek. "Decoherence and the transition from quantum to classical—revisited". In: *Quantum Decoherence*. Springer, 2006, pp. 1–31.

[21] Maximilian Schlosshauer. "Decoherence, the measurement problem, and interpretations of quantum mechanics". In: *Rev. Mod. Phys.* 76 (4 Feb. 2005), pp. 1267–1305. DOI: 10.1103/RevModPhys.76.1267. URL: https://link.aps.org/doi/10.1103/RevModPhys.76.1267.

[22] Jay M Gambetta, Jerry M Chow, and Matthias Steffen. "Building logical qubits in a superconducting quantum computing system". In: *npj Quantum Information* 3.1 (2017), p. 2.

[23] Michael Mc Gettrick and Billy Murphy. *Simulation of the CCC-Not Quantum Gate.* Technical Report NUIG-IT-061002, Department of Information Technology, NUI, Galway.

[24] Andrew W Cross et al. “Open quantum assembly language”. In: *arXiv preprint arXiv:1707.03429* (2017).

Appendix A

QISKit code

A.1 cccZ-gate implementation

```
from qiskit import QuantumProgram
import math
qp = QuantumProgram()

pi = math.pi

qr = qp.create_quantum_register('qr', 4)
cr = qp.create_classical_register('cr', 4)
qc = qp.create_circuit('cccZ', [qr], [cr])

qc.cu1(pi/4, qr[0], qr[3])
qc.cx(qr[0], qr[1])
qc.cu1(-pi/4, qr[1], qr[3])
qc.cx(qr[0], qr[1])
qc.cu1(pi/4, qr[1], qr[3])
qc.cx(qr[1], qr[2])
qc.cu1(-pi/4, qr[2], qr[3])
qc.cx(qr[0], qr[2])
qc.cu1(pi/4, qr[2], qr[3])
qc.cx(qr[1], qr[2])
qc.cu1(-pi/4, qr[2], qr[3])
qc.cx(qr[0], qr[2])
qc.cu1(pi/4, qr[2], qr[3])

qc.measure(qr[0], cr[0])
qc.measure(qr[1], cr[1])
qc.measure(qr[2], cr[2])
qc.measure(qr[3], cr[3])
```

A.2 4-qubit Grover's Algorithm

```
from qiskit import QuantumProgram
import math

import Qconfig
from IBMQuantumExperience import IBMQuantumExperience
api = IBMQuantumExperience(Qconfig.APItoken,
    {'url':Qconfig.config["url"]})
from qiskit.backends import discover_remote_backends
remote_backends = discover_remote_backends(api)

qp = QuantumProgram()

pi = math.pi

qr = qp.create_quantum_register('qr', 4)
cr = qp.create_classical_register('cr', 4)
qc = qp.create_circuit('Grover', [qr], [cr])
shots = 8192

########################
######## init #########
########################
qc.h(qr[0])
qc.h(qr[1])
qc.h(qr[2])
qc.h(qr[3])

########################
### Oracle for 0010 ###
########################
qc.x(qr[0])
qc.x(qr[2])
qc.x(qr[3])

qc.cu1(pi/4, qr[0], qr[3])
qc.cx(qr[0], qr[1])
qc.cu1(-pi/4, qr[1], qr[3])
```

