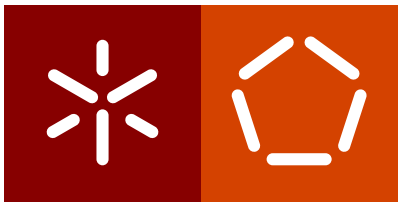


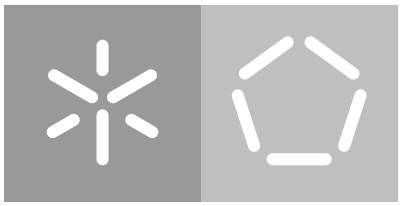
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Afonso Rodrigues

Validation of quantum simulations

Assessing efficiency and reliability in experimental implementations

November 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Afonso Rodrigues

Validation of quantum simulations

Assessing efficiency and reliability in experimental implementations

Master dissertation
Master Degree in Engineering Physics

Dissertation supervised by
Luís Barbosa
Carlos Tavares

November 2018

ACKNOWLEDGEMENTS

This work was only possible thanks to the contribution of several people to which I want to express my gratitude.

To my supervisor prof. Luís Barbosa, for giving me the opportunity to undertake this project, and for his constant support, availability and patience; also to my co-supervisor Carlos Tavares, for his early insight.

In no particular order of importance, to Miguel Nogueira, Ricardo Alves, Rui Costa, João Fonseca, Miguel Sanches, Joaquim Santos, and José Bruno, my friends and colleagues who accompanied me in this course and provided invaluable help.

To my parents and sister, and Maria Coelho, André Souto and Gil Pimentel for their unconditional encouragement on the difficult days.

Finally, I wish to thank University of Minho for these five years of learning, INESC TEC and specifically the HASLab team for supporting me in my first steps as a researcher.

I acknowledge use of the IBM Q for this work.

ABSTRACT

Quantum simulation is one of the most relevant applications of quantum computation for the near future, due to its scientific impact and also because quantum simulation algorithms are typically less demanding than generalized quantum computations. Ultimately, the success of a quantum simulation depends on the amount and reliability of information one is able to extract from the results. In such a context, this work reviews the theory behind quantum simulation, with a focus on digital quantum simulation. The concepts of efficiency and reliability in quantum simulations are discussed, particularly for implementations of digital simulation algorithms in state-of-the-art quantum computers. A review of approaches for quantum characterization, verification and validation techniques (QCVV) is also presented. A digital quantum simulation of the Schrödinger equation for a single particle in 1 spatial dimension was experimentally implemented and analyzed, along with a quantum state tomography procedure for characterization of the final quantum state and evaluation of simulation reliability.

From the literature, it is shown that digital quantum simulation is theoretically sound and experimentally feasible, with several applications in a wide range of physics-related fields. Nonetheless, a number of conditions arise that must be observed for a truly efficient implementation of a digital quantum simulation, from theoretical conception to experimental circuit design. The review of QCVV techniques highlights the need for characterization and validation techniques that could be efficiently implemented for current models of quantum computation, particularly in instances where classical verification is not tractable. However, there are proposals for efficient verification procedures when a set of parameters defining the final result of the simulation is known.

The experimental simulation demonstrated partial success in comparison with an ideal quantum simulation. From the results it is apparent that better coherence times, better reliability and finer control are as decisive for the advancement of quantum computing power as the more-publicized number of qubits of a given device.

RESUMO

A simulação quântica é uma das aplicações mais relevantes da computação quântica num futuro próximo, não só devido ao seu impacto científico como também porque os algoritmos de simulação quântica são tipicamente menos exigentes do que algoritmos quânticos numéricos. Em última análise, o sucesso de uma simulação quântica depende da quantidade e fiabilidade das informações que é possível extrair dos resultados. Neste contexto, este trabalho apresenta uma revisão da teoria da simulação quântica, com ênfase na simulação quântica digital. Os conceitos de eficiência e fiabilidade em simulações quânticas são discutidos, particularmente para implementações de algoritmos de simulação digital. Uma revisão de técnicas de caracterização, verificação e validação de sistemas quânticos (QCVV) é também apresentada. Uma simulação quântica digital da equação de Schrödinger para uma única partícula a uma dimensão espacial foi implementada experimentalmente e analisada, juntamente com um método de tomografia de estado quântico para a caracterização do estado quântico final e avaliação da fiabilidade da simulação.

A partir da literatura, é demonstrado que a simulação quântica digital é teoricamente sólida e experimentalmente viável, com várias aplicações em diversas áreas da física. No entanto, existem várias condições a ter em conta para uma implementação verdadeiramente eficiente de uma simulação quântica digital, da sua concepção teórica até à implementação experimental de circuitos. A revisão de técnicas QCVV destaca a necessidade de técnicas de caracterização e validação que possam ser eficientemente implementadas para modelos atuais de computação quântica, particularmente em instâncias em que a verificação clássica não é possível ou desejável. No entanto, existem propostas para técnicas de verificação que são eficientes quando se conhece, a priori, um conjunto de parâmetros característicos do resultado final da simulação.

A simulação experimental demonstrou sucesso parcial relativamente a uma simulação quântica ideal. A partir dos resultados, evidencia-se que melhores tempos de coerência, maior fiabilidade e controlo mais refinado são tão decisivos para o avanço da computação quântica quanto o número de qubits de um dispositivo.

CONTENTS

1

INTRODUCTION

1.1 THE CONTEXT: QUANTUM SIMULATION

The possibility of performing computational tasks deemed inefficient, or even impossible, on available classical computing power has increased the momentum on quantum computation and simulation research over the past decade. However, there are still major milestones to be reached before the first fault-tolerant, universal quantum computer is built. As of 2018, available quantum devices work by approximating quantum computations on physical qubits; quantum information and computation research is entering the *NISQ* (Noisy intermediate-scale quantum) era. Before a noise-resilient *logical* qubit - one that performs as theoretically predicted, holding its state in arbitrarily long quantum algorithms - is reached, error rates and coherence times need to be further improved, and error correcting codes allowing for implementation of a universal set of gates while keeping low overhead, need to be devised (Campbell et al., 2017).

Quantum simulation is currently one of the most relevant applications of quantum computation. This is true not only due to its scientific and industrial impact, but also because quantum simulation algorithms are typically less demanding than general quantum computations. For example, a quantum simulator with tens of qubits could already perform useful simulations under current technology, whereas thousands of qubits would be needed to factorize modest numbers using Shor's algorithm (Buluta and Nori, 2009). In fact, quantum simulators could even explore the presence of environmental errors and decoherence to simulate the presence of same phenomena on the simulated system (Lloyd, 1996). It is also believed that no known classical algorithm can, without compromises, efficiently simulate the dynamics of a quantum system (Preskill, 2018).

The biggest demonstrated classical numerical simulation of a quantum system was performed by a team of researchers from IBM on a conventional supercomputer, in October 2017. The team managed to effectively simulate a 56-qubit quantum system, which implicates that a scenario of quantum supremacy Boixo et al. (2018) would be achieved on a quantum computer with a greater amount of qubits and reasonable fidelity. Of course, quantum supremacy is dependent on many factors other than qubit number (e.g. universality, fidelity, entanglement

capabilities, decoherence), and, as classical computational power grows, so will this threshold, so the timeline for reaching quantum supremacy remains uncertain. Fortunately, quantum simulation is one of the most promising applications due to its relatively low computational requirements, which means that one may see a truly useful quantum simulator even before a universal quantum computer appears.

A significant reduction of noise and decoherence effects, together with suitable error-correcting algorithms, allow for the possibility of fault-tolerant quantum computing (Aharonov and Ben-Or, 1997), yet, the required overhead for error correction is still too demanding for current and near-future quantum systems. Moreover, the nature of quantum mechanics itself suggests that these effects can never be completely eliminated. Therefore, the necessity of validation protocols for quantum computation and quantum simulations becomes apparent.

In Cirac and Zoller (2012), the authors, inspired by the criteria devised a decade earlier by DiVincenzo (2000) for the physical implementation of a functional quantum computer, define a list of conditions which a quantum simulator must fulfill to demonstrate a classically intractable simulation of a many-body quantum system involving large-scale entanglement:

1. *Quantum system*: the simulator should possess a system of bosons or fermions, which can be stored in a lattice or confined in a limited space, and have a large number of degrees of freedom;

2. *Initialization*: the simulator should be able to prepare, within some bounded error, a known quantum state;

3. *Hamiltonian engineering*: the simulator should be able to devise an adjustable set of interactions with external fields or between particles, which can be local or have a longer range. Among the accessible Hamiltonians, there should be some that cannot be efficiently simulated with classical techniques;

4. *Detection*: the simulator should have the ability to perform measurements on the system, either on individual qubits, or collectively (without the need of addressing any individual site);

5. *Verification*: there should be a way of checking or increasing confidence in the results.

If the simulator is dealing with a model that cannot be classically simulated, by definition, there should be no way of verifying the result of the simulation using classical resources. The authors suggest alternatives such as benchmarking the simulator for problems with known solutions, comparing the results of the simulation through different methods or physical implementations, or even running the evolution backwards in time to check that it ends up in the initial state.

While proposed implementations, initialization, Hamiltonian evolution, and measurement of controllable quantum systems or quantum computers have been extensively studied, particularly over the last twenty years, only more recently, as the plausibility of a quantum supremacy scenario materializes, have verification and validation techniques been properly discussed by the scientific community. In an article with the remarkable title "Can one trust quantum simulators?" (Hauke et al., 2012), the authors argue that, to be truly useful, a quantum simulator must satisfy four conditions: *relevance*, for applications and understanding of the fields of interest; *controllability* of the parameters of the simulated model and state preparation, manipulation, evolution and detection of the relevant physical properties of the system; *reliability* of the observed physics of the quantum simulator in relation to an ideal model whose properties are being simulated; *efficiency*, more specifically in comparison with what is practically possible on a classical computer.

From this set of conditions for a quantum simulator it arises that the true advantage of quantum simulators would be shown for models that are computationally hard for classical computers - even though it may be desirable to set the parameters in a regime where the model is tractable by classical simulations, since this provides an elementary instance of validating the quantum simulation. This means there is a need for more sophisticated techniques of validation, in particular for systems that are inefficient to simulate classically. A proposed technique is the checking of the sensitivity of the quantum simulation in respect to the addition of noise and disorder, which is possible only with sufficient control over the simulation. The need for a careful analysis of reliability and efficiency in the presence of imperfections is emphasized.

1.2 SCHRÖDINGER EQUATION

Simulation of quantum systems builds on a basic mathematical tool: the Schrödinger equation which describes its evolution. Its relevance for the purpose of the current dissertation justifies the following brief introduction.

At the beginning of the twentieth century, experimental evidence suggested that atomic particles also exhibit a wave-like behaviour. Thus, it became reasonable to assume that a wave equation could explain the behaviour of atomic particles; E. Schrödinger Schrödinger (1926) was the first to publish such a wave equation, forming the basis for his work that resulted in him being awarded the Nobel Prize in Physics in 1933.

The Schrödinger equation is a partial differential equation which provides a mathematical model that allows for the determination of the wave function of a system, and describes its behavior over time. It is the quantum analogue to Newton's laws and the conservation of energy in classical mechanics.

The most general form of the equation is the time-dependent Schrödinger equation, which describes the wave function $\psi(\mathbf{r}, t)$ of a quantum system, at time t and position $\mathbf{r}$, for a given Hamiltonian $\hat{H}$:

$$i\hbar\frac{\partial}{\partial t}\psi(\mathbf{r}, t) = \hat{H}\psi(\mathbf{r}, t) \tag{1}$$

The Hamiltonian can be interpreted as describing the total energy of the system. It contains a set of operations concerning all the interactions affecting the state of the system, and as a physical observable, it is self-adjoint. The time evolution is defined by the exponential of the Hamiltonian, which makes it a unitary operator, as per Stone's theorem on one-parameter unitary groups. For a known wave function $\psi(\mathbf{r}, 0)$, the Schrödinger equation can provide knowledge about the wave function at an arbitrary time t_f, and allows for determination of outcome probability; $|\psi(\mathbf{r}, t_f)|^2$ is the probability of finding a quantum particle at a position $\mathbf{r}$ and time t_f.

Considering a one dimensional potential $V(x)$, a single particle of mass m is governed by the Hamiltonian:

$$\hat{H} = \frac{\hbar^2\hat{k}^2}{2m} + V(x) \tag{2}$$

where $\hbar$ is the reduced Planck constant, and $\hat{k}$ is the wave number of the particle. These quantities are related to the momentum p of the particle through the de Broglie equation for matter waves: $p^2 = \hbar^2\hat{k}^2 = \frac{-\hbar^2}{2m}\frac{d^2}{dx^2}$.

The general procedure for quantum simulation involves preparing an initial state $|\phi(0)\rangle$, finding the state $|\phi(t)\rangle$ of the quantum system at some time t and computing the value of some physical quantity of interest. For a time independent Hamiltonian, H, the solution to the Schrödinger equation:

$$i\hbar\frac{d}{dt}|\phi\rangle = H|\phi\rangle \tag{3}$$

is given by $|\phi(t)\rangle = e^{-i\hbar Ht}|\phi(0)\rangle$.

Since analytical solutions of Schrödinger's equation have only been found for a limited number of quantum systems, physicists often have to resort to numerical algorithms and what computational power is available to solve the equation for a given physical system and find its associated potential energy. Classical algorithms for quantum simulation exist, such as quantum Monte Carlo methods (Ceperley and Alder, 1986), which can provide either exact solutions to the equation, or polynomially scaling approximations, but generally not both; these methods also suffer from the "negative sign problem" (Troyer and Wiese, 2005) when applied to fermions, increasing computation time exponentially with the number of particles.

1.3 OBJECTIVES

This dissertation reviews the theory behind quantum simulation, with a focus on digital quantum simulation. The concepts of efficiency and reliability (i.e. error bounds) of a quantum simulation are also discussed, from its algorithmic formulation to the actual implementation in real-world quantum devices with real-world limitations, namely due to processor architectures and noise processes. The success of a quantum simulation ultimately depends on the ability to extract useful information from the simulator. A review of approaches for quantum characterization, verification and validation techniques (QCVV) is also presented.

A digital quantum simulation of the Schrödinger equation for a single particle in 1 spatial dimension was experimentally implemented and analyzed, along with a quantum state tomography procedure for characterization of the final quantum state and evaluation of simulation reliability.

Within the context outlined in the beginning of this introductory section, the main research questions addressed in this dissertation are:

1. What are the conditions for a truly efficient implementation of a digital quantum simulation of a physical system, given the restrictions imposed by current noisy intermediate-scale quantum (NISQ) devices?

2. What specifications should be considered for quantum characterization, verification or validation (QCVV) of quantum simulators and the results of such simulations?

While it has been proven that a universal quantum computer is, in theory, able to efficiently simulate the Hamiltonian of a physical system (Lloyd, 1996), which is limited to ℓ-local interactions, or even sparse Hamiltonians (Berry et al., 2017), i.e. with no more than a fixed number of nonzero entries in each column of its fixed representation, these works do not consider significant obstacles to implementation present in experimental settings, such as how to efficiently decompose the Hamiltonian into a sequence of implementable operations on a quantum computer (Vartiainen et al., 2004; Shende et al., 2006), how to efficiently find a mapping obeying the constraints of nearest-neighbour quantum chip architectures (Siraichi et al., 2018), or even how do these results hold in the presence of noise and decoherence (Aharonov and Ben-Or, 1997).

However, efforts to review and unify the theory behind experimental digital quantum simulation have been presented by Brown et al. (2010), and in Georgescu et al. (2014), which also reviews the concepts behind analog quantum simulation and provides an extensive review on the applications and implementations of quantum simulation.

The subject of verifying or validating a quantum computation overlaps with that of verification and validation of quantum simulations, particularly when discussing the use of quantum

computers for digital quantum simulation, which this work focuses on. As mentioned before, Hauke et al. (2012) asks "Can one trust quantum simulators?" and discusses in detail the requirements for near-future quantum simulators and emphasizing the need for a careful analysis of reliability and efficiency in the presence of imperfections.

The experimental and conceptual approaches to this problem are multidisciplinary. For example, Artiles et al. (2005) present quantum tomography in the context of statistical methods; Chuang and Nielsen (1997) originally proposed a procedure for quantum process tomography, based itself on quantum state tomography; with a focus on efficiency, da Silva et al. (2011) proposes a characterization method that matches experimental data with a subset of possible descriptions. Benchmarking techniques, such as proposed by Knill et al. (2008), allow for an estimation of the fidelity of a quantum device that is not independent from a specific quantum algorithm. From computational sciences and computational complexity theory arise different approaches based on the concept that current models of quantum computation do not generally allow the experimenter direct access to the quantum device; instead, interactions occur through classical or quantum channels. Some techniques are reviewed in Gheorghiu et al. (2017); the most prominent being quantum interactive proofing (Aharonov et al., 2017) and blind quantum computation (Fitzsimons, 2017). All of the techniques described have, arguably, potential use in the validation of quantum simulations.

The experimental part of this dissertation aims at providing a qualitative view on the degree of success to be expected from an experimental simulation, namely that of the Schrödinger equation for a single particle, on available quantum devices provided by the IBM Q initiative. It also serves to illustrate the obstacles, described above, to digital quantum simulation in quantum computers, and demonstrate the implementation and degree of success of a quantum state tomography technique proposed by Smolin et al. (2012).

The simulation algorithm itself was first outlined in Zalka (1998); Wiesner (1996). An algorithm for simulation of the Schrödinger equation in the circuit model of quantum computing is detailed and simulated (classically) in Benenti and Strini (2008). This work in particular follows the procedure proposed and experimentally demonstrated by Coles et al. (2018), while also expanding it for 3 qubits.

1.4 Outline

The dissertation is structured as follows: section 2 introduces the fundamental theoretical concepts behind quantum simulation, while distinguishing between digital quantum simulation (2.1) and analog quantum simulation (2.3). There is a focus on the efficiency and reliability of digital quantum simulation (2.2), particularly given the constraints of noisy intermediate-scale quantum computers. An overview on the physical implementations (2.4) and applications (2.5) of quantum simulations is also presented.

In section 3 a review of quantum characterization, validation and verification techniques is given, with a particular focus on quantum state tomography, quantum process tomography, and randomized benchmarking. General conditions for these procedures, as well as efficiency, are discussed.

Section 4 introduces the experimental procedure, with a description of the quantum devices (4.1) in which the experiment is realized, and a detailed description of the implementation of the simulation algorithm (4.2) and the state tomography procedure (4.3).

The results of the experimental procedure are discussed in section 5. Finally, section 6 concludes the dissertation with a number of suggestions for future work which could expand or build upon what is presented here.

2

QUANTUM SIMULATION

Numerical simulation plays an important role in science. Its use allows scientists to check, in detail, the predictions of a mathematical model of a physical system, specially when such models become too hard to solve analytically, or when details are required for specific values of parameters. However, opting for numerical simulation is only practical when its calculations can be done efficiently with available resources. Numerical simulation for mathematical models has historically been one step ahead of available computational power, which justifies the widespread demand for ever more powerful supercomputers. Many calculations require more computational power than what researchers have readily available, and this limitation is nearly ubiquitous independently of scientific field. For those working with quantum systems, however, this happens for rather small system sizes. This leaves open problems in important areas, such as quantum chemistry, high-energy physics or high temperature superconductivity, where progress is slow since for larger systems, actual models cannot be adequately tested or used for predictions.

To appreciate how quickly computational requirements grow with the size of a quantum system, one may consider a straightforward approach to storing and operating on a general quantum state $|\psi_n\rangle$ of n qubits, each one representing a two-state quantum system. The Hilbert space of this state grows exponentially with n, since it is spanned by 2^n orthogonal states $|j\rangle$, with $0 \leq j < 2^n$. Because the n qubits can have any degree of superposition between them, the expression for $|\psi_n\rangle$ becomes a sum over all these terms, each with a different coeficient c_j:

$$|\psi_n\rangle = \sum_{j=0}^{2^n-1} c_j \, |j\rangle \tag{4}$$

To store this description of the state on a classical computer, all complex coefficients $\{c_j\}$ need to be stored. Admitting each one requires two 4-byte floating point numbers, one for the real and another for the imaginary part of the number, each coefficient occupies 8 bytes of memory. Each additional qubit effectively doubles the amount of memory needed: a 28-qubit state would require around 1 gigabyte of memory, and for n=38 qubits, 1 terabyte would be necessary.

Complexity theory has, however, shown (Preskill, 1998) that bounded-error quantum polynomial time (BQP), the class of decision problems solvable by a quantum computer in polynomial time, with an error probability of at most 1/3, is contained in PSPACE, the set of all decision problems solvable by a Turing machine using a polynomial amount of space. This means that, in principle, a classical computer should need only a polynomial amount of space to store a quantum state of n qubits. In spite of that, the real difficulty and limiting factor of a classical numerical simulation of quantum systems, is the time necessary to perform any calculation over the state, which is exponential over the number of qubits constituting the system.

Although a quantum computer can, by design, efficiently store the quantum state under study, it is not a complete replacement for a classical computer. Taking into account the methods and results of the simulations, a classical computer allows access to the full quantum state, i.e. all 2^n complex numbers $\{c_j\}$ contained in equation (4). One realization and direct measurement of the system in a quantum device, by itself, could only tell whether one of the coefficients c_j is non-zero. For quantum simulation in particular, where a greater amount of information about the state is usually desired, accessing enough useful information typically requires a statistically significant number of repetitions of the simulation. In this context, classical simulations can be classified as a "strong simulations" (Nest, 2008), since they provide full information about the probability distribution, while repeated realization and measurement quantum systems, on a quantum device, only provides a sampling from the probability distribution, a "weak simulation". In this scenario, a wider class of classical algorithms exist which can efficiently perform quantum computations. Taking this differentiation into account, a quantum simulator would be particularly useful in cases when neither a strong nor weak simulation can be efficiently performed classically.

In a lecture titled *Simulating Physics with Computers* (Feynman, 1982), Richard Feynman posed the question "What kind of computer are we going to use to simulate physics?". Feynman suggested a device which does not approximate a simulation using numerical algorithms for differential equations, but exactly simulates the behaviour of physical systems. By designing a well-controlled system from the bottom up, one could create a computer whose constituent parts are governed by quantum dynamics generated by a desired Hamiltonian. Feynman's idealized machine is the most prominent inspiration for quantum computation, also proposed independently by Benioff (1980), and Deutsch (1985).

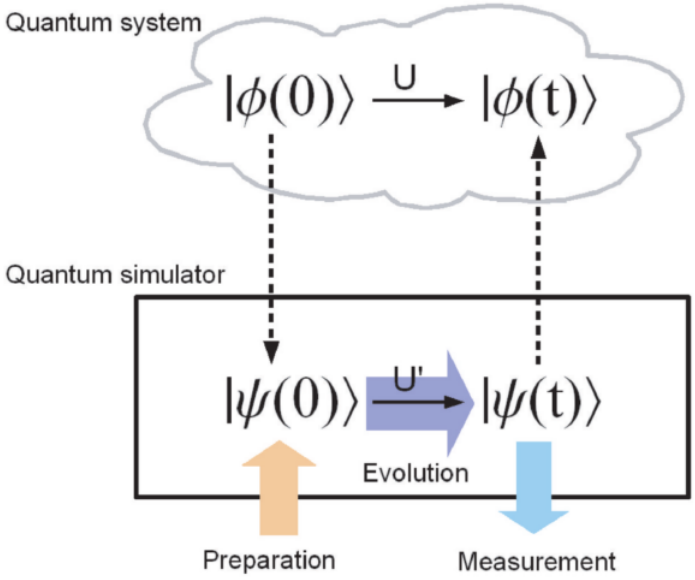


Figure 1.: Schematic representation of a quantum system and a corresponding quantum simulator (Georgescu et al., 2014).

Described in generic terms, a procedure for quantum simulation involves taking a quantum system with some degree of controllability, and:

1. Preparing an initial state (preparation);
2. Performing some kind of quantum processing (evolution);
3. Extracting information from the final state (measurement);

As is the case with general quantum algorithms, all three steps must be performed efficiently (i.e. scaling polynomially with the size of the system) to obtain a computation that is efficient overall. Step *b)* usually corresponds to the time evolution of the Hamiltonian, which is the case of the simulation implemented experimentally in this work, explained in detail in section 4.2. The problem can thus be stated mathematically by the expression:

$$|\psi(t)\rangle = e^{i\hat{H}t}\,|\psi(0)\rangle \tag{5}$$

Given an initial state $|\psi(0)\rangle$ and the Hamiltonian $\hat{H}$, which may itself be time-dependent, the simulation should lead to state $|\psi(t)\rangle$ at time t. It should be noted that quantum simulation is not restricted to recreating the temporal evolution of the simulated system. Other applications include, for example, phase estimation for computing eigenvalues of the Hamiltonian (Abrams and Lloyd, 1999; Wang et al., 2010b), computing partition functions (Lidar and Biham, 1997), or even using quantum computers to simulate classical physics more efficiently (Meyer, 2002; Yung et al., 2010).

Taking the general definition of a quantum simulation as a starting point, two approaches can be distinguished: *digital* quantum simulation, and *analog* quantum simulation. They are discussed in the next three sections.

2.1 DIGITAL QUANTUM SIMULATION

A digital quantum simulator (DQS) uses quantum bits to encode the initial state of the quantum system as a superposition of binary bit strings. Admitting the goal of the simulation is to get the simulator from state A to state B along a particular route, the implemented simulation algorithm drives the system in discrete limited steps, by turning on and off Hamiltonians from a set, each moving the system a controlled distance along a predetermined direction in the Hilbert space. This technique is comparable to a typical classical simulation, where the simulation model is mapped onto registers and standard gate operations available in a commercial computer, with the help of high-level programming languages and compilers. Some representative studies on DQS are Terhal and DiVincenzo (2002); Somma et al. (2002); Verstraete et al. (2009); a survey on the use of quantum computers for quantum simulation is presented in Brown et al. (2010).