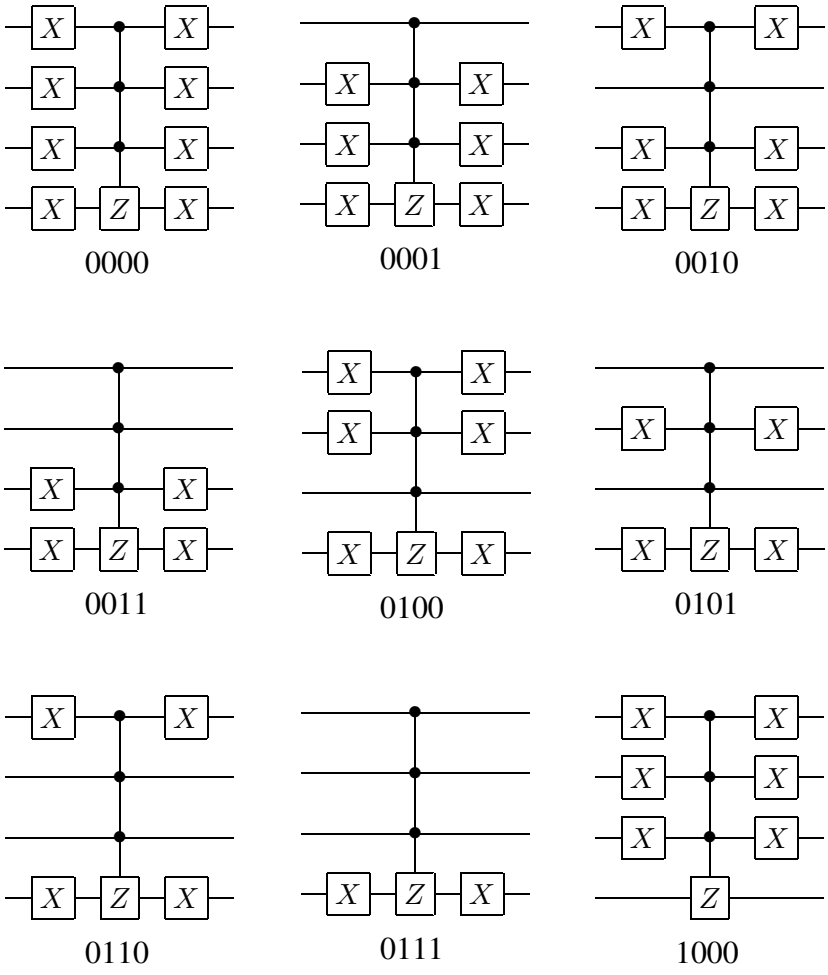
```
qc.cx(qr[0], qr[1])
qc.cu1(pi/4, qr[1], qr[3])
qc.cx(qr[1], qr[2])
qc.cu1(-pi/4, qr[2], qr[3])
qc.cx(qr[0], qr[2])
qc.cu1(pi/4, qr[2], qr[3])
qc.cx(qr[1], qr[2])
qc.cu1(-pi/4, qr[2], qr[3])
qc.cx(qr[0], qr[2])
qc.cu1(pi/4, qr[2], qr[3])

qc.x(qr[0])
qc.x(qr[2])
qc.x(qr[3])

########################
#### Amplification ####
########################
qc.h(qr[0])
qc.h(qr[1])
qc.h(qr[2])
qc.h(qr[3])

qc.x(qr[0])
qc.x(qr[1])
qc.x(qr[2])
qc.x(qr[3])

######## cccZ #########
qc.cu1(pi/4, qr[0], qr[3])
qc.cx(qr[0], qr[1])
qc.cu1(-pi/4, qr[1], qr[3])
qc.cx(qr[0], qr[1])
qc.cu1(pi/4, qr[1], qr[3])
qc.cx(qr[1], qr[2])
qc.cu1(-pi/4, qr[2], qr[3])
qc.cx(qr[0], qr[2])
qc.cu1(pi/4, qr[2], qr[3])
qc.cx(qr[1], qr[2])
```