This approach can be implemented in the circuit model of quantum computation by using compositions of quantum gates to build a desired Hamiltonian. A seminal work by Lloyd (1996) shows that any unitary operation can be written in terms of universal quantum gates; the same work also specifies the conditions necessary for the efficient simulation of quantum systems on a universal quantum computer, which are detailed below. Therefore, a universal digital quantum simulator can also be regarded as a quantum computer implementing quantum algorithms for physical modelling of a quantum system. The main advantage of DQS is precisely this universal character.

The decomposition of arbitrary Hamiltonians may at first seem problematic, since an arbitrary unitary operator requires exponentially many parameters to be specified, which is not efficient as its simulation will require exponential resources. However, as Feynman had predicted, any system consistent with general and special relativity evolves according to local interactions. An Hamiltonian evolution $\hat{H}$ over a system with N variables, with only local interactions, can be expressed as:

$$\hat{H} = \sum_{j=1}^{n} \hat{H}_j \tag{6}$$

Where each $\hat{H}_j$ acts on a limited space of dimension g_j containing at most ℓ of the N variables. By "local" interactions, the requirement is only that ℓ remains fixed as N increases; it is not necessary that the variables are spatially localized, which allows this procedure to include simulation of several non-relativistic models with long-range interactions. From

equation (6), the maximum number of distinct terms $\hat{H}_j$ is given by the binomial coefficient $\binom{N}{\ell} < N^\ell/\ell!$ which implies that n is polynomial in N. This is an ample upper bound for many practical cases, since for an Hamiltonian in which each variable interacts with at most ℓ nearest neighbours, $n \simeq N$.

The time evolution operator $U = e^{iHt}$, with H obtained from expression (6), can be divided into τ time steps, using the Trotter decomposition method (Trotter, 1959), as $e^{iHt} \approx (e^{iH_1 t/\tau} \dots e^{iH_n t/\tau})^\tau$. On a circuit model, this means that the local time evolution is simulated by local time evolution operators $e^{iH_1 t/\tau}, e^{iH_2 t/\tau}$ and so on up to $e^{iH_n t/\tau}$, and repeating τ times. To ensure that the simulation takes place within some desired accuracy, the time slicing needs to be regulated according to the Trotter-Suzuki formula (Suzuki, 1993):

$$e^{iHt} = (e^{iH_1 t/\tau} \dots e^{iH_n t/\tau})^\tau + \sum_{j'>j} [H_{j'}, H_j] t^2/2\tau + \sum_{k=3}^{\infty} err(k) \tag{7}$$

Where the higher order error terms $err(k)$ are bounded by $\|err(k)\|_{sup} \leq \tau \|Ht/\tau\|^k_{sup}/k!$. Here, $\left\|\hat{A}\right\|_{sup}$ represents the supremum, or maximum expectation value, of the operator $\hat{A}$ over the states of interest. Taking just the first term in equation (7) to approximate $e^{i\hat{H}t}$ results in a total error less than $\left\|\tau(e^{i\hat{H}t/\tau} - 1 - i\hat{H}t/\tau\right\|_{sup}$. For a given error ϵ and the second term of the equation, $\epsilon \propto t^2/\tau$. As such, a first order Trotter-Suzuki decomposition requires that $\tau \propto t^2/\epsilon$.

Once the accuracy within which the simulation is to take place is fixed, one can check that the simulation scales efficiently in the number of operations required. The size of the most general Hamiltonian $\hat{H}_j$ between ℓ variables is dependent on the dimensions of the individual variables, but will be bounded by a limiting size g. The Hamiltonians $\hat{H}$ and $\{\hat{H}_j\}$ can be time dependent as long as g remains fixed. As such, simulating $e^{iH_j t/\tau}$ requires g_j^2 operations, with $g_j \leq g$. According to equation (7), each local operator $\hat{H}_j$ is simulated τ times, bounding the total number of operations required for the simulation of $e^{i\hat{H}t}$ by $\tau n g^2$. Considering that $\tau \propto t^2/\epsilon$, the total number of operations, Op, is linearly proportional to

$$Op \propto t^2 n g^2/\epsilon \tag{8}$$

In this equation, only n is dependent on system size N, and from equation (6) it was established that n is polynomial in N, which proves that the number of operations is indeed efficient with problem size.

2.2 TOWARDS AN EFFICIENT IMPLEMENTATION OF DQS

LLoyd's proposal for digital quantum simulation of physical systems proves that a general many-body system can be efficiently simulated in terms of unitary operators with local interactions. While this work was a breakthrough for digital quantum simulation, efficient experimental implementations of simulation algorithms with current quantum devices require the observation of stricter conditions, both in terms of algebraic problems such as Hamiltonian decomposition and state preparation, and hardware-specific limitations, such as qubit mapping and error correction.

HAMILTONIAN DECOMPOSITION Several current models of quantum processors, such as those studied for experimental purposes in this work (described in section 4.1), have a 2-dimensional lattice architecture with only nearest neighbor interactions. Consequently, only one and two-qubit gates can be physically implemented.

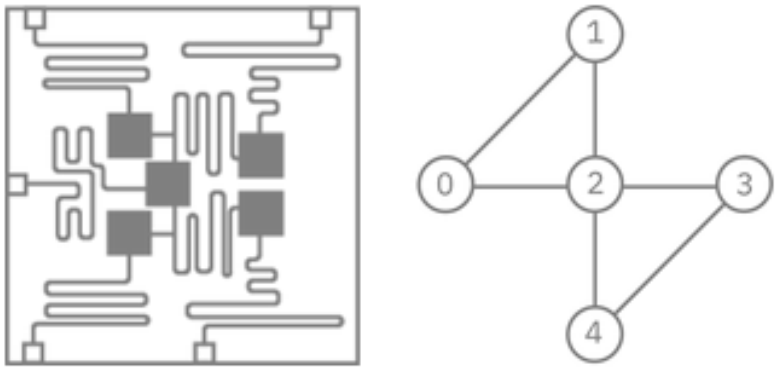


Figure 2.: Graphical representation of IBM's 5-qubit quantum device chip *Tenerife*, and corresponding qubit interaction model (IBM, 2018d).

One problem arising from this specification is: given two-qubit Hamiltonians, how can higher-dimensional qubit Hamiltonians be efficiently approximated?

In Bravyi et al. (2008), the efficient construction of higher order interactions from two-qubit Hamiltonians is tackled by using perturbation theory gadgets. In general, most unitary transformations on n, qubits will require an exponential number of gates. However, Bravyi et al. show that if one restricts the Hamiltonians of both system and simulation to be physically realistic, i.e. many-body qubit Hamiltonians $\hat{H} = \sum_j \hat{H}_j^\ell$ with a maximum of ℓ interactions per qubit, and where each qubit appears only in a constant number of the $\{\hat{H}_j^\ell\}$ terms, it is shown that the simulation is possible using two-body qubit Hamiltonians $\{\hat{H}_j^2\}$ with an absolute error given by $n\epsilon \left\|\hat{H}_j^{(\ell)}\right\|_{sup}$; where n is the number of qubits, ϵ is the precision, and

$\left\|\hat{H}_j^{(\ell)}\right\|_{sup}$ is the largest norm of the local interactions. For physical Hamiltonians, the ground state energy is proportional to $n\left\|\hat{H}_j^{(\ell)}\right\|_{sup}$. This allows for an efficient approximation of the ground state energy with arbitrarily small relative error ϵ.

In Raeisi et al. (2012), the first autonomous quantum algorithm for efficient simulation of Hamiltonian many-body quantum dynamics is presented. The algorithm designs a circuit for simulating the evolution generated by a general n-qubit ℓ-local Hamiltonian $\hat{H}^{(n)}$ withing a pre-specified tolerance ϵ that is efficient with the number of simulated qubits for fixed ℓ, and also scales near-optimally with the run-time t of the simulation. The algorithm specifically considers the case where the available gate set is composed of a two-qubit entanglement gate plus a finite number of one-qubit gates. The resultant circuits scale polynomially with the number of simulated qubits for fixed ℓ, with circuit size scaling near-optimally with the run time t of the simulation.

QUBIT ALLOCATION The dimension of physically implementable Hamiltonians is not the only limitation arising from architecture specifications such as demonstrated in fig. 2. The two-qubit entanglement gates composing the two-qubit Hamiltonians have to obey certain constraints, namely that certain quantum operations can only be applied to selected physical qubits (in fig. 2, for example, a two-qubit gate can be directly implemented between qubits 0 and 1, but not between qubits 0 and 3). Consequently, the logical qubits of a quantum circuit have to be mapped to the physical qubits of the quantum computer such that all operations can be conducted. Since it is often not possible to determine a mapping such that all constraints are satisfied throughout the whole circuit, this mapping may change over time. To this end, additional gates, e.g. realizing SWAP operations, are inserted in order to "move" the logical qubits to other physical ones. They affect the reliability of the circuit, as each further gate increases the potential for errors during the quantum computation, as well as the execution time of the quantum algorithm.

The qubit allocation problem is formally introduced in Siraichi et al. (2018). As of August 2018, the circuit compilation algorithm provided on IBM's software development for their quantum devices (IBM, 2018e) is based on random searching for a mapping satisfying all the constraints, and generally does not cope for circuits rich in two-qubit unitary gates, namely the set of circuits with gates whose algebra is contained in $SU(4)$, the special unitary group of 4×4 matrices with determinant 1. Motivated by this problem, in a very recent work, Zulehner and Wille (2018) propose and demonstrate a dedicated compiler that satisfies all the constraints imposed by IBM's quantum device architectures, allows for arbitrary qubit mappings, and generally outperforms IBM's current solutiofn. The provided compiler was adapted and used in the experimental part of this work.

STATE PREPARATION A crucial step in extracting useful results from a quantum simulation is starting with the right description of the system to be simulated, which is encoded in the initial state. An arbitrary pure state takes exponentially many parameters to specify, and hence exponential complexity to prepare.

The complexity of quantum circuits is often measured by the number of CNOT gates needed to perform the desired unitary operation. The reason for counting the number of CNOT gates is mainly experimental, since most proposed and demonstrated quantum processors only implement this operation for 2-qubit operations, and their realization is much more demanding and introduces more imperfections than the realization of one-qubit gates. It is known that, for a physical gate set composed of CNOT and single-qubit gates, the number of CNOT gates required to prepare an arbitrary n-qubit quantum state is exponential by a prefactor c, i.e $N_{CNOT} = c.2^n$. For the (currently) most efficient known algorithm for arbitrary state preparation, $c = 23/24$ (Plesch and Brukner, 2011).

Shende et al. (2006) propose a quantum algorithm that prepare an arbitrary n-qubit quantum state that is based on taking the inverse problem, i.e. designing a circuit that takes the desired pure state and transforms it into the basis state $|q_1 \cdots q_n\rangle = |0 \cdots 0\rangle$, and implementing the inverse operation, which is trivial using quantum gates. This is achieved by disentangling the least significant qubit into a separable product state $|q_1 \cdots q_{n-1}\rangle \otimes |0\rangle$, and recursively applying the algorithm to the $(n-1)$-qubit state. The algorithm uses $2^{n-1} - 2n$ CNOT gates, resulting in 10 CNOT gates for the 3-qubit state. Use of this algorithm is illustrated in the 3-qubit implementation (section 4.2.2).

The search algorithm by Grover (1996) can be extended for black-box state preparation. Soklakov and Schack (2006) demonstrate an algorithm for state preparation with arbitrary bounded fidelity that is efficient in the number of oracle calls, provided that the state itself can be described with an efficient (polynomial) number of parameters. Despite being asymptotically efficient, the number of sub-routines in these types of algorithms means that hundreds of operations could be necessary to prepare states with even a small amount of qubits. A more recent work by Sanders et al. (2018) focuses on reducing complexity of black-box state preparation algorithms for system sizes at the reach of NISQ devices.

QUANTUM ERROR CORRECTION Quantum computation is "fragile". A physical qubit does not hold its state indefinitely, but undergoes random bit-flips and loses its phase over time, i.e. undergoes decoherence. To overcome this and maintain a qubit state through longer times, researchers have come up with several quantum error correction techniques Knill et al. (2000), working on the principle of encoding a *logical qubit* within a specific number of *physical qubits*. The errors can then be detected and corrected without affecting the state of the logical qubit, which can hold information for longer than any of its underlying

physical qubits. The smallest number of physical qubits that can encode and protect a logical qubit against arbitrary errors is five Laflamme et al. (1996); Bennett et al. (1996).

The quantum threshold theorem Aharonov and Ben-Or (1997) states that there exists a threshold $\eta_0 > 0$, such that, for an arbitrary error tolerance $\epsilon > 0$ an "ideal" quantum circuit Q operating on n input qubits for t time steps using s one and two-qubit gates can be computed on a another quantum circuit Q' in the presence of local noise of error rate $\eta < \eta_0$ within ϵ total variation distance, with depth, size and width overheads which are polylogarithmic in n, s, t and $1/\epsilon$. Simply put, a quantum computer may efficiently suppress logical qubit error to arbitrarily low rates only if the required manipulations can be performed with a very low error, i.e. below a certain threshold. The theorem shows that, for error rates above the threshold, error correction procedures introduce more errors itself, than what is able to correct. If one manages to keep error rates under the threshold, the more physical qubits used to encode a logical one, the greater is the suppression of errors.

This threshold is dependent on the specific error correcting procedure. For one of the most prominent methods for error correction, called *surface code* (Fowler et al., 2012a,b) the threshold sits at approximately 1% (Wang et al., 2010a; Stephens, 2014). Surface code is the leading quantum error correction, since it requires only a 2-D square lattice of qubits that can interact with the nearest neighbor, an architecture implemented in several current noisy quantum computers, including all IBM quantum devices (section 4.1). For surface code error correction specifically, the minimum number of physical qubits necessary to encode a logical one is nine Horsman et al. (2012). One downside of surface code error correction is that, according to the Eastin-Knill theorem (Eastin and Knill, 2009), it cannot reliably achieve a universal set of gates without additional resources, which results in a stricter threshold for error correction. These additional resources are composed of high-fidelity ancilla qubits that need to be consistently produced and discarded, and are called *magic states* (Bravyi and Haah, 2012).

2.3 Analog quantum simulation

Another way to use quantum mechanics for the simulation of quantum systems is by analog quantum simulation. Succinctly, it involves taking a quantum system to mimic another by mapping the Hamiltonian of the system to be simulated, H_{sys} onto the controlled Hamiltonian of the quantum simulator, H_{sim}. Such a device is known as an analog quantum simulator, or AQS.

$$H_{sys} \longleftrightarrow H_{sim} \tag{9}$$

If a mapping between system and simulator is known, it can be used to construct an operator f such that $|\phi(0)\rangle$, the initial state of the system, can be mapped to the state of

the simulator $|\psi(0)\rangle$ by taking $|\psi(0)\rangle = f|\phi(0)\rangle$ (this is illustrated in fig. 1). After the simulation procedure is executed for time t, $|\psi(t)\rangle$ can be mapped back to $|\phi(t)\rangle$ via f^{-1}. For the Hamiltonians, $H_{sim} = fH_{sys}f^{-1}$.

The choice of a mapping depends on what needs to be simulated and on the degree of similarity in the dynamics of both systems; because of this, an AQS is generally a dedicated device restricted to simulating a limited class of quantum systems; the simulator typically acts as larger and more controllable "toy-model" of the system. Some representative studies on AQS are Fischer and Schützhold (2004); Porras and Cirac (2004); Zagoskin et al. (2007).

The accuracy of the simulation depends on the extent to which the simulator is able to reproduce the dynamics of the system to be simulated, since AQSs are usually emulating an effective many-body theory of the simulated system, they are limited by the extend to which the theory correctly captures the key physical properties of the system - a wrong model will always fail to produce meaningful results, no matter how flawless the implementation. It is in finding and applying the correct mapping that lies a big obstacle for AQS - sometimes the mapping is straightforward, but this is not always the case, and often researchers have to devise clever mappings involving additional externally applied fields, or ancillary systems, to mediate various interactions.

Analog quantum simulators have the advantage of being potentially useful even in the presence of larger error rates, up to a certain tolerance level. For example, one could use an AQS to study a quantum system with non-negligible noise and decoherence effects, and check if they lead to a given quantum phase transition. In this case, a qualitative answer might still be of interest. Even if the quantum simulator suffers from uncertainties in the control parameters, the phase transition under study could still be detected.

In comparison with DQS, initial-state preparation and measurement have not been thoroughly discussed and are often studied on a case-by-case basis. Because system and simulator are assumed to be very similar, it is expected that the preparation of the initial state can occur naturally in processes mimicking the natural relaxation of the simulated system to an equilibrium state. Furthermore, directly measuring some physical quantity of the simulator would yield information about its analogue in the simulated system. This may constitute an additional advantage of analog quantum simulation, since allowing for the direct measurement of its physical properties eliminates the need for computational processing and manipulation of results, as it happens with DQS.

2.4 Physical realizations

An extensive review of physical implementations and applications of quantum simulations is presented in Georgescu et al. (2014). One should note that any physical system than can be used as a quantum computer, such as the IBM devices presented in section 4.1, would also

work as a universal machine for digital quantum simulation. On the other hand, a quantum system that is not universal or a potential quantum computer could still implement analog quantum simulations.

A summary of possible and demonstrated physical implementations of quantum simulators follows, including distinguishing properties, strengths and main obstacles:

ATOMS AND IONS Neutral atoms in optical lattices (fig. 3 A)are well suited for mimicking solid-state systems, providing the highly desirable properties of being easily tunable and almost defect-free. A theoretical review (Lewenstein et al., 2007) discusses the potential of atoms in optical lattices in quantum simulators. Currently, addressing individual atoms in optical lattices is difficult, because the distance between neighboring lattice sites is comparable to the best achievable focusing widths of laser beams.

Ions can be trapped by electromagnetic fields, laser-cooled and manipulated with high precision for quantum simulation (Bohnet et al., 2016). Ion qubits have long coherence times, on the order of seconds, and sequences of high-fidelity quantum gates have been demonstrated experimentally (Lanyon et al., 2011).

Atoms in cavity arrays provide an alternative way of simulating the Bose-Hubbard model and quantum phase transitions, as well as spin models (Kay and Angelakis, 2008). The facility of single-site addressing, the use of only the natural hopping photon dynamics without external fields, and the recent experimental advances towards strong coupling, makes the prospect of using these arrays as efficient quantum simulators promising. As with ions, scaling may be an issue.

NUCLEAR AND ELECTRONIC SPINS Nuclear spins, manipulated by nuclear magnetic resonance (NMR) have been among the first experimental demonstrations of quantum algorithms and quantum simulation (Peng et al., 2009). Nuclear spin qubits have long coherence times, over 1 second, and high-fidelity quantum gates. Despite benefiting from well developed control techniques, NMR is not very flexible, and its main obstacle, as with most proposed implementations, is the lack of scalability.

Electron spins in semiconductor quantum dots (Hensgens et al., 2017) allow for flexible control over the confinement potential and can be excited optically. Since quantum dots with large tunnel coupling can act as "artificial molecules", they are particularly attractive for quantum simulation in chemistry (Lent et al., 2003). Quantum dot qubits benefit from similar decay times as nuclear spins, but may provide an advantage due to the very low temperatures (relative to the Fermi temperature) that can be reached and the natural long-range Coulomb interaction.

SUPERCONDUCTING CIRCUITS Superconducting circuits (You and Nori, 2011) have become a leading platform for the implementation of quantum information tasks. Quantum information can be encoded in different ways: in the number of superconducting electrons on a small island, in the direction of current around a loop, or oscillatory states of the circuit (fig. 3 H). Although macroscopic in size, these circuits display quantum behavior and can be seen as "artificial atoms", with the added advantage of being designed to tailor their characteristic frequencies and interaction strengths. State-of-the-art circuits have coherence times exceeding $100\mu s$, which is quite high considering the energy scales of the circuit are in the range of MHz up to 10 GHz. The fact that superconducting circuits can be produced in large numbers and "wired" together on a chip may facilitate the simulation of several lattice geometries.

Despite being a more recent and a comparatively less mature technology than trapped atoms/ions, or nuclear/electronic spins, superconducting quantum computing is in the basis of the currently most prominent private ventures into physical implementations of quantum computers, with research conducted separately by IBM (2016), Google (Castelvecchi, 2017), and Intel (2017).

OTHER SYSTEMS Photons can carry quantum information over long distances, hardly being affected by noise or decoherence. A serious drawback over optical implementation of qubits is the difficulty in implementing two-qubit gates and general entanglement procedures in the context of quantum computation, which limits flexibility and scalability of this approach. Nonetheless, entanglement with up to 8 photons has been experimentally demonstrated (Yao et al., 2012).

Other, less known, candidates for the implementation of quantum computation include NV centers in diamonds (Childress and Hanson, 2013), electrons trapped on the surface of helium (fig. 3 I) (Mostame and Schützhold, 2008), or chains of molecular nanomagnets controlled by external magnetic fields (Santini et al., 2011).

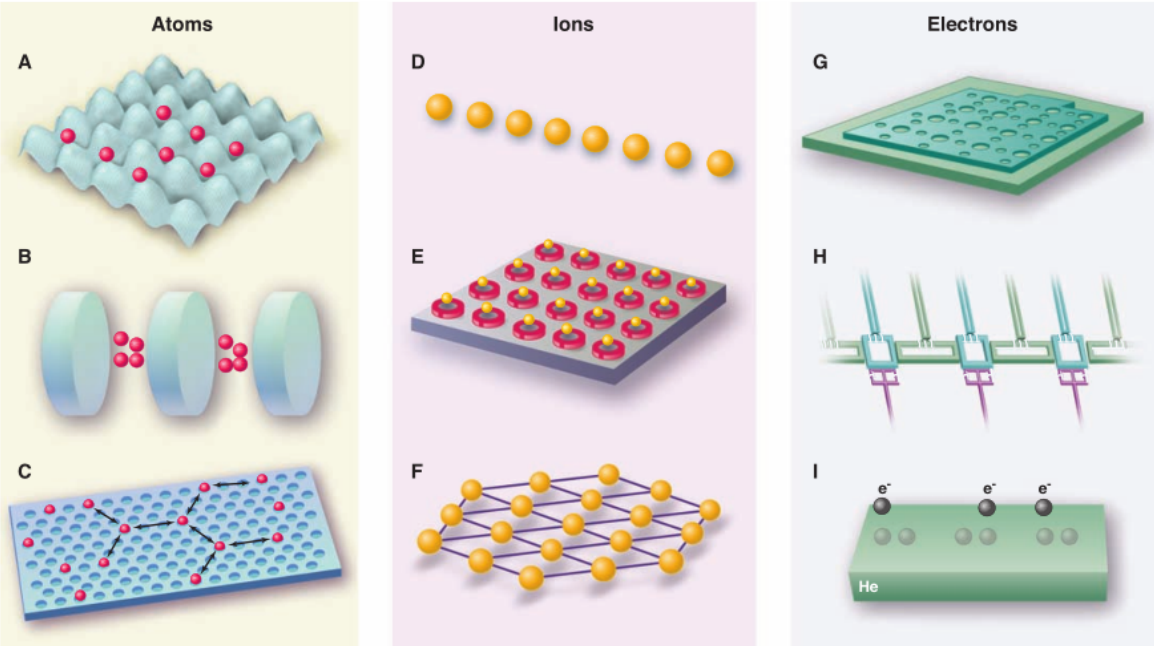


Figure 3.: One-dimensional or 2D arrays of qubits, plus control, could be used for the simulation of various models in condensed-matter physics. Examples of physical implementations that could implement such simulators include: atoms in optical lattices (A); 1D (B) or 2D (C) arrays of cavities; ions in linear ion chains (D); 2D arrays of planar traps (E); 2D Coulomb crystals (F); electrons in quantum dot arrays (G), in arrays of superconducting circuits (H), or trapped on the surface of liquid helium (I) (Buluta and Nori, 2009).

2.5 APPLICATIONS

Quantum simulators have numerous known applications in diverse areas of physics and chemistry (Lanyon et al., 2010), and even biology (Åqvist and Warshel, 1993; Dror et al., 2012). A summary of applications grouped by scientific fields of physics follows.

CONDENSED-MATTER PHYSICS One widely studied application of quantum simulations is the simulation of models in condensed matter physics. For several models in this class, an array of qubits plus their controls would make an ideal quantum simulator, since it can be thought of as a simplified, magnified lattice structure of a solid, that can be manipulated in different ways to test various models, such as changing the dimensionality or geometry of the array. Such an array could be realized, for example, with atoms in optical lattices (Preiss et al., 2015), atoms in arrays of cavities (Qin and Nori, 2016; Angelakis et al., 2007), ions in microtrap arrays (Chiaverini and Lybarger Jr, 2008), or in two-dimensional crystals (Porras and Cirac, 2006), or even with electrons in arrays of quantum dots (Byrnes et al., 2008). The simulator's control fields can be applied individually or to the entire array, directly realizing

the desired Hamiltonian (AQS) or reconstructing it out of one and two-qubit gates (DQS). The larger distances between qubits, relative to the simulated lattices, make quantum simulators more controllable and easier to measure. The magnification factor may reach three orders of magnitude. A representation of these examples is shown in figure 3.

Specifically, quantum simulators can help with current challenges in understanding models such as:

a) *Hubbard model*, which is the simplest model of interaction of particles on a lattice. For larger numbers of particles in more than one dimension, the model is difficult to treat with classical resources. Somma et al. (2002) considers the simulation of the Hubbard model in the context of DQS.

b) *Spin models*, used in physics mainly to explain magnetism, can be studied both through DQS (Lanyon et al., 2011; Tsomokos et al., 2010) or AQS (Monroe et al., 2015).

c) *Quantum phase transitions* describe an abrupt change in the ground state of the many-body system governed by its quantum fluctuations. They are an interesting and important subject, even if difficult to investigate both through classical simulation or experimental methods. A recent, 53-qubit analog simulation for the observation of this phenomenon was demonstrated recently (Zhang et al., 2017).

d) *Disordered and frustrated systems.* Disordered systems appear in many difficult problems in condensed-matter physics, such as transport, conductivity, spin glasses and some models of high-T_C superconductivity (De Nicola et al., 2014). Geometric frustration refers to the regime in which the geometric properties of the crystal lattice forbid the simultaneous minimization of all the interaction energies acting in a given region. As an example, a proposal making use of photon quantum simulation has been put forward (Ma et al., 2014).

e) *Spin glasses* typically occur when the interactions between spins are ferromagnetic for some bonds and anti-ferromagnetic for others, which causes spin orientation to become random and almost "frozen" in time. How much speedup may be gained with the use of quantum simulation is not a trivial question (Heim et al., 2015). There are analog simulation proposals for specific models (Tsomokos et al., 2008).

f) *Superconductivity.* The high-temperature superconductivity of compounds containing copper-oxide planes, for example, is still a puzzle that might be solved using large-scale simulations. The CuO_2 plane in a high-T_C superconductor could be studied through AQS using arrays of quantum dots (Manousakis, 2002).

HIGH-ENERGY PHYSICS The field of high-energy physics has also seen developments as an application of quantum simulators; Boghosian and Taylor IV (1998) originally suggested

of the use of quantum simulators for the study of relativistic quantum systems, such as gauge fields or Dirac fermions. Zitterbewegung (i.e. the hypothetical rapid oscillatory motion of elementary particles obeying the Dirac equation) has been simulated with a trapped ion (Gerritsma et al., 2010). The simulation of gauge theories, a very computationally intensive problem, has also been experimentally demonstrated on quantum devices (Martinez et al., 2016).

COSMOLOGY Analog models of gravity and cosmology models could also benefit from the use of quantum simulation. Studies have been made on the analogue of cosmological particle creation with trapped ions (Fey et al., 2018), or the analogue of quantum field effects in cosmological spacetime (Menicucci et al., 2010). Furthermore, the analogues of Hawking radiation could be investigated with several options, such as atoms (Giovanazzi, 2005) or superconducting circuits (Nation et al., 2009).

ATOMIC PHYSICS As mentioned in section 2.4, there are deep parallels between natural atoms and the atom-like properties formed by electrons in superconducting circuits. While natural atoms are driven using visible or microwave photons to excite electrons, these "artificial atoms" are driven by currents, voltage and microwave photons, allowing for the control of electron tunneling across Josephson junctions. This allows for the tuning of properties such as dipole moment or particular transition frequencies. Superconducting circuits can be used to test Bell and Mermin inequalities (Alsina and Latorre, 2016), Schrödinger-cat states, or study Landau-Zener-Stückelberg interferometry (Shevchenko et al., 2010). Simulation of the Schrodinger equation may be used to find the allowed energy levels of quantum mechanical systems such as atoms (as is discussed in detail and experimentally demonstrated in section 4.2), or transistors.

QUANTUM CHEMISTRY With the rising availability of quantum processors in this decade, interest in quantum simulation for quantum chemistry has greatly increased, quickly becoming one of its most anticipated applications (Mueck, 2015). Currently known exact first-principles calculations of molecular properties are intractable because their computational cost grows exponentially with both the number of atoms and basis set size. Lu et al. (2012) review the theory and early forays into experimental quantum simulation in quantum chemistry. An efficient quantum simulation of molecular energies was demonstrated experimentally by O'Malley et al. (2016). Promising experimental research into determination of molecular ground states has been recently demonstrated, through both analog (Argüello-Luengo et al., 2018) and digital quantum simulations (Hempel et al., 2018).

OTHER APPLICATIONS Classical simulation of the dynamics of open quantum systems is even more costly than that of closed quantum systems, since solving the Lindblad equation requires quadratically more resources than the Schrödinger equation for the same quantum system. As suggested by Lloyd (1996), one could explore the natural noise and decoherence properties of the simulator to aid in the simulation of open quantum systems. For instance, if the noise level of the simulator is lower than the noise level of the simulated system, it is straightforward to artificially supplement noise in the simulator to achieve a more faithful simulation; this concept has been demonstrated experimentally (Li et al., 2013). General methods for simulating the markovian dynamics of open quantum systems have also been investigated (Wang et al., 2011; Di Candia et al., 2015).

Several other topics in physics research are being discussed in the context of quantum simulation, such as boson sampling (Moylett and Turner, 2018), dynamical maps and transitions to quantum chaos (Schindler et al., 2013), neutrino oscillations (Di Molfetta and Pérez, 2016), or brownian motion (Maniscalco et al., 2004). Quantum mechanical models of biological processes also stand to benefit from the advancement of quantum simulation technology (Dorner et al., 2012), and experimental simulations have been demonstrated (Pearson et al., 2016).

2.6 SUMMARY

This chapter presents the theory behind the concept of quantum simulation, i.e. the use of a controllable quantum system to simulate another quantum system. An overview of physical implementations and applications of quantum simulators is shown.

An analog quantum simulation involves taking a quantum system and manipulating its Hamiltonian so it is possible to map it into the Hamiltonian of a system to be simulated; the simulator is thus restricted to simulating a limited class of systems. A digital quantum simulation may be performed by a quantum computer (which is regarded as a universal quantum simulator) by discretizing the time evolution of the system and encoding its state into a set of quantum bits.

Digital simulation of an Hamiltonian is, in theory, efficient up to an arbitrary degree of error. However, when discussing a truly efficient experimental implementation on current quantum computers, other steps comprising a simulation need to be taken into account, such as state preparation, unitary decomposition, qubit mapping and error correction.

3

QUANTUM CHARACTERIZATION, VERIFICATION AND VALIDATION

While the computational power of quantum simulation opens a new field of possibilities in science, it also comes with an important challenge: that of checking the accuracy of the simulation, particularly when a classical efficient simulation is not available.

In case of a quantum computation that solves a problem in NP, one can in retrospect efficiently verify the solution by classical means. However, not all interesting problems which quantum computation might solve are decision problems. One particularly important question, in quantum simulation particularly, is whether one has achieved a desired quantum state preparation. It may also be useful to obtain information about intermediate steps of a quantum computation or simulation.

In the context of checking the results of a quantum computation or simulation, several approaches have been discussed in the literature, each with different motivations and goals, requirements, and varying degrees of complexity. Quantum procedures designed to certify and calibrate designed performance fall into the general term of *quantum characterization, verification and validation* protocols (QCVV). With respect to the aim of the procedure, this work distinguishes between the three concepts.

Characterization of a quantum state aims to fully determine the mathematical description of the state. *Characterization* of a quantum process aims to determine the dynamics, properties and qualities of the quantum operation. These techniques, by definition, return the greatest amount of information about a given quantum state/process, but typically require more resources. They include quantum state tomography and quantum process tomography techniques, as well as randomized benchmarking.

Verification, or certification, procedures for quantum devices aim to check the correctness of the simulation, providing an answer to the question: *is the device working precisely as anticipated?*. These procedures allow the verifier to test just how reliable a given simulator is, providing a degree of *trust* in the computation or simulation itself.

Validation protocols aim to check the validity of solutions, i.e. *did the simulation produce a valid solution?*. A validation protocol aims to check the validity of a particular set of results from a computation or simulation experiment.

These concepts are not mutually exclusive, and a given procedure may address more than one of these definitions (e.g. quantum process tomography not only characterizes a given quantum process, but may also be used to verify it's fidelity or entanglement capabilities, among other quantities); there is no definitive consensus on how to categorize QCVV techniques in the literature, and there is also some overlap between the concepts themselves. This is due to the multidisciplinary approach and different motivations behind several techniques, which may find their roots in fields such as physics and engineering, or computational science and complexity theory.

In this chapter, some techniques that are relevant to the experimental work performed are discussed in greater detail. At the end of the chapter, other techniques are presented which may not have been demonstrated yet, but show some promise due to their higher sophistication, potential, or generally lower complexity requirements with system size.

3.1 QUANTUM STATE TOMOGRAPHY

Quantum state tomography is a general notion describing a set of procedures and statistical methods, using experimental data from a set of measurements, to fully determine a density matrix ρ describing an unknown quantum state in a finite-dimensional Hilbert space.

A density matrix ρ allows for a mathematical description of both pure and mixed quantum states. While a pure quantum state may be fully described by a *ket* vector, a mixed quantum state is a statistical mixture of pure quantum states $|\psi_i\rangle$, each one occurring with probability p_i; this is different from a quantum superposition, which occurs due to quantum phenomena and exactly describes the state. By contrast, a mixed state is a combination of probabilities of each possible quantum state, and it useful in cases where one has insufficient information about the state, i.e. when one part of the quantum system is inaccessible, or when noise and decoherence processes occur. A density matrix is mathematically described as:

$$\rho = \sum_i p_i \left|\psi_i\right\rangle \left\langle\psi_i\right| \tag{10}$$

A density matrix ρ has properties: $tr(\rho) = 1$ (i.e. the probabilities p_i sum to 1), and $\rho = \rho^\dagger$, $\rho \succeq 0$ (i.e. all eigenvalues are real and non-negative). $tr(A)$, representing the trace of an $n \times n$ matrix A, is defined to be the sum of the elements on the main diagonal of A: $tr(A) = \sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn}$.

A single copy of an unknown state does not allow for its characterization, even if the state is that of a single qubit, since no single measurement can distinguish between non-orthogonal quantum states such as $|0\rangle$ and $(|0\rangle + |1\rangle)/\sqrt{2}$ with certainty. To estimate ρ with arbitrary precision, the source system should be able to consistently repeat the preparation of the

same quantum state, and the measurements must be tomographically complete, i.e. the measurement operators must form an operator basis on the system's Hilbert space.

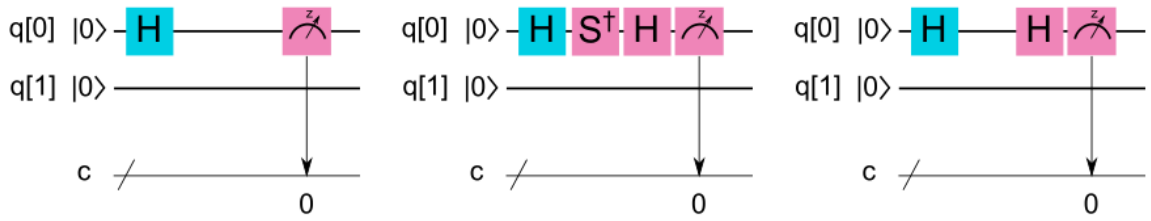


Figure 4.: Set of tomographically complete measurements for a single qubit state q[0] after application of a single Hadamard gate, in the Z, Y and X basis (Coles et al., 2018).

For example, a density matrix describing a single qubit system can be written as:

$$\rho = \frac{tr(\rho)I + tr(X\rho)X + tr(Y\rho)Y + tr(Z\rho)Z}{2} \tag{11}$$

Where $tr(A\rho)$ is interpreted as the average value of observable A. To estimate it, the key is to measure the observable a large number of times. For example, the estimation of $tr(Z\rho)$ involves measuring the observable Z, m times, each time obtaining an outcome z_i equal to +1 or -1. One can calculate the average of these quantities as $\sum_i z_i/m$, and use it as an estimate for the value of $tr(Z\rho)$. According to the central limit theorem, with large m this estimate approximates to a Gaussian distribution with mean $tr(Z\rho)$ and standard deviation $\Delta Z/\sqrt{m}$, where ΔZ is the standard deviation for a single measurement in Z, which is upper bounded by 1. The standard deviation for the estimate is then $1/\sqrt{2}$. The quantities $tr(X\rho)$ and $tr(Y\rho)$ can be estimated by repeating the procedure for these observables, allowing one to obtain an estimate for ρ with a degree of confidence dependent on the sample size.

Equation (11) can be generalized to an arbitrary density matrix on n qubits as:

$$\rho = \sum_{\vec{v_k}} \frac{\mathrm{tr}(\sigma_{v_1} \otimes \cdots \otimes \sigma_{v_n}\rho)\sigma_{v_1} \otimes \cdots \otimes \sigma_{v_n}}{2^n} \tag{12}$$

With sum occurring over vectors $\vec{v_k} = (v_1, \cdots, v_n)$ where the entries v_i are chosen from the set $0, 1, 2, 3$ corresponding to the Pauli operators, which can be represented mathematically as:

$$\sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix};\ \sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix};\ \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix};\ \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}; \tag{13}$$

Here, $\{\sigma_0, \sigma_1, \sigma_2, \sigma_2\}$ correspond to the observables $\{I, X, Y, Z\}$, respectively. For each vector $\vec{v}$, describing a measurement operator, there are two measurement outcomes, and each can be taken as projector E_k. The set of E_k form a *Positive Operator-Valued Measure* (POVM),

satisfying $\sum E_k = I$. Admitting we have m measurements for each operator, with m_j occurrences for each projector, the measurement frequency ω_j can be determined as $\omega_k = m_j/m$. The problem of characterizing a quantum state can now be defined as that of reconstructing ρ from the coupled set of projectors and measurement frequencies, $\{E_k, \omega_k\}$, i.e. matching $tr(E_k\rho)$ and ω_k. Several methods have been researched:

LINEAR INVERSION This method, derived from Born's rule (Born, 1926), is in practice the simplest for quantum state tomography, and it aims at determining the density matrix ρ by inverting the system of equations $tr(E_i\rho) = \omega_i$.

For a measurement outcome projector E_i and the density matrix ρ describing the system, Born's rule states that the probability of obtaining outcome E_i is given by $\mathrm{P}(E_i|\rho) = tr(E_i\rho)$. Given a histogram of observations for each measurement, one can use the measurement frequency ω_i as an approximation, $p_i = \omega_i$, to $\mathrm{P}(E_i|\rho)$ for each E_i:

$$\begin{pmatrix} E_1 \cdot \rho \\ E_2 \cdot \rho \\ E_3 \cdot \rho \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathrm{P}(E_1|\rho) \\ \mathrm{P}(E_2|\rho) \\ \mathrm{P}(E_3|\rho) \\ \vdots \end{pmatrix} \approx \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \end{pmatrix} = \vec{p} \tag{14}$$

One can then invert the system of equations to determine ρ. Although relatively fast, this method does not guarantee a valid density matrix, since it might contain negative eigenvalues, or return a sum of probabilities surpassing 1.

LINEAR REGRESSION This approach aims at correcting the disadvantages of linear inversion and minimizing computational complexity, by converting the quantum state tomography problem into a constrained quadratic optimization problem to obtain an estimation $\hat{\rho}$ of the density matrix:

$$\hat{\rho} = \underset{\rho}{argmin} \sum_i [\mathrm{tr}(E_i\rho) - \omega_i]^2 \ \text{ s.t. } \mathrm{tr}(\rho) = 1 \text{ and } \rho \succeq 0 \tag{15}$$

Where $\underset{\rho}{argmin}$ corresponds to the density matrix ρ for which the succeeding expression attains its lowest value. With linear regression the current estimation can be updated as the data is processed, reducing space requirements since not all terms need to be stored. However, the function takes the assumption that the residuals are Gaussian-distributed, which does not hold for a finite number of measurements. A proposal for a linear regression estimation procedure is presented in Qi et al. (2013).

MAXIMUM LIKELIHOOD First proposed in Hradil (1997), maximum likelihood estimation algorithms are currently the most popular and researched methods of estimation. As with linear regression, the domain of the density matrices is restricted to the proper space, but the aim is at finding the density matrix which maximizes the likelihood function to the experimental results. Admitting the measured states $\{|\psi_i\rangle \langle\psi_i|\}$ (each corresponding to a projector E_i) have been measured with frequencies ω_i, the likelihood, $L(\rho')$ associated with a given state ρ' is

$$L(\rho') = \prod_i \langle\psi_i| \rho' |\psi_i\rangle^{\omega_i} \tag{16}$$

The problem can be framed as that of maximizing the log-probability of observations (James et al., 2005), i.e.:

$$\hat{\rho} = \underset{\rho}{argmax} \sum_i \omega_i \ln \mathrm{tr}(E_i \rho) \;\; \text{s.t. } \mathrm{tr}(\rho) = 1 \text{ and } \rho \succeq 0 \tag{17}$$

Finding the expression of ρ for which this function attains its maximum value is non-trivial, and generally involves iterative methods. Maximum likelihood estimation is often stated to be comparatively slow (Scholten and Blume-Kohout, 2018), and recent research has attempted to find faster, less resource-intensive procedures for this method. In this work, the maximum likelihood estimation method proposed by Smolin et al. (2012) was implemented experimentally and is detailed in section 4.3.

OTHER STATE TOMOGRAPHY METHODS With the rising interest in quantum computation, several other types of quantum tomography procedures have been explored. *Bayesian mean estimation methods* address some of the problems of maximum likelihood estimation, by starting with a likelihood function but also allowing for a function describing the experimenter's prior knowledge about the system, which serves as a weight. The technique also provides optimal solutions which are *honest* in the sense that error bars are included in the estimate. In practice, it is not always clear how to choose these priors; Markov Chain Monte Carlo methods are known to be analytically intractable. However, recent research on Bayesian approaches (Kravtsov et al., 2013) have shown some encouraging results. A review and experimental demonstration of Bayesian estimation for quantum tomography is provided in Granade et al. (2016).

The methods described above use predetermined sets of measurements. One can try to benefit from using the information about the unknown state, obtained from the previous measurements, to optimize the next ones. This technique forms a new class of interactive tomography methods with a stronger focus on optimization and reducing computational complexity called *adaptive quantum tomography*; an overview is given in Straupe (2016).

In data science, machine learning methods have shown promising results in compressing high-dimensional data into low-dimension representations. The same principle may be applied to quantum tomography procedures, which can make use of machine learning and neural network algorithms, for more efficient data processing. Such a procedure is demonstrated in Torlai et al. (2018).

While the reconstruction of an unknown state may be useful in itself, state tomography techniques also allow to quantitatively evaluate the quality of a given state preparation. To measure the "closeness" between a state reconstruction and a desired pure quantum state, the quantum state fidelity function may be used.

Definition 3.1. The quantum state fidelity $F(\psi, \rho)$ between a pure quantum state $|\psi\rangle$ and a density matrix ρ is expressed as:

$$F(\psi, \rho) = \sqrt{\langle\psi|\ \rho\ |\psi\rangle} \tag{18}$$

For any two ψ and ρ, $0 \leq F(\psi, \rho) \leq 1$, with $F(\psi, \rho) = 0$ for two orthogonal states, and $F(\psi, \rho) = 1$ if ψ and ρ represent the same quantum state.

A reconstructed state can also be measured for its purity, which can be held as a metric for the introduction of noise processes during a computation/simulation particularly if the desired final state is expected to be pure.

Definition 3.2. The purity γ of a quantum state represented by a density matrix ρ is a scalar expressed as:

$$\gamma \equiv tr(\rho^2) \tag{19}$$

With $tr(\rho^2)$ representing the trace of the squared density matrix ρ. For any density matrix ρ representing a state in the Hilbert space with d dimensions, the purity is bounded by $\frac{1}{d} \leq \gamma \leq 1$, with the lower bound representing a completely mixed state and $\gamma = 1$ for a pure state.

In general, full tomography of quantum states is computationally intensive, since the set of measurement operators grow linearly with the number of dimensions of a system's Hilbert space, which grows exponentially with the number of qubits. Some techniques have been devised which sacrifice accuracy for lighter resources, but quantum tomography of multi-qubit states remains a very hard task, and it has not been experimentally demonstrated for states with over 14 qubits (Straupe, 2016). In Lanyon et al. (2017), a state tomography technique is provided which scales polynomially with system size, but it is restricted to state which can be written in the form of a matrix product state, whose description increases only polynomially with system size.

3.2 QUANTUM PROCESS TOMOGRAPHY

Quantum process tomography (QPT) allows for the characterization of the dynamics of a quantum system, which may then be checked against the mathematically predicted model for the system. It is the quantum analogue of *system identification* of classical systems, and it may be used, for example, to characterize the performance of an implemented quantum gate, or different noise processes in a system.

The first approach to performing quantum process tomography was proposed in Chuang and Nielsen (1997) and involves preparing an ensemble of quantum states, sending them through the process, and using quantum state tomography to identify the resultant states. The experimental procedure is as follows: for a system with a state space of d dimensions, d^2 pure orthogonal quantum states $|\psi_1\rangle, \cdots, |\psi_{d^2}\rangle$ are chosen so that the corresponding density matrices form a *basis set* for the space of matrices. Each state $|\psi_j\rangle$ is prepared and subjected to the quantum process in study. After the operation, one can use quantum state tomography techniques to determine the output state $\mathcal{E}(|\psi_j\rangle\langle\psi_j|)$. After repeating the process for all chosen states, the quantum operator $\mathcal{E}$ is determined by a linear extension to all states; additional processing is necessary to obtain a mathematical representation of the linear mapping from the experimental data.

Considering that the operator $\mathcal{E}$ maps an initial density matrix ρ_{in} to an output density matrix ρ_{out}, i.e.:

$$\rho_{in} \rightarrow \rho_{out} \Rightarrow \rho_{in} \rightarrow \frac{\mathcal{E}(\rho_{in})}{tr(\mathcal{E}(\rho_{in}))} \tag{20}$$

One can use a set of operation elements A_i to describe the operator using a so called *operator-sum representation*:

$$\mathcal{E}(\rho) = \sum_i A_i \rho A_i^\dagger \tag{21}$$

To relate the operation elements to measurable parameters, it is convenient to consider a description of $\mathcal{E}$ using a *fixed* set of operators $\tilde{A}_i$ which form a basis for the set of operators on the state space: $A_i = \sum_m a_{im}\tilde{A}_m$ for some set of complex numbers a_{im} which allows equation (21) to be written as:

$$\mathcal{E}(\rho) = \sum_{mn} \tilde{A}_m \, \rho \, \tilde{A}_n^\dagger \, \chi_{mn} \tag{22}$$

Where $\chi_{mn} \equiv \sum_i a_{im} a_{in}$ is an error correlation matrix which is positive Hermitian by definition. This shows that $\mathcal{E}$ can be completely described by a complex number matrix, χ, once the set of operators $\tilde{A}_i$ has been fixed. In general, χ will contain $d^4 - d^2$ independent parameters, because a general linear map between $d \times d$ matrices is described by d^4 parameters,

but there are d^2 additional constraints due to the fact that the trace of ρ sums to 1. For each input state density matrix $\rho_j = |\psi_j\rangle \langle\psi_j|$, the operator $\mathcal{E}(\rho_j)$ can be expressed as a linear combination of the basis states ρ_k:

$$\mathcal{E}(\rho_j) = \sum_k \lambda_{jk} \rho_k \tag{23}$$

Since $\mathcal{E}(\rho_j)$ is known from performing quantum state tomography on the set of output states, λ_{jk} can be determined. From equation (22) on may write:

$$\tilde{A}_m \, \rho_j \, \tilde{A}_n^\dagger = \sum_k \beta_{jk}^{mn} \rho_k \tag{24}$$

Where β_{jk}^{mn} are complex numbers determined from the $\tilde{A}_m$ and the ρ_j operators. Combining equations (23) and (24), and since each ρ_j is independent, it follows that for each k:

$$\sum_{mn} \beta_{jk}^{mn} \chi_{mn} = \lambda_{jk} \tag{25}$$

This relation is a sufficient condition for the matrix χ to give the correct quantum operation $\mathcal{E}$; $\lambda_j k$ is obtained from equation (23), and χ can be determined as:

$$\chi = \beta^{-1} \, \lambda \tag{26}$$

One may think of χ and λ as column vectors of dimension $d^4 \times 1$, and β as a $d^4 \times d^4$ matrix with columns indexed by mn and rows indexed by ij.

Concurrently to the work by Nielsen and Chuang, Poyatos et al. (1997) describes a procedure for the complete characterization of a quantum process in an open quantum system, particularly for the case of a universal two-qubit gate. The procedure can be scaled to a quantum gate involving an arbitrary number of qubits, since it has been shown that any computation on a universal quantum computer can be decomposed using only one and two-qubit quantum gates.

The methods of process tomography described above may be thought of as *indirect* methods of characterization of quantum dynamics, since they require the use of quantum state tomography to reconstruct the quantum process.

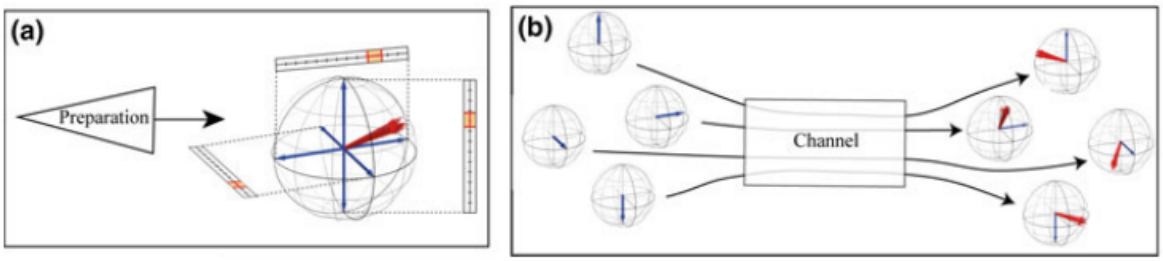


Figure 5.: Bloch sphere representation of quantum tomography procedures; a) quantum state tomography aims to characterize a quantum state preparation, by subjecting the state to a set of well calibrated measurements; b) quantum process tomography aims to reconstruct a description of a unitary operation, by reconstructing the output states (e.g. using state tomography) for a set of well-characterized input states (Ringbauer, 2017).