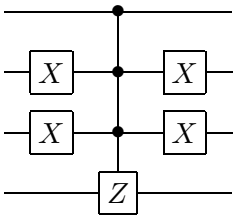
```
qc.cu1(-pi/4, qr[2], qr[3])
qc.cx(qr[0], qr[2])
qc.cu1(pi/4, qr[2], qr[3])
####### end cccZ #######

qc.x(qr[0])
qc.x(qr[1])
qc.x(qr[2])
qc.x(qr[3])

qc.h(qr[0])
qc.h(qr[1])
qc.h(qr[2])
qc.h(qr[3])

########################
####### Measure ########
########################
qc.barrier(qr)
qc.measure(qr[0], cr[0])
qc.measure(qr[1], cr[1])
qc.measure(qr[2], cr[2])
qc.measure(qr[3], cr[3])

# submit job #
qp.execute(['Grover'], backend='ibmqx5', shots = shots,
    max_credits = 5, timeout=1)

# submit job to ibmqx5 simulator #
#backend='local_qasm_simulator'
#coupling_map = {1:[0,2], 2:[3], 3:[4, 14], 5:[4],
#    6:[5,7,11], 7:[10], 8:[7],9:[8, 10], 11:[10],
#    12:[5, 11, 13], 13:[4, 14], 15:[0, 2, 14]}
#qp.execute(['Grover'], backend=backend, shots = shots,
#    coupling_map=coupling_map, max_credits = 5, timeout=1)
```

Appendix B

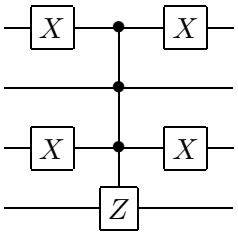
Quantum circuits

B.1 Oracles

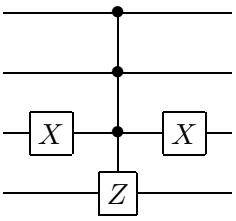




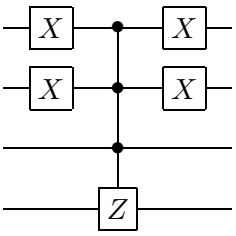
1001



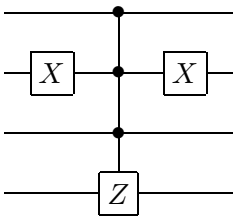
1010



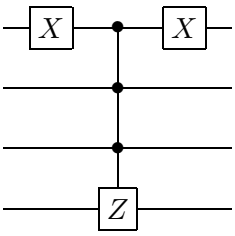
1011



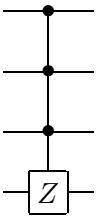
1100



1101

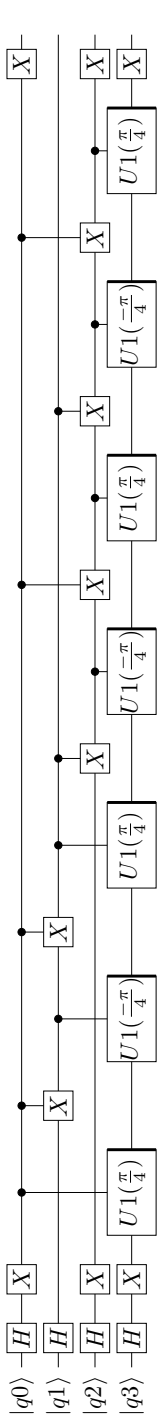


1110

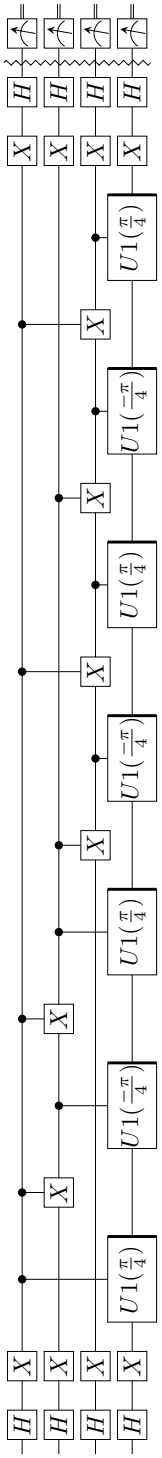


1111

B.2 Grover's algorithm



cont.



Appendix C

Acknowledgements

We acknowledge use of the IBM Q experience for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q experience team.