By contrast, *direct* methods of characterization of quantum dynamics provide a full characterization of quantum systems without any state tomography. A review and resource analysis for different strategies of quantum process tomography is given in Mohseni et al. (2008).

When applying quantum process tomography techniques to evaluate the performance of a quantum operation, the resulting mathematical description of the operation may be used in performance metrics such as gate fidelity, which measures the extent to what an experimentally implemented operator matches an ideal one.

Definition 3.3. Average gate fidelity between an experimentally implemented operation $\mathcal{E}$ and an ideal operation U, for a set of input states $|\psi\rangle$ is given by the expression:

$$F_G(U,\mathcal{E}) = \overline{\langle\psi| U^\dagger \mathcal{E}(|\psi\rangle\langle\psi|) U |\psi\rangle} \tag{27}$$

Gate fidelity values are bounded by $0 \leq F_G \leq 1$, with 0 meaning a completely orthogonal operation to the ideal one, and $F_G(U,U) = 1$. One may chose, instead, to estimate a minimum quantum gate fidelity by simply using the input $|\psi\rangle$ for which gate fidelity attains is lowest value.

As with quantum state tomography, general quantum process tomography techniques are very resource intensive, since they scale polynomially with the number of dimensions of the Hilbert space of the system, which itself grows exponentially with system size, i.e. number of qubits. This means general quantum process tomography is not efficient.

3.3 Randomized benchmarking

An important challenge of quantum computing experiments is to physically realize gates that have low error. From the quantum threshold theorem and the prospect of fault tolerant quan-

tum computing, the need for error rates below 10^{-2} becomes evident; the current consensus is that one should aim for error rates below $10^{-}4$ to avoid excessive resource overhead.

A possible approach to verifying error rates of a quantum gate is to use process tomography to characterize it and establish its behavior. This requires that the single-qubit gates and measurement operators employed in the procedure have lower error than the bound to be established on the gate under study, which makes QPT particularly sensitive to state preparation and measurement errors. Additionally, complete quantum gate characterization rapidly becomes experimentally intractable due to the exponentially large Hilbert space. To characterize error rates, however, a full mathematical description of the operation is not necessary.

Randomization has been suggested as a tool for characterizing features of quantum noise in Emerson et al. (2005). The authors propose implementing random unitary operators U followed by their inverses U^{-1}. Under the assumption that the noise model can be represented by a quantum operation acting independently between the implementations of U and U^{-1}, the effect of the randomization is to depolarize the noise. The average fidelity of the process applied to a pure initial state is the same as the average over pure states of the fidelity of the noise operation.

Randomized benchmarking (RB), as proposed in Knill et al. (2008), is designed for the estimation of average gate fidelity and simplifies this procedure by restricting the unitaries to Clifford gates and by not requiring that the sequence is strictly self-inverting. It is specifically tailored to compensate for preparation and measurement errors by considering only the exponential decay of sequences of random gates, but it comes at the cost of only obtaining information about the average gate error over the Clifford group, although some alternative approaches have been recently devised for extending RB to estimate the error of a single, arbitrary gate (Magesan et al., 2012).

A straightforward procedure for randomized benchmarking of a quantum processor can be outlined in three steps:

1. Perform a randomly chosen sequence of Clifford gates that ought to return the processor to its initial state;

2. Perform a measurement at the end of each sequence to see whether the device returned to the initial state; repeat steps 1 and 2 for a number of sequences;

3. Plot the observed "survival" probabilities against sequence length, and fit the results to an exponential decay curve. A decay rate r is then estimated from increase in error probability of the final measurements as a function of sequence length.

The generally accepted theory behind randomized benchmarking suggests that, for small error rates, r is approximately equal to the average, over all n-qubit Clifford gates, of *gate*

infidelity between the experimentally implemented gates and their ideal counterparts. Average gate infidelity is simply $1 - F_G$, the average gate fidelity as expressed in definition (3.3). A discussion on the actual significance of r and how it relates to average rate infidelity is presented in Proctor et al. (2017).

The reported gate errors for the quantum devices detailed in section 4.1 were obtained from the randomized benchmarking procedure detailed in Gambetta et al. (2012), which accounts for errors arising from cross-talk and unwanted interactions in multi-qubit systems. For a system with n qubits and dimension size $d = 2^n$, with a decay rate r from the exponential fit to the data obtained from the randomized benchmarking procedure, the average error rate η is estimated as:

$$\eta = \frac{(d-1)(1-r)}{d} \tag{28}$$

In Sanders et al. (2015) it is shown that the upper and lower bounds for actual quantum gate error rates may vary greatly and should be taken into account for estimations of gate errors in the context of the quantum threshold theory. These bounds can me made more accurate by making use of a proposed quantity called "Pauli distance", estimated from verification procedures akin to randomized benchmarking.

Randomized benchmarking provides a method for benchmarking the set of Clifford gates that is efficient with the number of qubits. While benchmarking the full unitary group would be ideal, this is an inefficient task since just generating a Haar-random unitary operator is inefficient in n. However, since the unitary group can be generated by adding just one single-qubit rotation not in the Clifford group, a benchmark for the Clifford group can actually provide useful information regarding a benchmark for a generating set of the full unitary group. In addition, it has been shown that any unitary operation can be implemented using Clifford gates, and single-qubit ancilla state (Bravyi and Kitaev, 2005).

3.4 Other verification and validation methods

The experimental, and data post-processing, requirements of full quantum tomography have made the study of alternative validation methods an active field of research in the last decade. The broad range of approaches stems for the multidisciplinary nature of quantum information and computation itself and it has been found that several concepts around validation and verification of classical systems can be translated into the quantum mechanical systems.

With the aim of verifying the preparation of a desired quantum state ρ by an experimental quantum system, (Flammia and Liu, 2011) propose the preparation of a number of copies of the state, which are then measured in a random subset of Pauli observables chosen according to an "importance weighting" rule, i.e. by selecting Pauli operators that are most likely to detect deviations from ρ. Although, for a system with n qubits, there are 4^n distinct Pauli

operators, sampling a constant number of them is enough to estimate the fidelity $F(\rho, \sigma)$ up to a constant additive error, for an arbitrary σ produced experimentally. The number of repetitions for each measurement depends on the state ρ, and in the worst case, it is $O(2^n)$[1], being much smaller in various cases of interest, such as stabilizer states, where it is constant, or the W state, where it is quadratic with n. Note however, that the procedure depends on having a theoretical ideal state ρ which will be compared against the experimental results.

Similar findings were independently demonstrated in da Silva et al. (2011), where the authors distinguish between two types of characterization: *certification*, which consists of estimating the fidelity between an experimental device and some theoretical target; and *learning*, which consists of identifying the theoretical description from a restricted set of possibilities that best matches the experimental data. For some "variational" states that can be specified with a small number of parameters, examples where these parameters can be extracted directly from experiments are provided. It is also shown that stabilizer states and Clifford group operations can be *learned* efficiently. For systems evolving according to local Hamiltonians, with $\frac{\partial}{\partial t}\hat{\rho} = \mathcal{G}\hat{\rho}$, the time evolution generator $\mathcal{G}$ can be *learned* with a number of experimental settings growing linearly with the system size, and a polynomial classic postprocessing complexity.

Other proposals are rooted in concepts from computational complexity theory, such as interactive proof systems. In an interactive proof system, a computer is modelled as the exchange of messages between two parties: a computationally weak, i.e. polynomial verifier, can interact with a more powerful but untrusted prover. In Aharonov et al. (2017), a quantum interactive proof protocol is devised, where the experimentalist is not purely classical but can store and manipulate a constant number - 3, at most - of qubits, exchanging them with an arbitrary quantum system (fig. 6.a). Moreover, it is proven that this relaxed version of quantum interactive proofing (QPIP*) contains all problems in the class BQP. It remains to be found whether or not a completely classical verifier can use quantum interactive proofing to test quantum evolutions efficiently.

1 The Big O notation describes the limiting behaviour of a function using asymptotic analysis. Here, it is used to bound the performance of the procedure in terms of measurement operations as a function of qubit number.

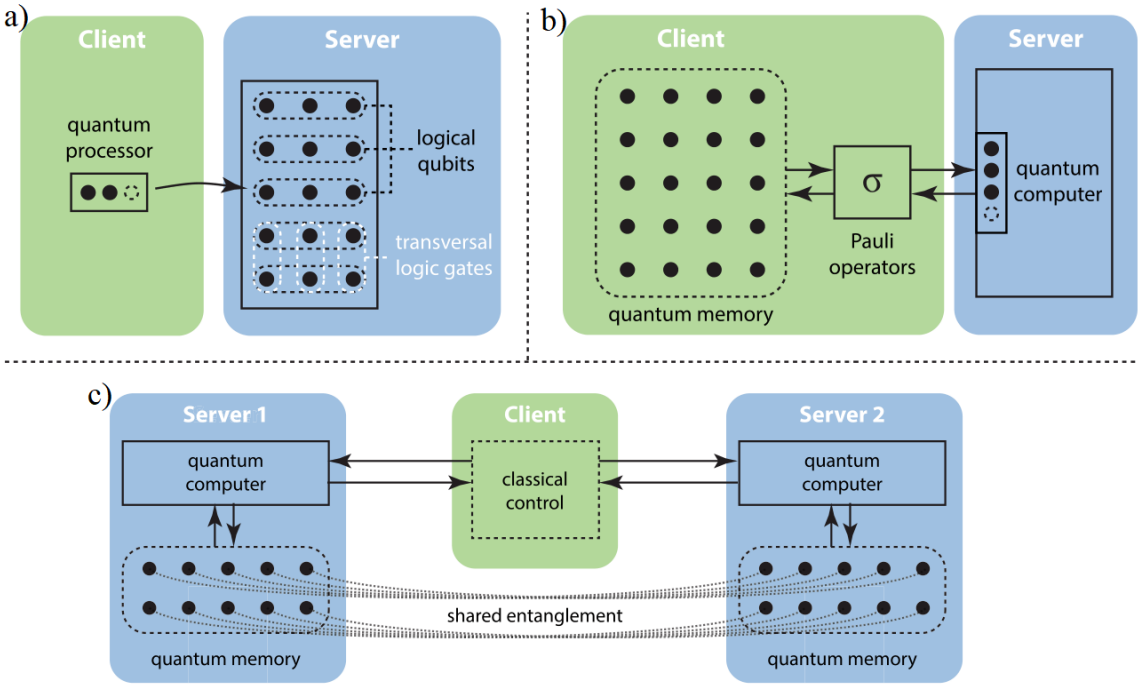


Figure 6.: Different protocols for private quantum computing (Fitzsimons, 2017); a) Quantum interactive proof protocol considered by Aharonov et al. (2017) in which the client has access to a quantum computer capable of performing arbitrary operations on a constant number of qubits; b) Blind quantum computing setting proposed by Childs (2001), where the the client has a large quantum memory together with the ability to perform Pauli operations on qubits and to transmit them to the server; c) BQC protocol where the client communicates through classical channels with two non-communicating servers who share some entangled particles (Broadbent et al., 2009).

Blind quantum computation protocols (Fitzsimons, 2017) emerged from the need to securely delegate quantum computation to an untrusted device while maintaining the privacy of the computation; despite having different motivations, BQC is similar, in intuition, to QPIP. BQC is even more relevant considering that current developments in quantum computation are centralized - access is provided on the cloud, and computations are delegated to these devices through the Internet. Despite its original goal, many BQC protocols also allow for verification of the computation being performed, by embedding hidden tests in the computation. While the ultimate goal in this area is to devise a BQC protocol which could be implemented between a client with no quantum capabilities and a single quantum server, progress has come by relaxing the restrictions to an ideal BQC protocol. Current proven protocols include settings where the client has access to some quantum computational power (fig. 6.b), or settings which allow for a purely classical client and multiple non-communicating quantum servers (fig. 6.b). As with QPIP, it remains an open question whether BQC is possible with a single quantum server, and a classical client.

3.5 SUMMARY

The experimental and conceptual approaches to the problem of validating a quantum computation/digital quantum simulation are multidisciplinary, and in this dissertation they are grouped into an umbrella term: quantum characterization, verification and validation techniques (QCVV).

This section focuses on specific techniques, such as quantum state tomography, which allows for the full characterization of a quantum system by requiring repeated copies of the system which are then measured over different operators. Similarly, quantum process tomography allows for the full characterization of a quantum process, e.g. by preparing different quantum states and creating a mapping of the transformations over the measured states. These techniques are inefficient since they scale exponentially with the number of qubits; they are also particularly sensitive to noise. The reliability of quantum computers is typically verified using randomized benchmarking. In its most general form, this technique efficiently estimates fidelity of a quantum gate independently of the computation being performed. Additional figures of merit may be used together with fidelity to estimate average error rates.

From computational sciences, and computational complexity theory, arise different approaches based on the concept that current models of quantum computation do not generally allow the experimenter direct access to the quantum device; instead, interactions occur through classical or quantum channels. The most prominent of these techniques are quantum interactive proofing and blind quantum computation.

4

EXPERIMENTAL PROCEDURE

Having discussed the requirements for efficiency and reliability of digital quantum simulations in real-world quantum devices, this chapter describes the steps necessary for an actual implementation of a simulation, namely that of the Schrödinger equation for a single particle in one dimension, using 2 or 3 qubits, plus one qubit as an ancilla (i.e. used in auxiliary operations and discarded before the measurement operations).

The goal of the experimental procedure is not to compare the performance of the algorithm or the implementations against the analytical solution of the Schrödinger equation of a single particle, but to evaluate the reliability of the quantum devices in comparison with an ideal version of a quantum simulation (enacted by a classical simulator), in light of the constraints and noise parameters of the devices.

4.1 QUANTUM DEVICES

The IBM Quantum Network (IBM, 2018a) is a cloud-based platform, developed by IBM, which makes it possible to program and remotely interact with a quantum processor housed in an IBM Research lab. IBM's implementation of quantum processors is based on superconducting qubits, which can be programmed according to the quantum circuit model of computation by applying quantum gates, either using its online GUI (IBM, 2018c), writing a quantum program in OpenQASM, a quantum assembly programming language (Cross et al., 2017), or through QISKit, an SDK for writing and executing quantum circuits (IBM, 2018e).

This work makes use of QISKit, a Python-based software development kit for IBM's quantum devices, to create quantum circuits, compile them according to the available gate set and qubit mapping restrictions, execute and extract the results, both on a local, classical simulator, and on remote quantum devices, specifically *IBM Q5 Tenerife* (ibmqx4), which contains a 5-qubit quantum processor and is currently available to the public, and *IBM Q20 Tokyo* (ibmq20) a more recent device with a 20-qubit quantum processor.

Both devices are based on superconducting *transmon* qubits, a type of superconducting charge qubit designed to have reduced sensitivity to charge noise and longer coherence times, which is achieved by increasing the ratio of the Josephson energy to the charging energy;

this type of qubit was first demonstrated experimentally in Koch et al. (2007). A charge qubit is formed by a superconducting island, also known as a Cooper-pair box, coupled by a Josephson junction to a superconducting reservoir. The state of the qubit is determined by the number of Cooper pairs which have tunneled across the junction. In contrast with the charge state of an atomic or molecular ion, the charge states of such an "island" involve a macroscopic number of conduction electrons of the island. The quantum superposition of charge states can be achieved by tuning the gate voltage that controls the chemical potential of the island. Measurement, control and coupling of the transmons is performed by means of microwave resonators with techniques of circuit quantum electrodynamics, also applicable to other superconducting qubits. This coupling is achieved by a putting a capacitor between the qubit and the resonator. The qubits are connected with coplanar waveguide bus resonators, and quantum operations are conducted by applying microwave pulses to the qubits.

IBM's interface allows the user to program a quantum algorithm using a broad set of single-qubit gates, including Pauli and Clifford gates, general unitary and phase shift gates; and multi-qubit gates such controlled-NOT, swap, CCNOT (i.e. a NOT gate with two controls), or Fredkin gates. However, these are compiled into the two types of quantum operations which can be directly implemented physically. One is a unitary operation

$$U(\theta, \phi, \lambda) = R_Z(\phi) R_Y(\theta) R_Z(\lambda) \tag{29}$$

acting on a single qubit, composed of a Bloch sphere qubit rotation on the z-axis, followed by a rotation on the y-axis and another rotation on the z-axis (i.e. a generalized Euler rotation). At the hardware level, these operations are performed by a series of Gaussian derivative and Gaussian flattop pulses with amplitude and angle parameters defined by the expression (29). The other physically implementable operation is a controlled NOT gate (CNOT, or CX) - if the so-called control qubit (denoted as $\bullet$ in quantum circuits) is in basis state $|1\rangle$, the state of the target qubit (denoted as $\oplus$ in quantum circuits) is inverted, i.e. a $NOT \equiv X$ operation is performed; if the control qubit is in basis state $|0\rangle$, the target qubit goes unaltered. This is physically achieved by creating cross-resonance interaction between neighboring qubits that are connected by a superconducting bus resonator. These two operations form an universal basis, which means that any quantum algorithm can be conducted using only single-qubit unitary and CNOT operations.

Besides the restriction regarding the available gates, there are further physical constraints given by the physical architecture of the chip. In fact, CNOT gates can be directly applied only to qubits that are connected.

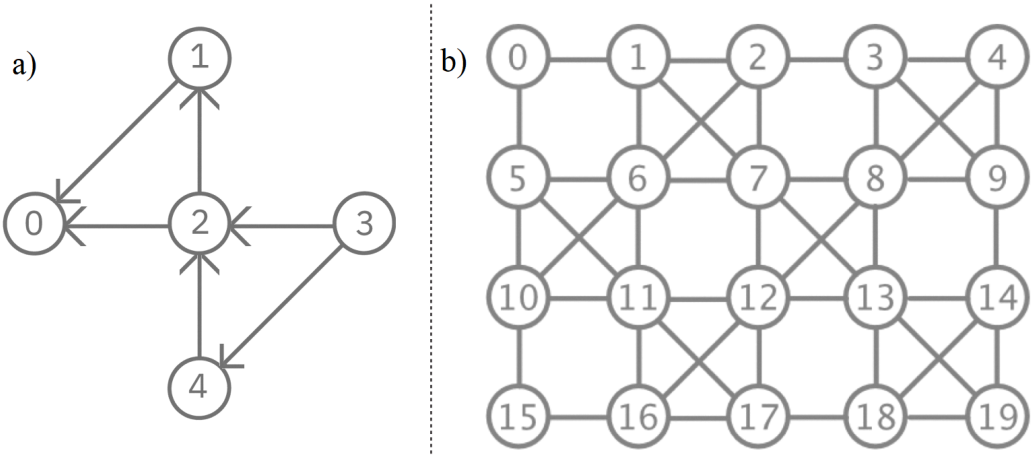


Figure 7.: Quantum device mapping scheme with specifications for qubit interactions; a) for IBM Q5 *Tenerife*; b) for IBM Q20 *Tokyo* (IBM, 2018d).

These restrictions are represented in fig. 7, for both quantum devices used experimentally; qubits are represented by vertices, and an arrow pointing from qubit q_i to qubit q_j indicates that only CNOT with q_i as control, and q_j as target can be applied. In the case of IBM Q20, the edges are bidirectional, i.e. both q_i and q_j can be used as either control or target.

Additionally, each operation performed with quantum gates introduces noise in the system, which results in imperfect computations since there is no error correction technique applied. According to IBM, CNOT gates are less accurate than single-qubit operations by approximately a factor of 10. The error rates are not fixed and depend on the calibration of the device. Each device is typically calibrated twice daily, and from each calibration a list of qubit-specific operation error rates is provided, following the procedure described in Gambetta et al. (2012) and discussed in section 3.3, as well as the associated measurement error rates.

Besides error rates, coherence times impose limits on the amount of operations a given algorithm may experimentally perform to achieve results with reasonable fidelity, since physical gate operations have an associated execution time. The backend information provided by IBM (2018b) for *Tenerife* includes times for the pulses to be performed for each gate; in the case of a single qubit unitary gate, the specific implementation times are 0 ns, 70 ns, and 140 ns for physical gates $U(0,0,\lambda)$, $U(0,\phi,\lambda)$, and $U(\theta,\phi,\lambda)$ respectively, where $\theta,\phi,\lambda \neq 0$. In the case of $U(0,0,\lambda)$, a physical change to the system is avoided with a software-side frame change is enacted, which explains the null execution time. It should be noted that the unitary operation $U(0,0,\lambda)$, also known as *phase shift*, changes the phase of the state $|1\rangle$, which does not, by itself, affect measurement probabilities in the computational basis. For *Tenerife*, the execution times for CNOT gates vary slightly between qubits, so an average value of 410 ns was considered.

One can distinguish between two measures of decoherence:

1. T_1 is the "longitudinal coherence time" (also known as "amplitude damping"), and it measures loss of energy from the system.

2. T_2 is the "transverse coherence time" (also known as "phase damping").

One way to estimate T_1 is to initialize a qubit to the ground state $|0\rangle$ (for , apply an X gate to turn it into $|1\rangle$, and measure it in the computational basis after a time t. The probability of the qubit staying in the $|1\rangle$ state is expected to follow an exponential decay curve e^{-t/T_1}. To experimentally determine T_2, one can initialize a qubit to the ground state $|0\rangle$, apply an Hadamard transform H to change it into $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and wait for a time t before applying another transform H and measuring the qubit on the computational basis. The decay in the probability of obtaining a $|0\rangle$ measurement should follow the expression $\frac{e^{-t/T_2}+1}{2}$.

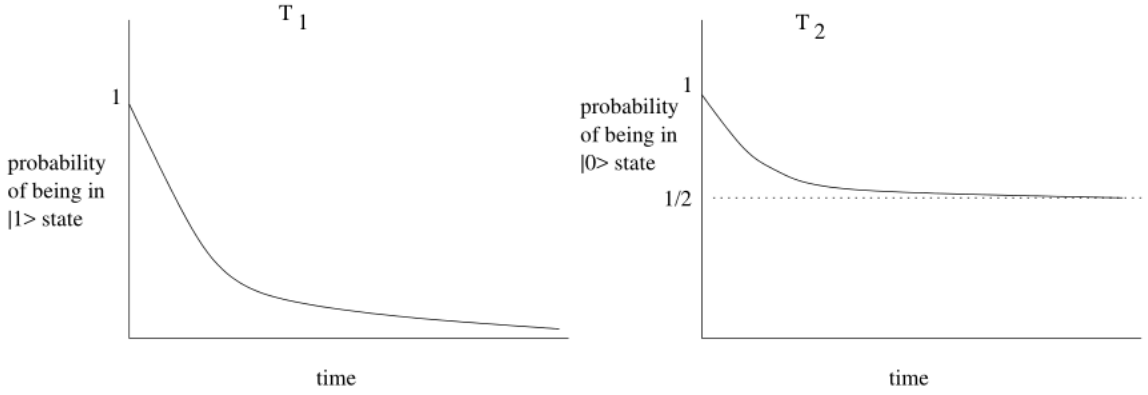


Figure 8.: Expected experimental curves for T_1 and T_2 (Chuang, 2003).

Since IBM provides values for coherence times T_1 and T_2 these times can be compared with an estimated time for the execution. These quantities will be discussed along with the results of the simulations.

4.2 Simulation algorithm

The simulation algorithm performed in this work follows, and expands upon, the outline presented in Coles et al. (2018); a similar algorithm for the simulation of the Schrödinger equation is demonstrated in Benenti and Strini (2008). The general method for simulating the Schrödinger equation of a particle in one dimension was originally suggested by Zalka (1998) and Wiesner (1996).

In the one-dimensional simplification, the motion of the system can be restricted to a region $-d \leq x \leq d$, which can be decomposed into 2^n intervals of length $\Delta x = \frac{2d}{2^n}$, such that it is possible to approximate the wave function over a discrete grid with points x_m, where

$i \in [1, 2^n]$. Each point x_m, determined by the discretization of position, can be encoded in a state $|q_m\rangle$ using n qubits such that $m \in [1, 2^n]$ and $|q_m\rangle = |q\rangle_{n-1} \otimes |q\rangle_{n-2} \otimes ... \otimes |q\rangle_0$. The wave function can then be expressed as:

$$|\psi(x,t)\rangle = \frac{1}{N_F} \sum_{i=1}^{2^n} \psi(x_m, t) \, |q_m\rangle \tag{30}$$

Where $N_F = \sqrt{\sum_{i=0}^{2^n} |\psi(x_m,t)|^2}$ acts as a normalization factor.

The time evolution operator over a step Δt is split into two steps using Trotter decomposition: $e^{-\frac{i}{\hbar}[H_0+V(x)]\Delta t} \approx e^{-\frac{i}{\hbar}H_0\Delta t} \, e^{-\frac{i}{\hbar}V(x)\Delta t}$. This approximation is only exact up to terms of order $(\Delta t)^2$ since the operators H_0, pertaining to momentum, and $V(x)$, pertaining to position, do not commute. Admitting, for simplicity, $\hbar = m = 1$, the wave function for a point in space x_1 after a time step Δt can then be determined as:

$$\psi(x_i, t + \Delta t) = e^{-i\hat{k}^2\Delta t} \, e^{-iV(x_m)\Delta t} \, \psi(x_m, t) \tag{31}$$

Where k is the wavenumber of the particle (see section 1.2 for the theory behind the equation). In this approximation, the time evolution operator consists in alternating applications of the phase shift operator in the position and momentum representations. The Fourier transformation can be used to link these operators by first applying the direct Fourier transform, F, to get into the momentum representation, where $e^{-i\hat{k}^2\Delta t}$ is diagonal. The inverse Fourier transform, F^-1, is then applied to return the system to the position representation, where $e^{-iV(x_m)\Delta t}$ is diagonal. The wave function at a time $l\Delta t$ is obtained by applying l times the operator

$$F^{-1} \, e^{-i\hat{k}^2\Delta t} \, F \, e^{-iV(x_m)\Delta t} \tag{32}$$

From this, the simulation of the Schrödinger equation on a quantum computer can be outlined into the following steps:

1. Prepare the encoded initial state on the quantum computer, by applying the necessary transformations, $\hat{U}_{prep}$, over n qubits representing $N = 2^n$ points;
2. Apply a diagonal phase transformation of the form $e^{-iV(x_m)\Delta t}$;
3. Apply the Quantum Fourier Transform to change the system into momentum representation;
4. Apply a diagonal phase transformation of the form $e^{-i\hat{k}^2\Delta t}$;
5. Apply the inverse QFT to return to the coordinate representation;
6. Repeat steps (2) through (5) until an arbitrary time $l\Delta t$ is reached.

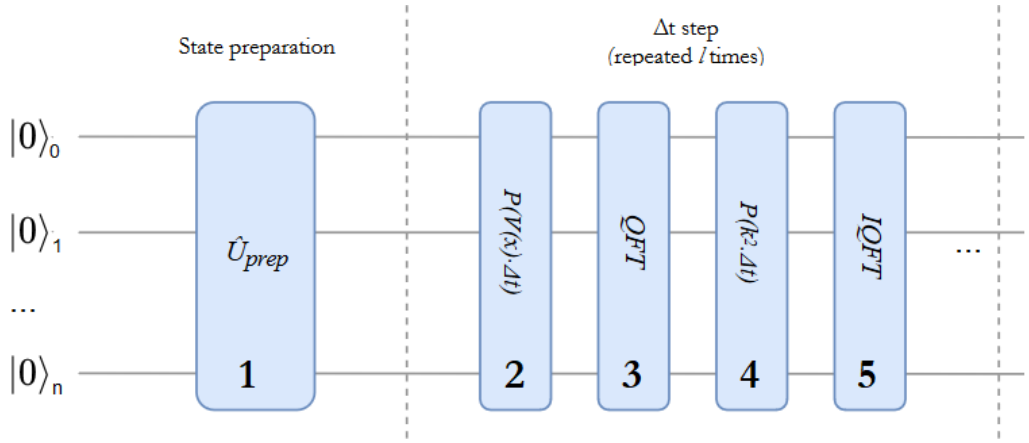


Figure 9.: Circuit schematic for the algorithm over the defined steps.

For all steps of the simulation, qubits obey the same order (*endianness*) in which they are arranged into larger values. For example, position x_3 may be encoded, using a 4 qubit state, as either $|0011\rangle$ or $|1100\rangle$, but step initialization, the Fourier transform and phase transformations all depend on this numerical order and should admit the same convention. In appendix A, a more detailed explanation of the quantum computing programming model is presented.

In this simulation, the chosen initial state $\psi(x,0)$ takes the form of a discrete Π-function. For simplicity, the case of a free particle is considered, such that the potential $V(x) = 0$, making the diagonal phase transformation pertaining to $V(x)$ a trivial step (i.e. the identity operation).

On a quantum computer, one can perform the quantum Fourier transformation as described in (Nielsen and Chuang, 2010, chap. 5), which is the quantum analogue to the discrete Fourier transformation. For this brief theoretical explanation, take into account a particular description of an n-qubit quantum state state $|j\rangle$, written using the binary representation $j = j_1\, j_2 \cdots j_n$ or, more formally, $j = j_1 2^{n-1} + j_2 2^{n-2} + \cdots + j_n 2^0$. Using the notation $0.j_l\, j_{l+1} \cdots j_m$ to represent the binary fraction $j_l/2 + j_{l+1}/4 + \cdots + j_m/2^{m-l+1}$, the quantum Fourier transform can be described by the product representation:

$$|j_1, \cdots, j_n\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0.j_n}\,|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}\,|1\rangle) \cdots (|0\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_n}\,|1\rangle)}{2^{n/2}} \tag{33}$$

This unitary operation can be performed efficiently for a system with n qubits, with complexity $O(n^2)$ in Hadamard gates (H) and controlled phase shift gates, i.e. a phase shift operation $R_m \equiv P(2\pi i/2^m)$ on target qubit. On the circuit model of computation, the algorithm for the quantum Fourier transform can be represented graphically (fig. 10).

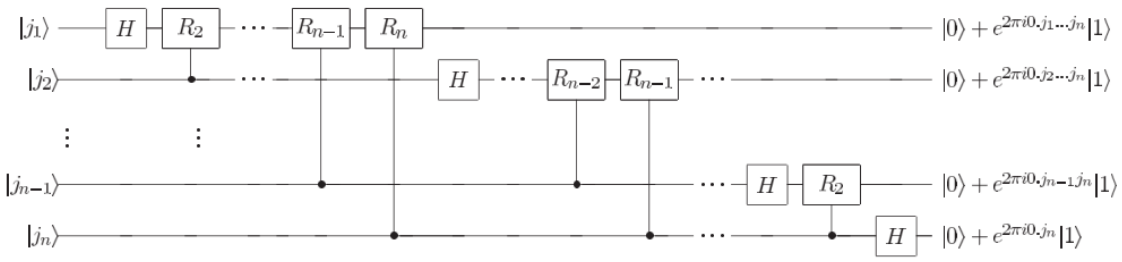


Figure 10.: Efficient circuit for the quantum Fourier transform. Not shown are the swap gates at the end of the circuit which reverse the order of the qubits, or normalization factors of $1/\sqrt{2}$ in the output (Nielsen and Chuang, 2010).

The momentum expression adopted in this simulation takes into consideration how each position is encoded into a binary string, which is itself a description of a quantum state. For the 2 and 3-qubit simulation:

$$\hat{k} = -\sqrt{\frac{1}{2^{2n-3}}\frac{\phi}{\Delta t}}\left(1 + \sum_{j=1}^{n} 2^{n-j}\hat{Z}_j\right) \tag{34}$$

Where ϕ is the characteristic phase shift experienced by the state on time step Δt. As such, in this encoding, the phase-shift operation $e^{-i\hat{k}^2\Delta t}$ contains one and two commuting qubit operations, obtained by expanding the phase transformation from equation (34):

$$\exp\left(-i\hat{k}^2\Delta t\right) = \exp\left(\frac{i\phi}{2^{2n-3}}(1 + \sum_{j=1}^{n} 2^{n-j}\hat{Z}_j)^2\right) \tag{35}$$

This particular operation employs an extra qubit as an ancilla for phase transformation operations, a technique shown, as an example, in fig. 11.

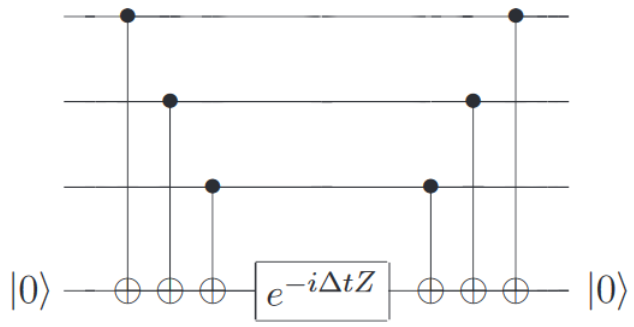


Figure 11.: Quantum Circuit for simulating the Hamiltonian $\hat{H} = \hat{Z}_1 \otimes \hat{Z}_2 \otimes \hat{Z}_3$ for time Δt, using one ancilla qubit (Nielsen and Chuang, 2010).

After the realization of the phase transformation, the inverse quantum Fourier transform is then performed so the system returns to coordinate representation. This is achieved by

applying the inverse unitary operator of the direct QFT (equivalent to applying the inverse gates by inverse order).

The system may then be measured on the computational basis over each qubit, or characterized with quantum state tomography techniques.

4.2.1 *2-qubit implementation*

The initial wave function $\psi(x,0)$ needs to be encoded using $n = 2$ qubits, representing a 4-point grid. The Π-function is discretized as $\psi\{x_0, x_1, x_2, x_3\} = \{0, 1, 1, 0\}$, and it can be encoded, up to a normalization constant, as the superposition state $|q_1\, q_2\rangle = |01\rangle + |10\rangle$.

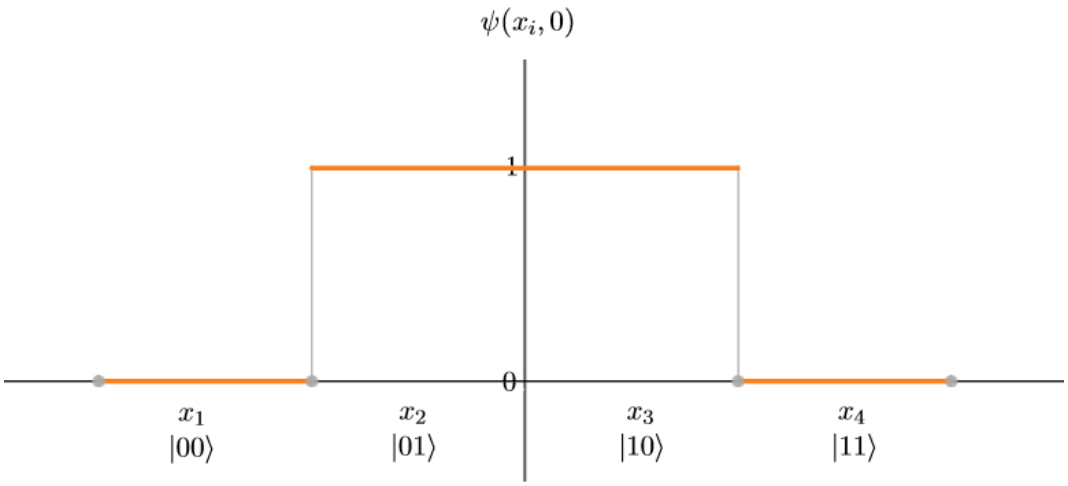


Figure 12.: Graphic representation of the amplitudes, up to a normalization constant, of the Π-function as a superposition of 2-qubit states.

The state can be prepared directly using gate-based operators on a quantum computer as:

$$\hat{U}_{prep} = \hat{X}_1 \,.\, \hat{CX}_{12} \,.\, \hat{X}_1 \,.\, \hat{H}_1 \tag{36}$$

Where $\hat{X}_1$ and $\hat{H}_1$ are the operators corresponding to the Pauli-X and Hadamard gate, respectively, each acting on qubit 1, and $\hat{CX}_{12}$ is the conditional-NOT gate, acting on qubit 2 with qubit 1 as control (the algebraic description of these operations is presented in appendix A). The quantum circuit gates for initial state preparation can also be represented schematically:

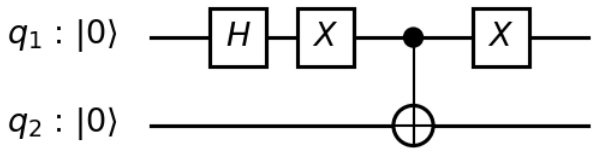


Figure 13.: Circuit schematic for the operations composing the state preparation, $\hat{U}_{prep}$.

The quantum Fourier transform is then applied, followed by an X gate performed on the most significant qubit to center the momentum representation on zero (the quantum equivalent to centering the frequency representation around zero for a classical discrete Fourier transform). Algebraically, $Q\hat{F}T = \hat{X}_2 \,.\, \hat{H}_2 \,.\, C\hat{P}_{(\frac{\pi}{2})12} \,.\, \hat{H}_1$.

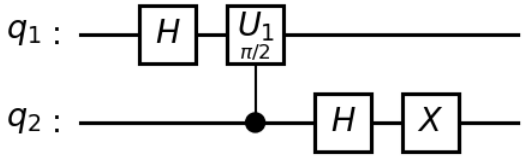


Figure 14.: Circuit schematic for the operations composing the 2-qubit centered quantum Fourier transform, $Q\hat{F}T$. The transform is followed by an X gate to center the momentum representation.

The SWAP operation is avoided by simply changing qubit references and admitting the inverse order. By expanding $\hat{k}$ on equation (35) for $n = 2$, ignoring global phase:

$$e^{-i\hat{k}^2\Delta t} = e^{i\phi(2\hat{Z}_1+\hat{Z}_2+2\hat{Z}_1\otimes\hat{Z}_2)} = e^{i2\phi\hat{Z}_1}\, e^{i\phi\hat{Z}_2}\, e^{i2\phi(\hat{Z}_1\otimes\hat{Z}_2)} \tag{37}$$

Where $e^{i2\phi(\hat{Z}_1\otimes\hat{Z}_2)}$ is applied following the technique demonstrated in fig. 11.

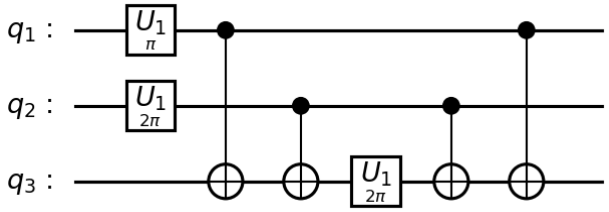


Figure 15.: Circuit schematic of the phase transformation operations $e^{-i\hat{k}^2\Delta t}$ for the 2-qubit simulation, with $\phi = \pi$, using one ancilla qubit. The order of the qubits is inverted due to the previously applied quantum Fourier transform.

After performing the phase transformation, the system is returned to the position representation by applying the inverse transformation to $Q\hat{F}T$. This completes one iteration of the simulation over the time step Δt, and each qubit may then be measured.

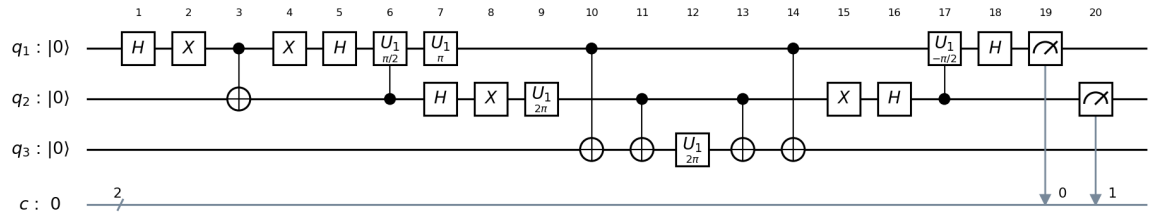


Figure 16.: Circuit representation of the 2-qubit free particle simulation for $\phi = \pi$- The third qubit is used as an ancilla, and is discarded (i.e. not measured) at the end of the simulation. The fourth line represents a 2-bit classical register containing the results of the measurement for each qubit.

4.2.2 *3-qubit implementation*

In the case of the 3-qubit simulation, the initial state is encoded in $n^3 = 8$ discrete intervals as $\psi\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\} = \{0, 0, 1, 1, 1, 1, 0, 0\}$, and represented as the superposition of 3-qubit states, up to a normalization constant, $|q_1, q_2, q_3\rangle = |010\rangle + |011\rangle + |100\rangle + |101\rangle$. In practice, the discrete representation of the Π-function consists in simply doubling the amount of intervals in fig. 12:

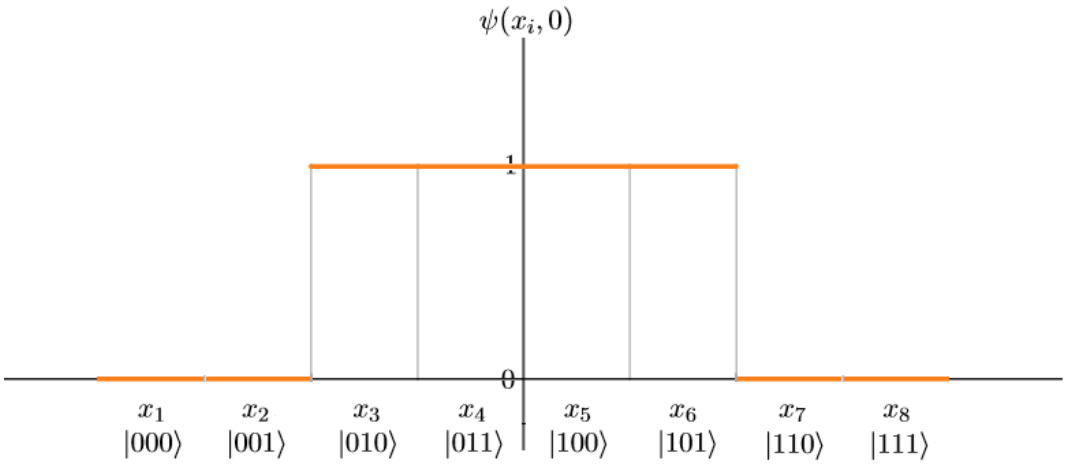


Figure 17.: Graphic representation of the amplitudes, up to a normalization constant, of the Π-function as a superposition of 3-qubit states.

The initial state was prepared from register $|000\rangle$ using the algorithm proposed in Shende et al. (2006). The algorithm is already implemented in QISKit's libraries and, as mentioned previously, is based on taking the inverse problem, i.e. designing a circuit for obtaining the n-qubit state $|q_1 \cdots q_n\rangle = |0 \cdots 0\rangle$, and implementing the inverse operation, which is trivial using quantum gates. This is achieved by disentangling the least significant qubit into a separable

product state $|q_1 \cdots q_{n-1}\rangle \otimes |0\rangle$, and recursively applying the algorithm to the $(n-1)$-qubit state. The algorithm uses $2^{n-1} - 2n$ CNOT gates, resulting in 10 CNOT gates for the 3-qubit state.

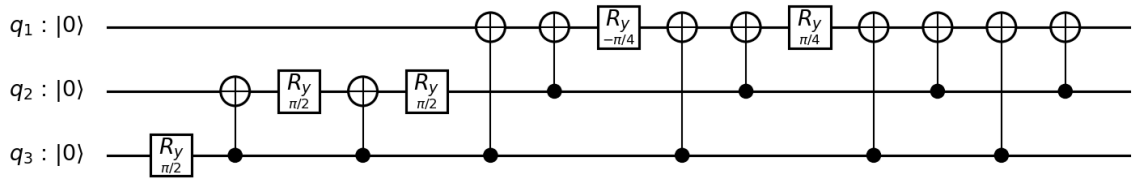


Figure 18.: Circuit schematic for the operations composing the initial state preparation for the 3-qubit simulation.

The centered quantum Fourier transform was then applied followed, which for a 3-qubit system corresponds to $Q\hat{F}T = \hat{X}_3.\hat{H}_3.C\hat{P}_{(\frac{\pi}{2})23}.\hat{H}_2.C\hat{P}_{(\frac{\pi}{4})13}.C\hat{P}_{(\frac{\pi}{2})12}.\hat{H}_1$.

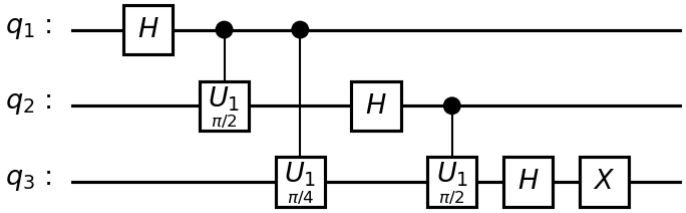


Figure 19.: Circuit schematic for the operations composing the 3-qubit centered quantum Fourier transform, $Q\hat{F}T$.

The phase transformation operation was obtained by expanding equation (35) for n=3:

$$\exp\{-i\hat{k}^2\Delta t\} = \exp\{i\phi\left(\hat{Z}_1 + \frac{1}{2}\hat{Z}_2 + \frac{1}{4}\hat{Z}_3 + 2\hat{Z}_1 \otimes \hat{Z}_2 + \hat{Z}_1 \otimes \hat{Z}_3 + \frac{1}{2}\hat{Z}_2 \otimes \hat{Z}_3\right)\} \tag{38}$$

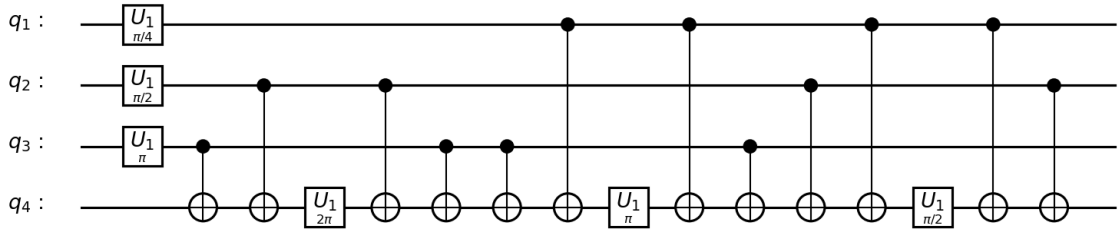


Figure 20.: Circuit schematic of the phase transformation operations $e^{-i\hat{k}^2\Delta t}$ for the 3-qubit simulation, with $\phi = \pi$, using one ancilla qubit. The order of the qubits is inverted due to the previously applied quantum Fourier transform.

After performing the phase transformation, the system is returned to the position representation by applying the inverse transformation to $Q\hat{F}T$, completing one iteration of the simulation over the time step Δt. Each qubit may then be individually measured.

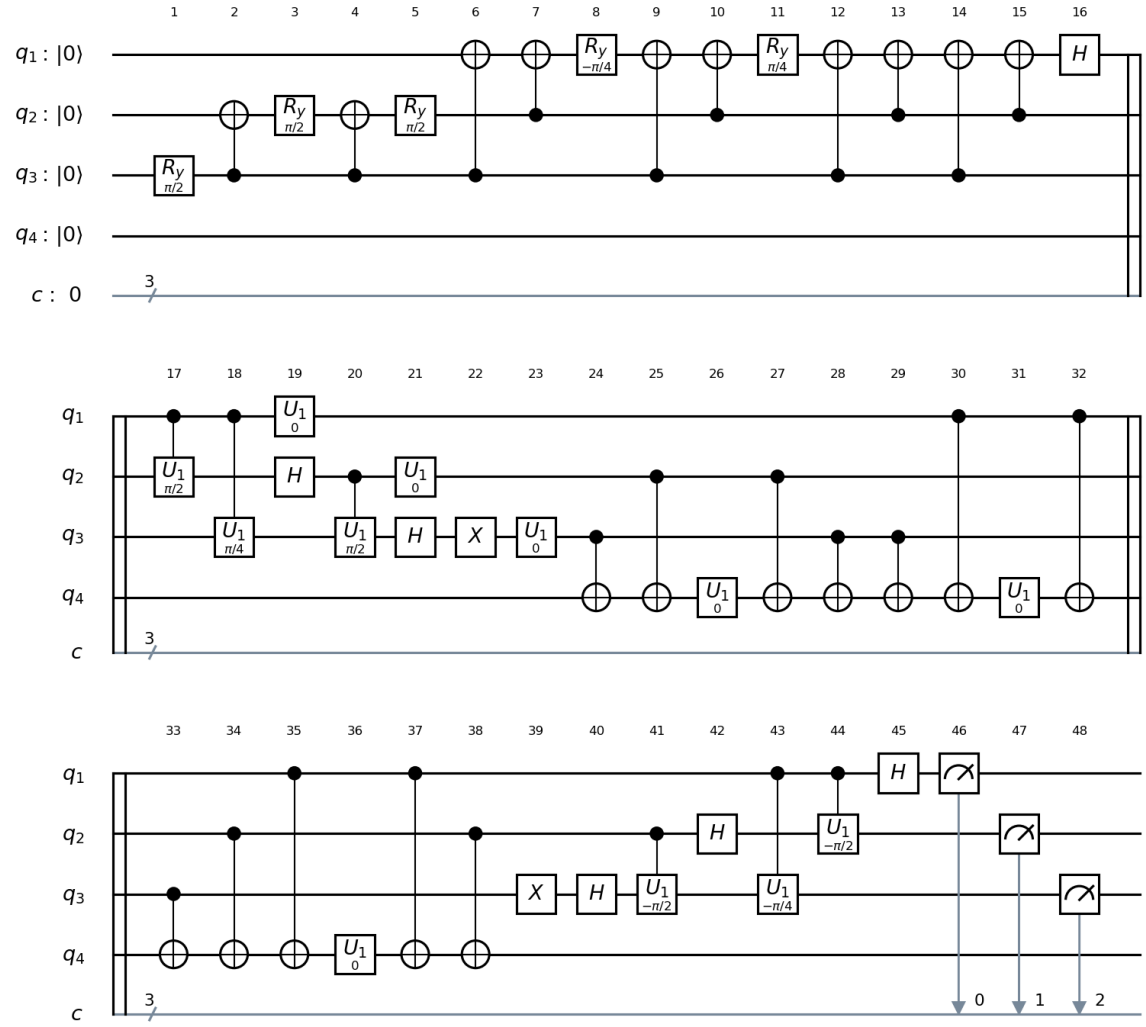


Figure 21.: Circuit representation of the 3-qubit free particle simulation for $\phi = 0$- The fourth qubit is used as an ancilla, and is discarded (i.e. not measured) at the end of the simulation. The last line represents a 3-bit classical register containing the results of the measurement for each qubit.

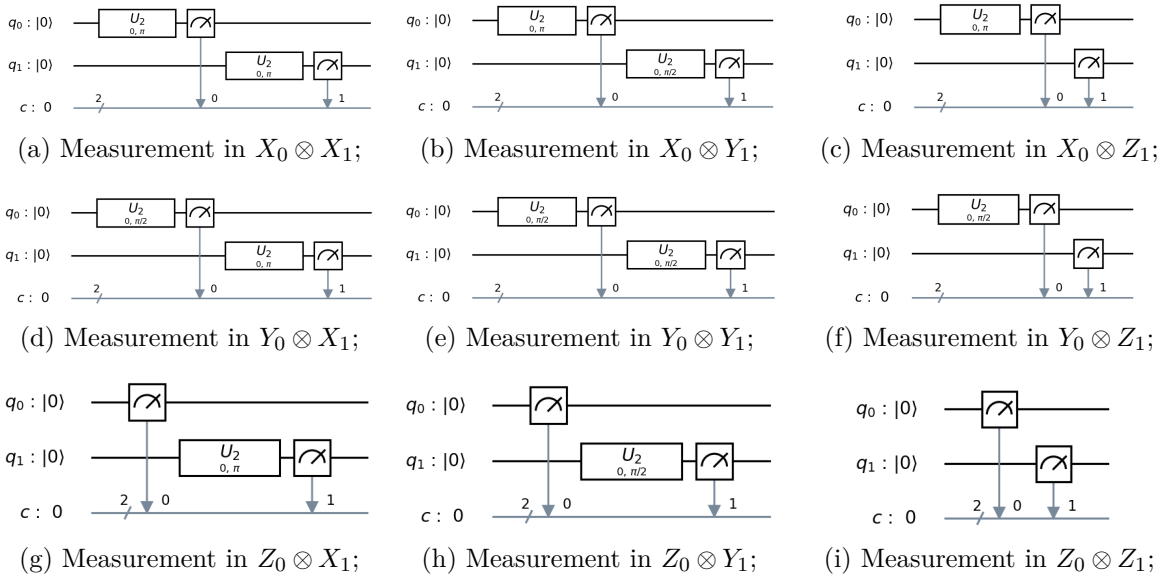
4.3 QUANTUM STATE TOMOGRAPHY

Measuring each qubit on the computational basis after the simulation, for a large number of repetitions, allows one to estimate the probability distribution of each measurement, which corresponds to the squared modulus of each amplitude of the simulated wave function. With

this information alone it is not possible to quantify the fidelity of the simulation, or characterize the quantum state being simulated, since a measurement collapses the state of the system into an eigenstate of the associated measurement operator.

The general principle behind quantum state tomography is that by repeatedly performing different measurements, forming a basis, of quantum systems described by identical density matrices, frequency counts can be used to infer probabilities, and these probabilities are combined to determine a density matrix which fits the best with the observations; the underlying theory is detailed in section 3.1. The specific state tomography algorithm implemented in the experimental procedures computes the maximum-likelihood density matrix ρ describing a mixed quantum state given a set of measurement outcomes in a complete orthonormal operator basis. The algorithm for processing the measurement data is implemented in QISKit's tool library.

For a n-qubit system, 3^n different measurements have to be performed, each with an associated quantum circuit.



(a) Measurement in $X_0 \otimes X_1$; (b) Measurement in $X_0 \otimes Y_1$; (c) Measurement in $X_0 \otimes Z_1$;

(d) Measurement in $Y_0 \otimes X_1$; (e) Measurement in $Y_0 \otimes Y_1$; (f) Measurement in $Y_0 \otimes Z_1$;

(g) Measurement in $Z_0 \otimes X_1$; (h) Measurement in $Z_0 \otimes Y_1$; (i) Measurement in $Z_0 \otimes Z_1$;

Figure 22.: For the state tomography procedure $3^2 = 9$ measurement operators were created in QISKit for the 2-qubit simulation. Each was appended to the end of the simulation algorithm, forming 9 distinct quantum circuits.

For the 2-qubit simulation, 9 quantum circuits were created (fig. 22), and each one was executed 100 times. The 3-qubit simulation was measured using 27 distinct quantum circuits, also with 100 executions each. The data obtained from the measurements for each operator can be obtained by calling the function `tomography_data` on the results of the execution, along with the simulation circuit and set of measurement operators. A density matrix ρ can then be built by calling the function `fit_tomography_data` on the state tomography

data. This particular method (Smolin et al., 2012) constrains positivity by setting negative eigenvalues to zero and re-scaling the positive eigenvalues.

The reconstructed matrix ρ can be compared with the density matrix of the ideal state by determining the state fidelity (definition 3.1), and its purity can be measured (definition 3.2). Since the desired final state of the simulation is analytically known for both the 2 and 3-qubit simulations, the results of the tomographic characterization may be used to evaluate the performance of the state tomography technique itself.

4.4 PROCEDURE

The experimental procedure can be separated into two parts. First, for the 2-qubit simulation, three separate simulations of the Schrödinger equation for the wave function of a single particle in 1 spatial dimension were experimentally implemented, each with a different value of ϕ, the characteristic phase shift (detailed in section 4.2) experienced over one iteration of the simulation; in the 3-qubit case, the simulation was implemented for a single value of ϕ. Then, each implementation was executed:

1. In the classical simulator of a quantum system provided by IBM's software development kit, representing an ideal universal quantum simulator without noise or decoherence (this is possible due to the low number of qubits of the simulation);

2. Using the software's integrated compiler for quantum circuits, in two specific quantum devices (see section 4.1), IBM Q5 *Tenerife* and IBM Q 20 *Tokyo*;

3. Using the compiling procedure recently proposed and provided by Zulehner and Wille (2018), in the same two quantum devices. This technique will be referred to as an 'alternative compiler' for the purposes of discussing the results.

It should be noted that the classical simulator not only performs numerical simulation of quantum algorithms, but also emulates the *randomness* inherent to quantum state measurements. Each specific implementation was executed 1000 times, in the case of the 2-qubit simulation, and 2000 times, in the case of the 3-qubit simulation. After each execution, the qubits were individually measured on the computational basis. The probability distribution of the measurements should follow the discrete wave function amplitudes of the particle's Schrödinger equation after a time Δt.

For the results of each implementation, a frequency of correct measurements can be estimated; here, *correct measurements* are taken as measurement results with an expected non-zero probability after an ideal simulation. For example, for the quantum state represented in fig. 12, the frequency of correct measurements would be the frequency of measurements that

resulted returned either $|01\rangle$ or $|10\rangle$. This figure can be compared with a rough estimation of the probability that the simulation suffered from no errors during the execution.

For each device, IBM provides a list of qubit-specific error rates for single-qubit gates, CNOT gates and the measurement operators. These are estimated from randomized benchmarking (detailed in section 3.3). The error rates are updated each time the device is calibrated, which occurs daily. The discussion of the experimental results will take a naive approach for estimation of simulation error probabilities. Admitting that operations acting on distinct sets of qubits can occur simultaneously, a quantum circuit has an associated *circuit depth*, namely the number of time steps required for the simulation. Each time step is associated with a circuit *layer* which contains only gates acting on distinct sets of qubits. A circuit layer is "greedy" in the sense that it contains the largest possible number of operations fitting one circuit layer.

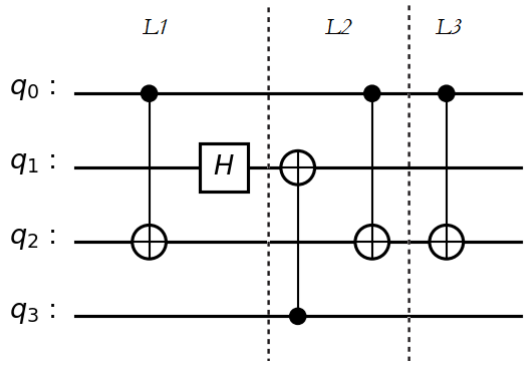


Figure 23.: A quantum circuit with three distinct circuit layers.

For IBM devices, single-qubit operations and CNOT operations have distinct execution times. The error rates are qubit dependent, but CNOT gates have an average error rate that is one order of magnitude larger than single-qubit operations. As such, for each experimental quantum circuit implementation the number of layers containing CNOT gates and the number of layers containing only single qubit operations can be distinctly determined. The execution time of the circuit can be estimated as:

$$T_E = N_{CNOT}\, T_{CNOT} + N_U\, T_U \tag{39}$$

Where N_{CNOT} and N_U are the number of layers containing at least one CNOT gate, and the number of layers containing only single-qubit gates, respectively; T_{CNOT} and T_U are the average execution times for layers containing CNOT and for layers containing only single-qubit gates. For the analysis, execution times $T_U = 70ns$ and $T_{CNOT} = 410ns$ were considered; these were estimated from the backend information provided by IBM in IBM (2018b), which includes times for the pulses to be performed for each gate. Since gate execution times for *Tokyo* were not provided by IBM at the time of writing, the same values were admitted for

this device. The total execution times can be compared with the provided coherence times T_1 and T_2 in both devices. The manufacturer proves a list of qubit-specific values for T_1 and T_2 in each device; for simplification, the average value was considered.

For each layer a distinct error rate can be considered: er_{CNOT} for layers containing at least one CNOT gate, and er_U for layers with only single-qubit operations. A figure of merit P_S can be used as a simplistic estimation for the probability of a simulation without the occurrence of errors (i.e. an "ideal" simulation):

$$P_S = (1 - er_{CNOT})^{N_{CNOT}} . (1 - er_U)^{N_U} . (1 - er_M) \tag{40}$$

Where er_M is average the error rate for measurement operators. This is a very simplified approach to the study of probabilities of simulation errors, since it does not take into account the probability of an error changing the system between two correct states, or errors occurring more than once; as such it is expected that this figure will suggest worse results than what is actually detected. However, one can still study how this quantity correlates to the frequency of correct measurements.

For the second part of the experimental procedure, each n-qubit implementation was repeated, but this time each was measured on a different Pauli basis, for a total of 3^n specific quantum circuits, forming a tomographically complete set of measurements for each implementation. The state was reconstructed from the maximum likelihood technique (detailed in section 3.1) proposed by Smolin et al. (2012). From the reconstructed density matrix, one can compare it with the ideal final state, which can be trivially calculated analytically, by estimating quantum state fidelity (definition 3.1) to the ideal final state, and quantum state purity (definition 3.2), since it is expected that the final state is pure. The tomography procedures were also executed in the classical simulator, which allows to check the performance of the procedure in the absence of noise. Using these quantities, one can infer how much of the fidelity was lost due to noise and decoherence. For example, a low state fidelity with a high gate purity would indicate an inaccurate implementation of the algorithm, since the resulting low fidelity could not be explained by noise.

5

RESULTS AND DISCUSSION

The results of the procedure detailed in section 4.4 were compared with the expected final state, by solving for equation:

$$\psi(x_i, t + \Delta t) = e^{-i\hat{k}^2 \Delta t}\, e^{-iV(x_m)\Delta t}\, \psi(x_m, t) \tag{41}$$

Where $\hat{k}$ is obtained from equation (34) and $V\{x_m\} = 0$. Considering the simulation is done for a single time step Δt, it is trivial to solve for the final state as a function of the characteristic phase shift ϕ. Starting with the wave function encoding $|\psi(x_m, 0)\rangle = |01\rangle + |10\rangle$, up to a normalization constant, one can expect, after Δt, the state:

$$\begin{aligned} &\phi = 0: \ |\psi(x_m, \Delta t)\rangle = |01\rangle + |10\rangle \\ &\phi = \pi/2: \ |\psi(x_m, \Delta t)\rangle = |00\rangle + |01\rangle + |10\rangle + |11\rangle \\ &\phi = \pi: \ |\psi(x_m, \Delta t)\rangle = |00\rangle + |11\rangle \end{aligned} \tag{42}$$

Or, for the 3-qubit simulation, which was implemented only for $\phi = 0$:

$$\phi = 0: \ |\psi(x_m, 0)\rangle = |\psi(x_m, \Delta t)\rangle = |010\rangle + |011\rangle + |100\rangle + |101\rangle \tag{43}$$

The estimated coherence times and error rates for both devices were obtained from the average of the qubit-specific coherence times and error rates provided by the SDK.

Device	$T_1(\mu s)$	$T_2(\mu s)$	$E_U(10^{-3})$	$E_{CNOT}(10^{-2})$	$E_M(10^{-2})$
ibmqx4 (Tenerife)	49.8	24.8	1.72	4.54	4.88
ibmq20 (Tokyo)	84.6	55.0	1.45	3.05	7.57

Table 1.: Average device parameters for coherence times T_1 and T_2, and average single-qubit (E_U), CNOT (E_{CNOT}) and measurement (E_M) error rates.

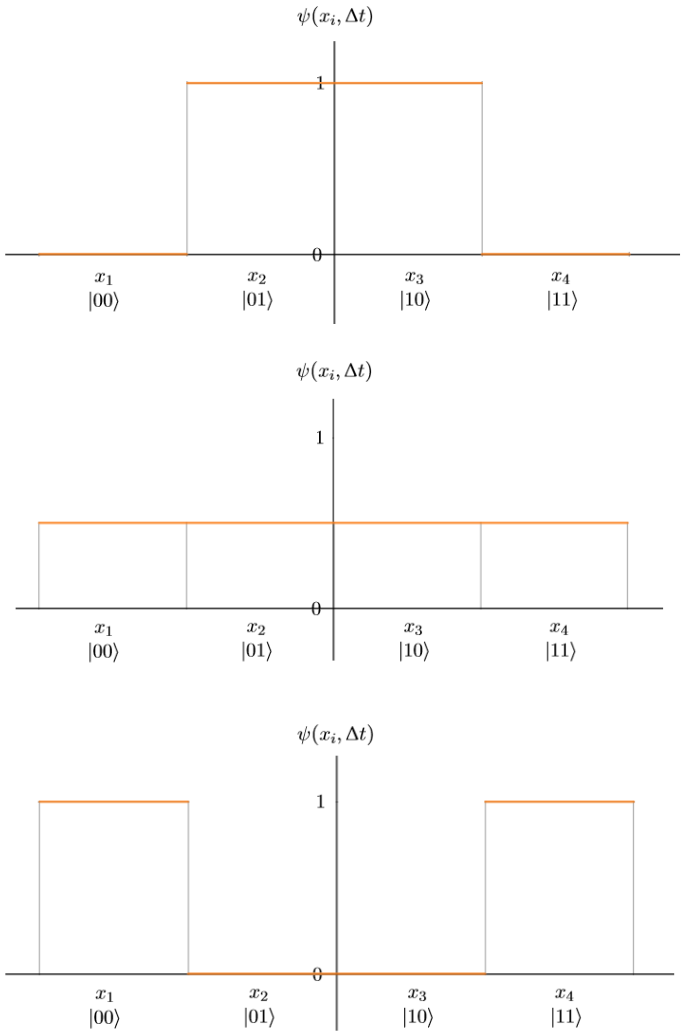


Figure 24.: Graphic representation of the amplitudes, up to a normalization constant, of the probability distribution of the desired final wave function as a superposition of 2-qubit states, for $\phi = 0$ (top); $\phi = \pi/2$ (middle); $\phi = \pi$ (bottom).

5.1 2-qubit simulation

Before executing each simulation, the compiled quantum circuits were analyzed to determine the number of layers for each execution, N_U and N_{CNOT}. These quantities allow for estimation of execution time, given that each single-qubit unitary operation takes an average of $70ns$, and a CNOT operation takes of $500ns$

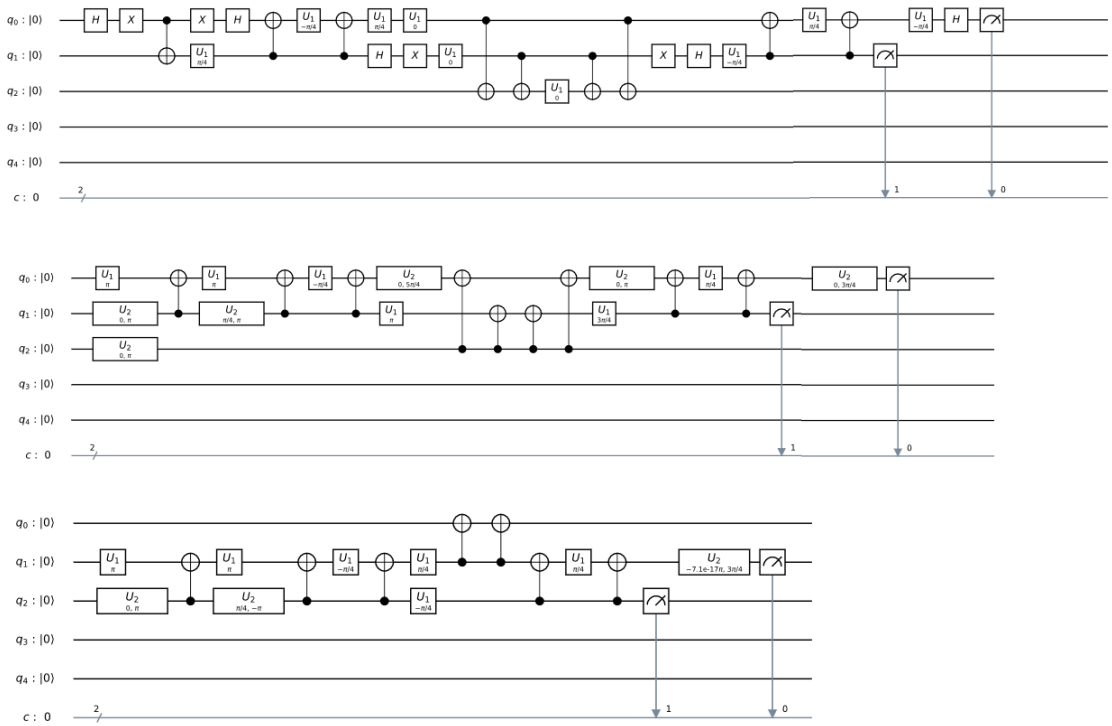
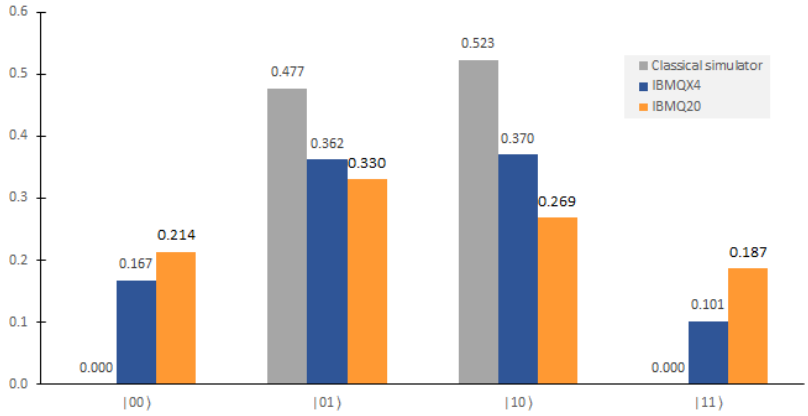


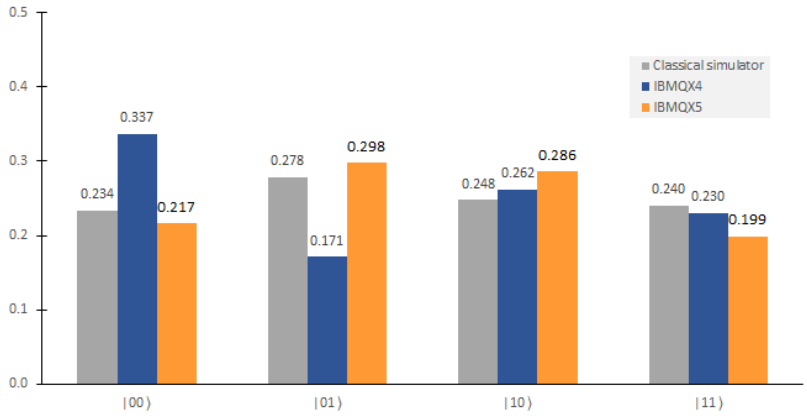
Figure 25.: Quantum circuit of the 2-qubit quantum simulation for $\phi = 0$: in an ideal simulator (top); compiled for *ibmqx4*, using QISKit's compiler (middle); compiled for *ibmqx4*, using the alternative compiler by Zulehner and Wille (2018) (bottom).

A comparison is shown, as an example, in fig. 25 for the 2-qubit simulation on an ideal simulator and on *ibmqx4*, and $\phi = 0$. The compilers 'condense' consecutive single-qubit gates as one unitary gate, and further optimizations are performed such that the circuit complies with the physical gate set and mapping constraints of the device. In this particular example, the circuit compilation provided by QISKit resulted in a circuit with 7 single-qubit layers and 9 CNOT layers, while the compiler provided by Zulehner et al. resulted in a circuit with 6 single-qubit layers and 7 CNOT layers, a marginal improvement.

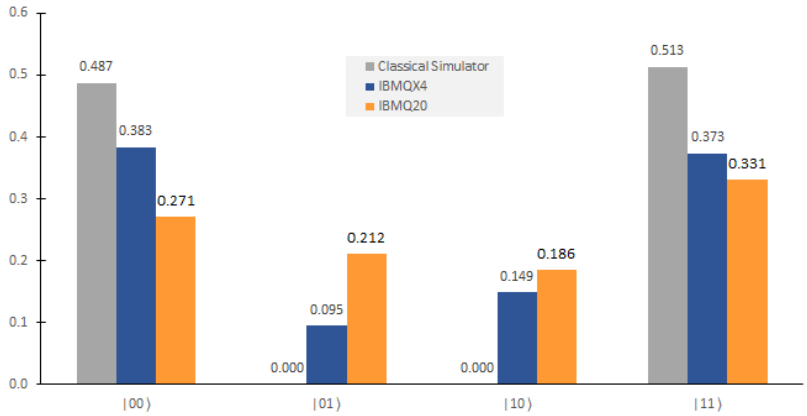
The 2-qubit simulation was executed for $\phi = \{0, \pi/2, \pi\}$ and each qubit measured in the computational basis.



(a) $\phi = 0$



(b) $\phi = \pi/2$



(c) $\phi = \pi$

Figure 26.: Probability distribution of the 2-qubit simulation for 3 distinct ϕ, for the classical simulator, ibmqx4, and ibmq20, using QISKit's standard compiler.

For $\phi = 0$, *ibmqx4* obtained a frequency of correct measurements $f_C = 0.73$, while in the case of *ibmq20*, $f_C = 0.60$. For $\phi = \pi$, *ibmqx4* returned correct states with frequency $f_C = 0.76$, while for *ibmq20*, $f_C = 0.60$. For $\phi = \pi/2$, the desired final state is a uniform superposition of all possible states. Here *ibmqx4* returned the biggest deviations from the expected frequencies, namely for the states $|00\rangle$ and $|01\rangle$.

The implementations described above were repeated, but instead of performing standard measurements, the quantum state tomography procedure described in 4.3 was applied. This allows to characterize the density matrix of the final state on each implementation, from which the state fidelity (definition 3.1) and state purity (definition 3.2) can be estimated. The tables of the experimental data for each chosen parameter of ϕ follow.

$\phi = 0$	Classical	ibmqx4 (Tenerife)		ibmq20 (Tokyo)	
Results	Simulator	QISKit	Alternative	QISKit	Alternative
$\|00\rangle$	0.000	0.167	0.123	0.214	0.388
$\|01\rangle$	0.477	0.362	0.366	0.330	0.257
$\|10\rangle$	0.523	0.370	0.456	0.269	0.129
$\|11\rangle$	0.000	0.101	0.055	0.187	0.226
Fidelity	0.96	0.64	0.59	0.45	0.32
Purity	0.93	0.49	0.43	0.34	0.33
N_U	15	7	6	7	17
N_{CNOT}	9	9	7	14	15
T_E		4.2	3.3	6.2	7.3
P_S		0.62	0.68	0.59	0.57

Table 2.: Experimental results of the 2-qubit simulation for $\phi = 0$, for the classical simulator and quantum devices ibmqx4 and ibmq20.

$\phi = \pi/2$	Classical	ibmqx4 (Tenerife)		ibmq20 (Tokyo)	
Results	Simulator	QISKit	Alternative	QISKit	Alternative
$\|00\rangle$	0.234	0.337	0.227	0.217	0.394
$\|01\rangle$	0.278	0.171	0.247	0.298	0.237
$\|10\rangle$	0.248	0.262	0.305	0.286	0.125
$\|11\rangle$	0.240	0.230	0.221	0.199	0.244
Fidelity	0.99	0.59	0.61	0.39	0.32
Purity	0.98	0.46	0.49	0.36	0.35
N_U	15	8	8	8	17
N_{CNOT}	9	9	9	15	15
T_E		4.3	4.3	6.7	7.3
P_S		0.62	0.62	0.57	0.57

Table 3.: Experimental results of the 2-qubit simulation for $\phi = \pi/2$, for the classical simulator and quantum devices ibmqx4 and ibmq20.

$\phi = \pi$	Classical	ibmqx4 (Tenerife)		ibmq20 (Tokyo)	
Results	Simulator	QISKit	Alternative	QISKit	Alternative
$\|00\rangle$	0.487	0.383	0.342	0.271	0.438
$\|01\rangle$	0.000	0.095	0.183	0.212	0.222
$\|10\rangle$	0.000	0.149	0.146	0.186	0.136
$\|11\rangle$	0.513	0.373	0.329	0.331	0.204
Fidelity	0.98	0.62	0.57	0.45	0.43
Purity	0.96	0.47	0.49	0.39	0.41
N_U	15	8	10	8	16
N_{CNOT}	9	9	9	17	15
T_E		4.3	4.4	7.5	7.3
P_S		0.62	0.62	0.54	0.57

Table 4.: Experimental results of the 2-qubit simulation for $\phi = \pi$, for the classical simulator and quantum devices ibmqx4 and ibmq20.

5.2 3-QUBIT SIMULATION

For the 3-qubit simulations, the same procedure was adopted. Each circuit is previously compiled so the number of layers can be determined, such that execution time and probability of an ideal simulation can be estimated. However only the implementation for $\phi = 0$ is analyzed.

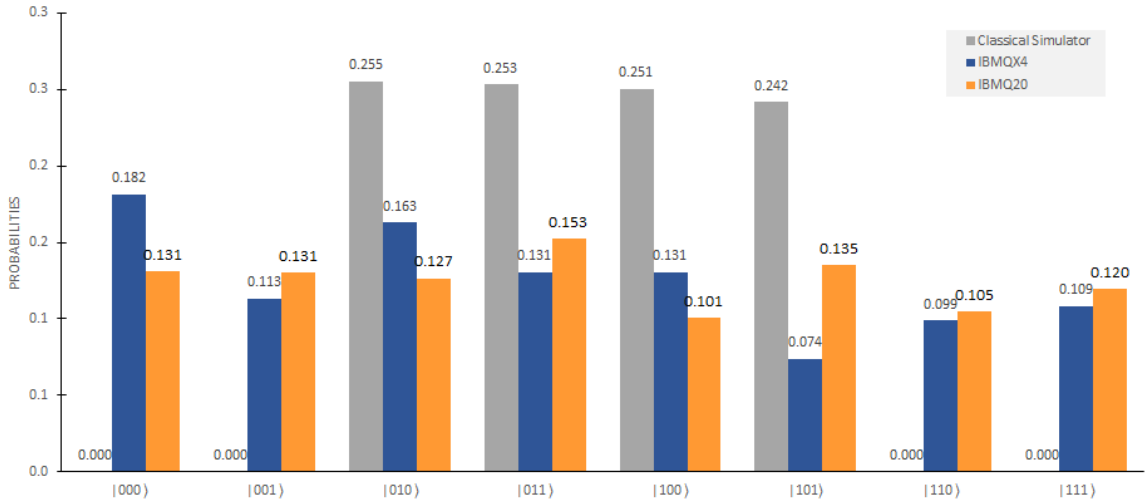


Figure 27.: Probability distribution of the 3-qubit simulation for $\phi = 0$, for the classical simulator, ibmqx4, and ibmq20.

The probability distributions for both *ibmqx4* and *ibmq20* seem to have a noisy superposition of all possible states without any resemblance to the desired final state, as the comparison with the results on a classical simulator shows. This correlates with the relatively large estimated execution times and low probability of ideal simulations, as it can be seen in table 5.

$\phi = 0$	Classical	ibmqx4 (Tenerife)		ibmq20 (Tokyo)	
Results	Simulator	QISKit	Alternative	QISKit	Alternative
$\|000\rangle$	0.000	0.182	0.181	0.131	0.148
$\|001\rangle$	0.000	0.113	0.127	0.131	0.1135
$\|010\rangle$	0.255	0.163	0.225	0.127	0.129
$\|011\rangle$	0.253	0.131	0.101	0.153	0.134
$\|100\rangle$	0.251	0.131	0.112	0.101	0.1355
$\|101\rangle$	0.242	0.074	0.078	0.135	0.105
$\|110\rangle$	0.000	0.099	0.096	0.105	0.1045
$\|111\rangle$	0.000	0.109	0.082	0.120	0.1305
Fidelity	0.96	0.16	0.10	0.15	0.10
Purity	0.93	0.20	0.19	0.17	0.18
N_U	30	37	38	24	51
N_{CNOT}	34	47	39	52	49
T_E		26.1	22.2	27.7	28.1
P_S		0.10	0.15	0.18	0.19

Table 5.: Experimental results of the 3-qubit simulation for $\phi = 0$, for the classical simulator and quantum devices ibmqx4 and ibmq20.

5.3 DISCUSSION

When studying and measuring quantum systems, particularly when the final state contains some form of superposition of basis states, some "randomness" in the results of a measurement is to be expected. The Copenhagen interpretation of quantum mechanics, currently the most widely accepted expression of the meaning of quantum systems, considers that the measurement process is itself unpredictable due to the indeterministic nature of a quantum system (prior to observation). It is why a single execution of a simulation does not allow for characterization of a quantum state, and the results of a quantum simulation are ideally considered in the limit of an infinite number of executions with associated measurements.

This unpredictability can be observed by comparing the frequency of results for the classical simulator in fig. 26 and 27 with the expected probability distribution of the final wave function (fig. 24). Even though each implementation was executed and 1000 times, for the 2-qubit simulation, and 2000 times for the 3-qubit one, there is still a noticeable variance result frequencies versus the theoretical expression of probability amplitudes. Despite this variance, the state tomography technique implemented successfully reconstructed the state

with fidelities ranging between 0.96 and 0.99, for the 2-qubit simulation with 100 shots per measurement circuit (total of 900 for the 9 measurement circuits), and a fidelity of 0.96 for the 3-qubit simulation, where each of the 27 measurement circuits was also executed 100 times, totalling 2700 executions for characterization. As such, this characterization method can be considered to have been successful given the number of executions and the reconstruction technique. However, one may quickly predict how its exponential scaling with the number of qubits makes it unfeasible for larger systems: using this technique to fully characterize the state of a 5-qubit quantum device, such as *ibmqx4*, would require implementing a total of $3^5 = 243$ measuring circuits; the full characterization of *ibmq20* would require approximately 3×10^9 circuits; this is before actual processing of the measurement data begins.

The tomographic reconstruction technique also reduces the degree of purity of the quantum state, which is expected given that the reconstruction data has a non-zero degree of uncertainty for a finite number of measurements. For the 2-qubit simulation, the reconstructed state purity ranged between 0.93 and 0.98, while for the 3-qubit simulation the state purity was estimated at 0.93. This is despite the assurance that the classical simulation produced consistently pure states, since this particular model of simulation does not account for decoherence or noise processes.

For the simulation implemented on the quantum devices, *Tenerife* (*ibmqx4*) and *Tokyo* (*ibmq20*), the compilation technique provided in QISKit, and the alternative by Zulehner and Wille (2018) were analyzed. Both compilers managed to reduce the number of single-qubit layers relative to the explicit implementation in the ideal simulator, which is achieved by condensing consecutive gates for a given qubit into a single physically implementable unitary gate $U(\theta, \phi, \lambda)$. The number of CNOT layers never diminished aside for the 2-qubit simulation for $\phi = 0$, where the alternative compiler managed to provide a significant reduction of 2 CNOT layers (from 9 to 7 layers). In fact, the alternative compiler did a better job at optimizing the number of required CNOT gates, particularly for the 3-qubit simulation on both devices. However neither compiler could eliminate unnecessary redundancy in some particular cases where two consecutive CNOT were applied in the same pair of qubits (e.g. fig. 25, middle and bottom rows). CNOT gates are self invertible, which means that two consecutive applications of this gate are equivalent with an identity operation, and as such the operations can simply be discarded. In situations where optimization techniques would preferably be avoided, the software allows for implementation of *barriers* that force the compiler to treat different sections of the circuit as separate operations.

It should be noted that throughout this work, no way of bypassing the SDK compiler was found, even if the circuit produced by the alternative compiler observed all the physical constraints of the architecture. This is particularly troubling in the case of *ibmq20*, where the compilation of the same circuit using QISKit's own compiler resulted in a significantly less efficient compilation in comparison with *ibmqx4*, despite the former having significantly better

qubit connectivity. As an example, it can be observed in fig. 7 that qubits $\{1, 7, 6, 5, 11\}$ from *ibmq20* have at least the same functional connectivity as qubits $\{0, 1, 2, 3, 4\}$ from *ibmqx4*. This constituted a limiting factor in the efficiency improvements of the alternative compiler; as such, the layer comparison between compilers, provided in tables 2 - 5 does not do justice to the improvement potential of this alternative. However, for a true comparison of simulation results and fidelity, the circuit that was experimentally executed (i.e after re-compilation by the SDK) is the one to be considered. It turns out that an option to turn off compiling functionality is indeed accessible through the SDK, but it was only found after treatment of these specific results. Considering compiler performance, the underlying conclusion is that the mapping and optimization of quantum circuits for 2-dimensional qubit architectures is a very recent problem, and one that would greatly benefit from a deeper study into the field of optimization and mapping techniques for quantum circuits.

As for the simulation results on both real devices, for the 2-qubit simulation (fig. 26), *ibmqx4* returned generally better quality results for $\phi = 0, \pi$; in the implementation for $\phi = \pi/2$ *ibmq20* returned a more uniform probability distribution, which is desirable for that particular case. These results are expected even though *ibmq20* has lower error rates for single-qubit and CNOT gates (table 1), since post-compilation circuits for this device, in comparison with the equivalent implementation for *ibmqx4*, resulted in 55-89% more CNOT gates, which greatly increased estimated execution times (T_E) and reduced the estimated probability of an error-free simulation (P_E). In both devices, state fidelity and state purity of the reconstructed density matrix dramatically decreased and generally correlate with error in distribution probabilities observed in fig. 26 as well as with the estimated figures for P_E. These results, in part attest to the sensitivity of tomographic techniques in noisier settings - as stated in the literature Gambetta et al. (2012), quantum tomography techniques are particularly sensitive to noise, and that is one of the reasons other methods for verification of quantum systems are generally considered (e.g. randomized benchmarking, section 3.3).

Fidelity ranges were worse for *ibmq20* $(0.32 - 0.45)$ than in *ibmqx4* $(0.57 - 0.64)$, which is consistent with measurement results. A decrease in fidelity figures was typically accompanied by lower state purity estimates, which is expected when the reconstructed density matrix differs from the desired one due to noisy processes instead of approximation errors (or a wrong implementation).

Besides error rates, coherence times (table 1) may also have slightly deteriorated the results. Coherence times are inherently linked with quantum gate error rates and their associated execution time - isolating for other factors, a longer coherence time implicates a lower rate of decoherence, and therefore error, for the interval when the gate is being executed. However, estimated execution times were one order of magnitude smaller than T_1 and T_2 for both devices, which is reasonable for approximate quantum computations.

As for the 3-qubit implementations, no simulation executed on either quantum device produced meaningful results, as it can be observed in fig. 27. The compiled circuits contained a large number of CNOT layers, which pushed estimated execution times to the limits of decoherence, particularly for *ibmqx4* - average coherence times for this device ($T_1 = 49.0\mu s, T_2 = 24.8\mu s$) are significantly smaller than those on *ibmq20* ($T_1 = 84.6\mu s, T_2 = 55.0\mu s$). As described on section 4.1 T_1, also known as "amplitude damping", refers to the gradual loss of energy of a qubit, i.e. its tendency to decay to the ground state, $|0\rangle$. The measurement results returned by *ibmqx* seem to indeed have a slight bias toward $|000\rangle$ and other low energy states. Even without accounting for decoherence specifically, accumulated error rates, from the number of gates executed, greatly diminished the probability of a successful simulation (P_E). This conjecture is validated by the low fidelity and purity of the reconstructed density matrix obtained through the state tomography technique.

Regarding the compilers, in the 3-qubit case the standard compiler performed significantly worse in terms of total number of CNOT layers, which have the most error. The difference in compiled circuit depth (i.e. total number of layers) between devices much less pronounced than for the 2-qubit simulations. One interesting comparison is between compilation techniques for the implementation of the simulation in *ibmq20*. The alternative compiler, despite managing to reduce the number of CNOT layers by 3 and increase the number of single-qubit layers by 27 more than doubling its amount (relatively to the standard compiler), the estimated time execution was just marginally larger, while the probability of an ideal simulation increased slightly. This illustrates just how tasking the execution of CNOT layers is in comparison with single-qubit layers.

Since the experimental algorithm relies not only in a heavy use of CNOT gates which are slower and more error-prone, but also requires finer control over superposition and amplitudes of states (unlike, for example, a quantum search algorithm which is supposed to return a single output state), particularly for quantum state tomography, it is not unforeseen that the results would shed a harsh light on the performance of these quantum devices.

One underlying conclusion is that, if experimental quantum computation is to progress into more complex quantum algorithms and circuits, this family of quantum devices would greatly benefit from an increase in coherence times and decrease in error rates, more so than an increase in the number of qubits. In fact, one of the near-term challenges for quantum computation scientists should be to find useful quantum algorithms that make use of most of the qubits available on recent quantum devices such as *ibmq20*, while keeping a complexity low enough to produce meaningful results. Other potentially useful ways to improve on the quality of these results are the development of better optimization and mapping schemes, as well as implementation of error correction procedures.

6

CONCLUSIONS

The purpose of this work is to overview the theoretical concepts around quantum simulation and quantum tomography, as well as study more sophisticated validation techniques that scale in an efficient way with system size, and build towards a potentially useful experimental application of quantum simulation and validation techniques. As it progressed, it became evident not only that the concepts of efficiency and reliability are closely related in the field of digital quantum simulation, but also that there is a pressing need for a thorough discussion of these concepts from theoretical conception to experimental implementation on NISQ devices. In such a context, an effort was made towards outlining the necessary conditions for efficiently implementing a digital simulation given the constraints imposed by present-day quantum devices.

In this dissertation, the fundamental theory in analog and digital quantum simulation was introduced, as well as the essential theoretical concepts necessary for the understanding of a simulation of the Schrödinger equation. From this and the overview of physical realizations and applications of quantum simulators, it follows that quantum simulation has the potential to be efficient, useful and within close reach of researchers beyond proof-of-concept applications.

A review of current and promising characterization and validation techniques for quantum simulation and computation was presented. Similarly to the hypothesis that arbitrary state preparation is not efficient unless derived from an efficient (i.e. polynomial) description of the state, it is believed that a full characterization of a quantum state cannot be efficiently performed unless one restricts its parameters to a polynomial set of possibilities. Verification and validation of quantum processes and computations may, however, be performed efficiently.

Randomized benchmarking is the most widely used technique for verification of current quantum computers. The fidelity figures reported from implementation of this technique are generally compared with the error rates provided by the quantum threshold theorem (Aharonov and Ben-Or, 1997), which may engender optimism that current technology is near the threshold required for fault-tolerant quantum computation. Sanders et al. (2015) gives a sobering assessment of this comparison, by determining an upper-bound on error rates from average gate fidelity estimations. For example, it is shown that it is possible for a two-qubit

gate with 99% fidelity to have an error rate of up to 13%; conversely, a two-qubit gate must have a fidelity of over 99.9995% to ensure an error rate below 1%. These findings illustrate the need for more thorough benchmarking protocols that provide better grounded expectations of quantum device performance.

The experimental part included implementing 2 and 3-qubit simulation algorithms (plus one ancilla qubit) of the Schrödinger equation of a single particle in 1 dimension. These were run on a classical simulator, and also implemented in IBM's quantum devices, the 5-qubit *ibmqx4 - Tenerife* and the more recent 20-qubit *ibmq20 - Tokyo*. For the 2-qubit simulations, even though the quantum device presented approximately correct results, significant error rates were observed. The implemented quantum state tomography techniques were able to successfully reconstruct the state in the classical simulator (which emulates a quantum simulator without any noise or decoherence), showing that the technique and associated reconstructed method are sound; in the case of the quantum devices, there was a substantial reduction in accuracy, demonstrating its sensitivity to noise and error processes. The 3-qubit simulation pushed both devices past their capability to return useful results; this was expected from the study of average error rates, decoherence times and depth of the implemented quantum circuits. Two techniques of compilation and mapping were experimentally compared; despite performing adequately, the results highlighted the problem of finding an optimal mapping using efficient resources, as well as the potential for more sophisticated optimization schemes.

Quantum simulation technology has a great room for improvement, more so in controllability and scalability performance. Quantum simulators cannot yet handle large arrays of qubits while maintaining experimentally acceptable levels of noise and decoherence. It should be noted that a scenario of quantum supremacy is not necessary to find useful uses for the technology, since even small-scale quantum simulators allow for the investigation of quantum mechanical phenomena. Research into quantum simulators, by itself, may also have a positive impact on the development of related fields, such as adiabatic quantum computation, measurement-based quantum computation, and topological quantum computation.

From the theoretical review and experimental results, the underlying conclusion is that, while there has been an extraordinary progress in implementation of quantum devices over the past decade, there is a need for more accurate, and thorough, techniques for verification of fidelity and reliability of quantum computers and quantum simulators. Experimental performance of available quantum devices may seem disappointing if one is expecting NISQ devices to provide ideal results. Arguably, current quantum devices are neither groundbreaking nor irrelevant, and should instead be regarded as a step towards more powerful quantum technologies to be developed in the future.

6.1 FUTURE WORK

From this work, several routes may be taken towards improving and expanding on the concepts described. One could venture towards exploring the fundamental theory of a class of analog quantum simulators, as well as their efficiency and reliability characteristics and techniques of validation and verification, in a similar way as it was realized in this work for digital quantum simulations. Analog quantum simulators have been getting notably more sophisticated, and are already being employed to study quantum dynamics in regimes which may be beyond the reach of classical simulators (Zhang et al., 2017). One obstacle is increasing accuracy in control, since current simulators only crudely approximates the model system in study. For that reason, analog simulators are best suited for studying features that are relatively robust with respect to introducing small sources of error. A major challenge for research using analog quantum simulators is identifying accessible properties of quantum systems which are robust with respect to error, while also hard to simulate classically.

Verification and validation techniques besides quantum state tomography, quantum process tomography and randomized benchmarking could be explored, with a bigger emphasis towards efficiency and resilience against noise. An experimental implementation and study of some of these techniques, such as Flammia and Liu (2011), on publicly available quantum devices should be within reach and provide a deeper insight on their strengths and weaknesses in noisy quantum devices. More sophisticated quantum tomography techniques, such as adaptive quantum tomography (Granade et al., 2016) or approaches based on machine learning (Torlai et al., 2018). One could also study the possibility of using currently available quantum devices for the experimental demonstration of validation protocols directed towards cloud-based quantum computations, such as quantum interactive proofing (Aharonov et al., 2017) or blind quantum computation (Fitzsimons, 2017).

One very recent problem with great potential for application is the development of compilation and mapping algorithms for quantum computers with 2 dimensional qubit lattices with nearest neighbor interactions, such as those described in this work. This problem is formally introduced by Siraichi et al. (2018), and the proposal by Zulehner and Wille (2018) was already implemented experimentally in the experimental part of this work. The implementation of simplified, and less demanding, error correction schemes could also be approached and experimentally studied on quantum devices while taking into account chip architecture.

The Schrödinger equation simulation algorithm presented here can easily be scaled for a larger number of qubits, even if such a simulation is past the limits of current devices. A more challenging prospect would be that of using the equation for the simulation of a particle in 2 dimensions, or simulating the Schrödinger equation for basic molecules, i.e. ab initio quantum chemistry digital simulations.

BIBLIOGRAPHY

Daniel S Abrams and Seth Lloyd. Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Physical Review Letters*, 83(24):5162, 1999.

Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 176–188. ACM, 1997.

Dorit Aharonov, Michael Ben-Or, Elad Eban, and Urmila Mahadev. Interactive proofs for quantum computations. *arXiv preprint arXiv:1704.04487*, 2017.

Daniel Alsina and José Ignacio Latorre. Experimental test of mermin inequalities on a five-qubit quantum computer. *Physical Review A*, 94(1):012314, 2016.

Dimitris G Angelakis, Marcelo Franca Santos, and Sougato Bose. Photon-blockade-induced mott transitions and x y spin models in coupled cavity arrays. *Physical Review A*, 76(3): 031805, 2007.

Johan Åqvist and Arieh Warshel. Simulation of enzyme reactions using valence bond force fields and other hybrid quantum/classical approaches. *Chemical reviews*, 93(7):2523–2544, 1993.

Javier Argüello-Luengo, Alejandro González-Tudela, Tao Shi, Peter Zoller, and J Ignacio Cirac. Analog quantum chemistry simulation. *arXiv preprint arXiv:1807.09228*, 2018.

LM Artiles, RD Gill, and MI Gut¸ă. An invitation to quantum tomography. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):109–134, 2005.

Giuliano Benenti and Giuliano Strini. Quantum simulation of the single-particle schrödinger equation. *American Journal of Physics*, 76(7):657–662, 2008.

Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22(5):563–591, 1980.

Charles H Bennett, David P DiVincenzo, John A Smolin, and William K Wootters. Mixed-state entanglement and quantum error correction. *Physical Review A*, 54(5):3824, 1996.

Dominic W Berry, Andrew M Childs, Richard Cleve, Robin Kothari, and Rolando D Somma. Exponential improvement in precision for simulating sparse hamiltonians. In *Forum of Mathematics, Sigma*, volume 5. Cambridge University Press, 2017.

Bruce M Boghosian and Washington Taylor IV. Simulating quantum mechanics on a quantum computer. *Physica D: Nonlinear Phenomena*, 120(1-2):30–42, 1998.

Justin G Bohnet, Brian C Sawyer, Joseph W Britton, Michael L Wall, Ana Maria Rey, Michael Foss-Feig, and John J Bollinger. Quantum spin dynamics and entanglement generation with hundreds of trapped ions. *Science*, 352(6291):1297–1301, 2016.

Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595, 2018.

Max Born. Zur quantenmechanik der stoßvorgänge. *Zeitschrift für Physik*, 37(12):863–867, Dec 1926. ISSN 0044-3328. doi: 10.1007/BF01397477. URL https://doi.org/10.1007/BF01397477.

Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Physical Review A*, 86(5):052329, 2012.

Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005.

Sergey Bravyi, David P DiVincenzo, Daniel Loss, and Barbara M Terhal. Quantum simulation of many-body hamiltonians using perturbation theory with bounded-strength interactions. *Physical review letters*, 101(7):070503, 2008.

Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. Universal blind quantum computation. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 517–526. IEEE, 2009.

Katherine L Brown, William J Munro, and Vivien M Kendon. Using quantum computers for quantum simulation. *Entropy*, 12(11):2268–2307, 2010.

Iulia Buluta and Franco Nori. Quantum simulators. *Science*, 326(5949):108–111, 2009.

Tim Byrnes, Na Young Kim, Kenichiro Kusudo, and Yoshihisa Yamamoto. Quantum simulation of fermi-hubbard models in semiconductor quantum-dot arrays. *Physical Review B*, 78(7):075320, 2008.

Earl T Campbell, Barbara M Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172, 2017.

Davide Castelvecchi. Quantum computers ready to leap out of the lab in 2017. *Nature News*, 541(7635):9, 2017.

David Ceperley and Berni Alder. Quantum monte carlo. *Science*, 231(4738):555–560, 1986.

J Chiaverini and WE Lybarger Jr. Laserless trapped-ion quantum simulations without spontaneous scattering using microtrap arrays. *Physical Review A*, 77(2):022324, 2008.

Lilian Childress and Ronald Hanson. Diamond nv centers for quantum computing and quantum networks. *MRS bulletin*, 38(2):134–138, 2013.

Andrew M Childs. Secure assisted quantum computation. *arXiv preprint quant-ph/0111046*, 2001.

Isaac Chuang. Lecture 19: How to build your own quantum computer, November 2003.

Isaac L. Chuang and M. A. Nielsen. Prescription for experimental determination of the dynamics of a quantum black box. *Journal of Modern Optics*, 44-11(12):2455–2467, 1997. ISSN 13623044. doi: 10.1080/09500349708231894.

J. Ignacio Cirac and Peter Zoller. Goals and opportunities in quantum simulation. *Nature Physics*, 8(4):264–266, 2012. ISSN 1745-2473. doi: 10.1038/nphys2275. URL http://www.nature.com/doifinder/10.1038/nphys2275.

Patrick J Coles, Stephan Eidenbenz, Scott Pakin, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, et al. Quantum algorithm implementations for beginners. *arXiv preprint arXiv:1804.03719*, 2018.

Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

Marcus P da Silva, Olivier Landon-Cardinal, and David Poulin. Practical characterization of quantum devices without tomography. *Physical Review Letters*, 107(21):210404, 2011.

Francesco De Nicola, Linda Sansoni, Andrea Crespi, Roberta Ramponi, Roberto Osellame, Vittorio Giovannetti, Rosario Fazio, Paolo Mataloni, and Fabio Sciarrino. Quantum simulation of bosonic-fermionic noninteracting particles in disordered systems via a quantum walk. *Physical Review A*, 89(3):032322, 2014.

David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A*, 400(1818):97–117, 1985.

Roberto Di Candia, Julen S Pedernales, Adolfo del Campo, Enrique Solano, and Jorge Casanova. Quantum simulation of dissipative processes without reservoir engineering. *Scientific reports*, 5:9981, 2015.

Giuseppe Di Molfetta and Armando Pérez. Quantum walks as simulators of neutrino oscillations in a vacuum and matter. *New Journal of Physics*, 18(10):103038, 2016.

David P DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik: Progress of Physics*, 48(9-11):771–783, 2000.

Ross Dorner, John Goold, and Vlatko Vedral. Towards quantum simulations of biological information flow. *Interface focus*, page rsfs20110109, 2012.

Ron O Dror, Robert M Dirks, JP Grossman, Huafeng Xu, and David E Shaw. Biomolecular simulation: a computational microscope for molecular biology. *Annual review of biophysics*, 41:429–452, 2012.

Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Physical review letters*, 102(11):110502, 2009.

Joseph Emerson, Robert Alicki, and Karol Życzkowski. Scalable noise estimation with random unitary operators. *Journal of Optics B: Quantum and Semiclassical Optics*, 7(10):S347, 2005.

Christian Fey, Tobias Schaetz, and Ralf Schützhold. Ion-trap analog of particle creation in cosmology. *Physical Review A*, 98(3):033407, 2018.

Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982. ISSN 00207748. doi: 10.1007/BF02650179.

Uwe R Fischer and Ralf Schützhold. Quantum simulation of cosmic inflation in two-component bose-einstein condensates. *Physical Review A*, 70(6):063615, 2004.

Joseph F Fitzsimons. Private quantum computation: an introduction to blind quantum computing and related protocols. *npj Quantum Information*, 3(1):23, 2017.

Steven T Flammia and Yi-Kai Liu. Direct fidelity estimation from few pauli measurements. *Physical Review Letters*, 106(23):230501, 2011.

Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3): 032324, 2012a.

Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. Towards practical classical processing for the surface code. *Physical review letters*, 108(18):180501, 2012b.

Jay M Gambetta, AD Córcoles, Seth T Merkel, Blake R Johnson, John A Smolin, Jerry M Chow, Colm A Ryan, Chad Rigetti, S Poletto, Thomas A Ohki, et al. Characterization of addressability by simultaneous randomized benchmarking. *Physical review letters*, 109(24): 240504, 2012.

IM Georgescu, Sahel Ashhab, and Franco Nori. Quantum simulation. *Reviews of Modern Physics*, 86(1):153, 2014.

Rene Gerritsma, Gerhard Kirchmair, Florian Zähringer, E Solano, R Blatt, and CF Roos. Quantum simulation of the dirac equation. *Nature*, 463(7277):68, 2010.

Alexandru Gheorghiu, Theodoros Kapourniotis, and Elham Kashefi. Verification of quantum computation: An overview of existing approaches. *arXiv preprint arXiv:1709.06984*, 2017.

S Giovanazzi. Hawking radiation in sonic black holes. *Physical review letters*, 94(6):061302, 2005.

Christopher Granade, Joshua Combes, and DG Cory. Practical bayesian tomography. *New Journal of Physics*, 18(3):033024, 2016.

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

Philipp Hauke, Fernando M Cucchietti, Luca Tagliacozzo, Ivan Deutsch, and Maciej Lewenstein. Can one trust quantum simulators? *Reports on Progress in Physics*, 75(8):082401, 2012. ISSN 0034-4885. doi: 10.1088/0034-4885/75/8/082401. URL http://stacks.iop.org/0034-4885/75/i=8/a=082401?key=crossref.5111d425c71ba09555b72de5122310dc.

Bettina Heim, Troels F Rønnow, Sergei V Isakov, and Matthias Troyer. Quantum versus classical annealing of ising spin glasses. *Science*, 348(6231):215–217, 2015.

Cornelius Hempel, Christine Maier, Jonathan Romero, Jarrod McClean, Thomas Monz, Heng Shen, Petar Jurcevic, Ben Lanyon, Peter Love, Ryan Babbush, et al. Quantum chemistry calculations on a trapped-ion quantum simulator. *arXiv preprint arXiv:1803.10238*, 2018.

Toivo Hensgens, Takafumi Fujita, Laurens Janssen, Xiao Li, CJ Van Diepen, Christian Reichl, Werner Wegscheider, S Das Sarma, and Lieven MK Vandersypen. Quantum simulation of a fermi–hubbard model using a semiconductor quantum dot array. *Nature*, 548(7665):70, 2017.

Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

Zdenek Hradil. Quantum-state estimation. *Physical Review A*, 55(3):R1561, 1997.

IBM. Ibm makes quantum computing available on ibm cloud, 2016. URL https://www-03.ibm.com/press/us/en/pressrelease/49661.wss. Online; accessed 1-October-2018.

IBM. Ibm q - quantum computing, 2018a. URL https://www.research.ibm.com/ibm-q/. Online; accessed 1-October-2018.

IBM. Qiskit backend information - tenerife, 2018b. URL https://github.com/Qiskit/qiskit-backend-information/tree/master/backends/tenerife/V1. Online; accessed 1-October-2018.

IBM. Ibm q experience - composer, 2018c. URL https://quantumexperience.ng.bluemix.net/qx/editor. Online; accessed 1-October-2018.

IBM. Quantum devices & simulators - ibm q, 2018d. URL https://www.research.ibm.com/ibm-q/technology/devices/. Online; accessed 1-October-2018.

IBM. Qiskit | quantum information science kit, 2018e. URL https://qiskit.org/. Online; accessed 1-October-2018.

Intel. Intel delivers 17-qubit superconducting chip with advanced packaging to qutech, 2017. URL https://newsroom.intel.com/news/intel-delivers-17-qubit-superconducting-chip-advanced-packaging-qutech/. Online; accessed 1-October-2018.

Daniel FV James, Paul G Kwiat, William J Munro, and Andrew G White. On the measurement of qubits. In *Asymptotic Theory Of Quantum Statistical Inference: Selected Papers*, pages 509–538. World Scientific, 2005.

Alastair Kay and Dimitris G Angelakis. Reproducing spin lattice models in strongly coupled atom-cavity systems. *EPL (Europhysics Letters)*, 84(2):20001, 2008.

Emanuel Knill, Raymond Laflamme, and Lorenza Viola. Theory of quantum error correction for general noise. *Physical Review Letters*, 84(11):2525, 2000.

Emanuel Knill, D Leibfried, R Reichle, J Britton, RB Blakestad, John D Jost, C Langer, R Ozeri, Signe Seidelin, and David J Wineland. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1):012307, 2008.

Jens Koch, M Yu Terri, Jay Gambetta, Andrew A Houck, DI Schuster, J Majer, Alexandre Blais, Michel H Devoret, Steven M Girvin, and Robert J Schoelkopf. Charge-insensitive qubit design derived from the cooper pair box. *Physical Review A*, 76(4):042319, 2007.

KS Kravtsov, SS Straupe, IV Radchenko, NMT Houlsby, F Huszár, and SP Kulik. Experimental adaptive bayesian tomography. *Physical Review A*, 87(6):062122, 2013.

Raymond Laflamme, Cesar Miquel, Juan Pablo Paz, and Wojciech Hubert Zurek. Perfect quantum error correcting code. *Physical Review Letters*, 77(1):198, 1996.

Ben P Lanyon, Cornelius Hempel, Daniel Nigg, Markus Müller, R Gerritsma, F Zähringer, P Schindler, JT Barreiro, M Rambach, G Kirchmair, et al. Universal digital quantum simulation with trapped ions. *Science*, 334(6052):57–61, 2011.

Benjamin P Lanyon, James D Whitfield, Geoff G Gillett, Michael E Goggin, Marcelo P Almeida, Ivan Kassal, Jacob D Biamonte, Masoud Mohseni, Ben J Powell, Marco Barbieri, et al. Towards quantum chemistry on a quantum computer. *Nature chemistry*, 2(2):106, 2010.

BP Lanyon, C Maier, M Holzäpfel, T Baumgratz, C Hempel, P Jurcevic, I Dhand, AS Buyskikh, AJ Daley, M Cramer, et al. Efficient tomography of a quantum many-body system. *Nature Physics*, 13(12):1158, 2017.

Craig S Lent, Beth Isaksen, and Marya Lieberman. Molecular quantum-dot cellular automata. *Journal of the American Chemical Society*, 125(4):1056–1063, 2003.

Maciej Lewenstein, Anna Sanpera, Veronica Ahufinger, Bogdan Damski, Aditi Sen, and Ujjwal Sen. Ultracold atomic gases in optical lattices: mimicking condensed matter physics and beyond. *Advances In Physics*, 56(2):243–379, 2007.

Jian Li, MP Silveri, KS Kumar, J-M Pirkkalainen, A Vepsäläinen, WC Chien, J Tuorila, MA Sillanpää, PJ Hakonen, EV Thuneberg, et al. Motional averaging in a superconducting qubit. *Nature communications*, 4:1420, 2013.

Daniel A Lidar and Ofer Biham. Simulating ising spin glasses on a quantum computer. *Physical Review E*, 56(3):3661, 1997.

Seth Lloyd. Universal quantum simulators. *Science*, pages 1073–1078, 1996.

Dawei Lu, Boruo Xu, Nanyang Xu, Zhaokai Li, Hongwei Chen, Xinhua Peng, Ruixue Xu, and Jiangfeng Du. Quantum chemistry simulation on quantum computers: theories and experiments. *Physical Chemistry Chemical Physics*, 14(26):9411–9420, 2012.

Xiao-song Ma, Borivoje Dakić, Sebastian Kropatschek, William Naylor, Yang-hao Chan, Zhexuan Gong, Lu-ming Duan, Anton Zeilinger, and Philip Walther. Towards photonic quantum simulation of ground states of frustrated heisenberg spin systems. *Scientific reports*, 4: 3583, 2014.

Easwar Magesan, Jay M Gambetta, Blake R Johnson, Colm A Ryan, Jerry M Chow, Seth T Merkel, Marcus P Da Silva, George A Keefe, Mary B Rothwell, Thomas A Ohki, et al. Efficient measurement of quantum gate error by interleaved randomized benchmarking. *Physical review letters*, 109(8):080505, 2012.

Sabrina Maniscalco, Jyrki Piilo, F Intravaia, F Petruccione, and A Messina. Simulating quantum brownian motion with single trapped ions. *Physical Review A*, 69(5):052101, 2004.

Efstratios Manousakis. A quantum-dot array as model for copper-oxide superconductors: A dedicated quantum simulator for the many-fermion problem. *Journal of low temperature physics*, 126(5-6):1501–1513, 2002.

Esteban A Martinez, Christine A Muschik, Philipp Schindler, Daniel Nigg, Alexander Erhard, Markus Heyl, Philipp Hauke, Marcello Dalmonte, Thomas Monz, Peter Zoller, et al. Real-time dynamics of lattice gauge theories with a few-qubit quantum computer. *Nature*, 534 (7608):516–519, 2016.

Nicolas C Menicucci, S Jay Olson, and Gerard J Milburn. Simulating quantum effects of cosmological expansion using a static ion trap. *New Journal of Physics*, 12(9):095019, 2010.

David A Meyer. Quantum computing classical physics. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 360(1792): 395–405, 2002.

Masoud Mohseni, AT Rezakhani, and DA Lidar. Quantum-process tomography: Resource analysis of different strategies. *Physical Review A*, 77(3):032322, 2008.

C Monroe, WC Campbell, EE Edwards, R Islam, D Kafri, S Korenblit, A Lee, P Richerme, C Senko, and J Smith. Quantum simulation of spin models with trapped ions. *Proceedings of the International School of Physics 'Enrico Fermi,'Course 189*, pages 169–187, 2015.

Sarah Mostame and Ralf Schützhold. Quantum simulator for the ising model with electrons floating on a helium film. *Physical review letters*, 101(22):220501, 2008.

Alexandra E Moylett and Peter S Turner. Quantum simulation of partially distinguishable boson sampling. *Physical Review A*, 97(6):062329, 2018.

Leonie Mueck. Quantum reform. *Nature chemistry*, 7(5):361, 2015.

PD Nation, MP Blencowe, AJ Rimberg, and E Buks. Analogue hawking radiation in a dc-squid array transmission line. *Physical review letters*, 103(8):087004, 2009.

M Nest. Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond. *arXiv preprint arXiv:0811.0898*, 2008.

M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. ISBN 9781139495486. URL https://books.google.pt/books?id=-s4DEy7o-a0C.

PJJ O'Malley, Ryan Babbush, ID Kivlichan, Jonathan Romero, JR McClean, Rami Barends, Julian Kelly, Pedram Roushan, Andrew Tranter, Nan Ding, et al. Scalable quantum simulation of molecular energies. *Physical Review X*, 6(3):031007, 2016.

Jason Pearson, GuanRu Feng, Chao Zheng, and GuiLu Long. Experimental quantum simulation of avian compass in a nuclear magnetic resonance system. *Science China Physics, Mechanics & Astronomy*, 59(12):120312, 2016.

Xinhua Peng, Jingfu Zhang, Jiangfeng Du, and Dieter Suter. Quantum simulation of a system with competing two-and three-body interactions. *Physical review letters*, 103(14):140501, 2009.

Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Physical Review A*, 83(3):032302, 2011.

Diego Porras and J Ignacio Cirac. Effective quantum spin systems with trapped ions. *Physical review letters*, 92(20):207901, 2004.

Diego Porras and J Ignacio Cirac. Quantum manipulation of trapped ions in two dimensional coulomb crystals. *Physical review letters*, 96(25):250501, 2006.

J. F. Poyatos, J. I. Cirac, and P. Zoller. Complete Characterization of a Quantum Process: The Two-Bit Quantum Gate. *Physical Review Letters*, 78(2):390–393, 1997. ISSN 0031-9007. doi: 10.1103/PhysRevLett.78.390. URL https://link.aps.org/doi/10.1103/PhysRevLett.78.390.

Philipp M Preiss, Ruichao Ma, M Eric Tai, Alexander Lukin, Matthew Rispoli, Philip Zupancic, Yoav Lahini, Rajibul Islam, and Markus Greiner. Strongly correlated quantum walks in optical lattices. *Science*, 347(6227):1229–1233, 2015.

John Preskill. Lecture notes for physics 229: Quantum information and computation. *California Institute of Technology*, 16, 1998.

John Preskill. Quantum computing in the nisq era and beyond. *arXiv preprint arXiv:1801.00862*, 2018.

Timothy Proctor, Kenneth Rudinger, Kevin Young, Mohan Sarovar, and Robin Blume-Kohout. What randomized benchmarking actually measures. *Physical review letters*, 119 (13):130502, 2017.

Bo Qi, Zhibo Hou, Li Li, Daoyi Dong, Guoyong Xiang, and Guangcan Guo. Quantum state tomography via linear regression estimation. *Scientific reports*, 3:3496, 2013.

Wei Qin and Franco Nori. Controllable single-photon transport between remote coupled-cavity arrays. *Physical Review A*, 93(3):032337, 2016.

Sadegh Raeisi, Nathan Wiebe, and Barry C Sanders. Quantum-circuit design for efficient simulations of many-body quantum dynamics. *New Journal of Physics*, 14(10):103017, 2012.

Martin Ringbauer. *Exploring Quantum Foundations with Single Photons*. Springer, 2017.

Yuval R Sanders, Joel J Wallman, and Barry C Sanders. Bounding quantum gate error rate based on reported average fidelity. *New Journal of Physics*, 18(1):012002, 2015.

Yuval R Sanders, Guang Hao Low, Artur Scherer, and Dominic W Berry. Black-box quantum state preparation without arithmetic. *arXiv preprint arXiv:1807.03206*, 2018.

P Santini, S Carretta, F Troiani, and G Amoretti. Molecular nanomagnets as quantum simulators. *Physical review letters*, 107(23):230502, 2011.

Philipp Schindler, Markus Müller, Daniel Nigg, Julio T Barreiro, Esteban A Martinez, Markus Hennrich, T Monz, Sebastian Diehl, Peter Zoller, and Rainer Blatt. Quantum simulation of dynamical maps with trapped ions. *Nature Physics*, 9(6):361, 2013.

Travis L Scholten and Robin Blume-Kohout. Behavior of the maximum likelihood in quantum state tomography. *New Journal of Physics*, 20(2):023050, 2018.

Erwin Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Physical review*, 28(6):1049, 1926.

Vivek V Shende, Stephen S Bullock, and Igor L Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6): 1000–1010, 2006.

SN Shevchenko, Sahel Ashhab, and Franco Nori. Landau–zener–stückelberg interferometry. *Physics Reports*, 492(1):1–30, 2010.

Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 113–125. ACM, 2018.

John A Smolin, Jay M Gambetta, and Graeme Smith. Efficient method for computing the maximum-likelihood quantum state from measurements with additive gaussian noise. *Physical review letters*, 108(7):070502, 2012.

Andrei N Soklakov and Rüdiger Schack. Efficient state preparation for a register of quantum bits. *Physical Review A*, 73(1):012307, 2006.

Rolando Somma, Gerardo Ortiz, James E Gubernatis, Emanuel Knill, and Raymond Laflamme. Simulating physical phenomena by quantum networks. *Physical Review A*, 65(4):042323, 2002.

Ashley M Stephens. Fault-tolerant thresholds for quantum error correction with the surface code. *Physical Review A*, 89(2):022321, 2014.

Stanislav Sergeevich Straupe. Adaptive quantum tomography. *JETP letters*, 104(7):510–522, 2016.

Masuo Suzuki. Improved trotter-like formula. *Physics Letters A*, 180(3):232–234, 1993.

Barbara M Terhal and David P DiVincenzo. Classical simulation of noninteracting-fermion quantum circuits. *Physical Review A*, 65(3):032325, 2002.

Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, 14(5):447, 2018.

Hale F Trotter. On the product of semi-groups of operators. *Proceedings of the American Mathematical Society*, 10(4):545–551, 1959.

Matthias Troyer and Uwe-Jens Wiese. Computational complexity and fundamental limitations to fermionic quantum monte carlo simulations. *Physical review letters*, 94(17):170201, 2005.

Dimitris I Tsomokos, Sahel Ashhab, and Franco Nori. Fully connected network of superconducting qubits in a cavity. *New Journal of Physics*, 10(11):113020, 2008.

Dimitris I Tsomokos, Sahel Ashhab, and Franco Nori. Using superconducting qubit circuits to engineer exotic lattice systems. *Physical Review A*, 82(5):052311, 2010.

Juha J Vartiainen, Mikko Möttönen, and Martti M Salomaa. Efficient decomposition of quantum gates. *Physical review letters*, 92(17):177902, 2004.

Frank Verstraete, Michael M Wolf, and J Ignacio Cirac. Quantum computation and quantum-state engineering driven by dissipation. *Nature physics*, 5(9):633, 2009.

David S Wang, Austin G Fowler, and Lloyd CL Hollenberg. Quantum computing with nearest neighbor interactions and error rates over 1%. *arXiv preprint arXiv:1009.3686*, 2010a.

Hefeng Wang, Lian-Ao Wu, Yu-xi Liu, and Franco Nori. Measurement-based quantum phase estimation algorithm for finding eigenvalues of non-unitary matrices. *Physical Review A*, 82(6):062303, 2010b.

Hefeng Wang, Sahel Ashhab, and Franco Nori. Quantum algorithm for simulating the dynamics of an open quantum system. *Physical Review A*, 83(6):062317, 2011.

Stephen Wiesner. Simulations of many-body quantum systems by a quantum computer. *arXiv preprint quant-ph/9603028*, 1996.

Xing-Can Yao, Tian-Xiong Wang, Ping Xu, He Lu, Ge-Sheng Pan, Xiao-Hui Bao, Cheng-Zhi Peng, Chao-Yang Lu, Yu-Ao Chen, and Jian-Wei Pan. Observation of eight-photon entanglement. *Nature photonics*, 6(4):225–228, 2012.

JQ You and Franco Nori. Atomic physics and quantum optics using superconducting circuits. *Nature*, 474(7353):589, 2011.

Man-Hong Yung, Daniel Nagaj, James D Whitfield, and Alán Aspuru-Guzik. Simulation of classical thermal states on a quantum computer: A transfer-matrix approach. *Physical Review A*, 82(6):060302, 2010.

Alexandre M Zagoskin, Sergey Savel'ev, and Franco Nori. Modeling an adiabatic quantum computer via an exact map to a gas of particles. *Physical review letters*, 98(12):120503, 2007.

Christof Zalka. Simulating quantum systems on a quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, pages 313–322. The Royal Society, 1998.

Jiehang Zhang, Guido Pagano, Paul W Hess, Antonis Kyprianidis, Patrick Becker, Harvey Kaplan, Alexey V Gorshkov, Z-X Gong, and Christopher Monroe. Observation of a many-body dynamical phase transition with a 53-qubit quantum simulator. *Nature*, 551(7682): 601, 2017.

Alwin Zulehner and Robert Wille. Compiling su (4) quantum circuits to ibm qx architectures. *arXiv preprint arXiv:1808.05661*, 2018.

A

QUANTUM COMPUTING

A.1 HILBERT SPACE AND THE BRA-KET NOTATION

In quantum mechanics, wave functions and other quantum states can be represented as vectors in a complex Hilbert space, an abstract vector space possessing the structure of an inner product. When dealing with the algebra of quantum algorithms operating with n qubits, these spaces are limited to 2^n dimensions. Bra–ket notation is a standard notation for describing quantum states. It uses angle brackets (the $\langle$ and $\rangle$ symbols), and a vertical bar between objects to denote the scalar product of vectors or the action of a linear functional on a vector in a complex vector space.

Quantum superpositions can be described as **vector sums** of the constituent states. For example, an electron in the state $|1\rangle + i\,|2\rangle$ is in a quantum superposition of the state $|1\rangle$ and $|2\rangle$.

The **scalar product** is written as $\langle\phi \mid \psi\rangle$, where the left part is called the *bra*, typically represented as a row vector, and the right part is called the *ket*, typically represented as a column vector. A bra is the Hermitian conjugate of a ket with the same label.

In quantum mechanics, the product $\langle\phi|\,|\psi\rangle$ is the probability amplitude that determines how $|\psi\rangle$ is linearly decomposed into $|\phi\rangle$. The probability itself is the absolute square of the amplitude, $|\,\langle\phi|\,|\psi\rangle\,|^2$

The **outer product** is written as $|\psi\rangle\,\langle\phi|$ which can also be represented as a matrix multiplication, since a column vector times a row vector equals a matrix.

One of the uses of the outer product is to construct projection operators. Given a ket $|\psi\rangle$ of norm 1, the orthogonal projection onto the subspace spanned by $|\psi\rangle$ is $|\psi\rangle\,\langle\psi|$.

Two Hilbert spaces V and W may form a third space $V \otimes W$ by a **tensor product**. If $|\psi\rangle$ is a ket in V and $|\phi\rangle$ us a ket in W, the direct product of the two kets is a ket in $V \otimes W$. The direct product may be written in various notations: $|\psi\rangle|\phi\rangle,\ |\psi\rangle \otimes |\phi\rangle,\ |\psi\phi\rangle,\ |\psi, \phi\rangle$.

The tensor product is useful to describe quantum systems composed of multiple subsystems.

A **linear operator** is a linear map that inputs a ket and outputs a ket. In an N-dimensional Hilbert space, $|\psi\rangle$ can be written as an $N \times 1$ column vector, and then a linear operator A is an $N \times N$ matrix with complex entries. The ket $A|\psi\rangle$ can be computed by regular matrix multiplication.

Dynamics of a quantum state are described by unitary linear operators U on the Hilbert space of quantum states. Such that the transformation acting on a state $|\psi\rangle \rightarrow U|\psi\rangle$. Measurements are observable physical quantities (such as energy or momentum) represented by self-adjoint operators in Hilbert space. For a given state $|\psi\rangle$, the expectation value of the observable O is obtained by computing $\langle\psi| O |\psi\rangle$.

Wave function **normalization** is the scaling of a wave function so that its norm is 1.

A.2 Quantum computing programming model

A *qubit* (short for quantum bit) is a two-dimensional quantum mechanical system that is in a state $|q\rangle = \alpha|0\rangle + \beta|1\rangle$, where the ket notation:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \tag{44}$$

is shorthand for the vectors encoding the two basis states. α and β are the complex numbers with $|\alpha|^2 + |\beta|^2 = 1$. If the qubit gets measured, it will be observed with state $|0\rangle$ with probability $|\alpha|^2$, or in state $|1\rangle$ with probability $|\beta|^2$. This normalization of probability amplitudes allows for an alternative representation of a single qubit state:

$$|q\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)e^{i\varphi}|1\rangle \tag{45}$$

where $0 \leq \theta \leq \pi$ and $0 \leq \varphi < 2\pi$. From this it is clear that there is a one-to-one correspondence between qubit states ($\mathbb{C}^2$) and the points on the surface of a unit sphere ($\mathbb{R}^3$). This is called the Bloch sphere representation of a qubit state.

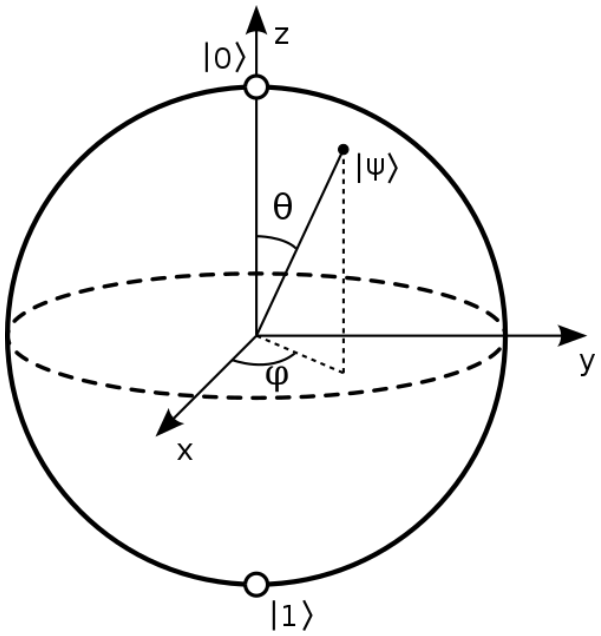


Figure 28.: Bloch sphere representation of a single qubit state $|\psi\rangle$

The joint state of a system of qubits is described by the tensor product $\otimes$. For a system of, two qubits, for example, each in a state $|q_j\rangle = \alpha_j |0\rangle + \beta_j |1\rangle$, for $j = 1, 2$, the state is:

$$|q_1\rangle \otimes |q_2\rangle = |q_1 q_2\rangle = \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle \tag{46}$$

A measurement of both qubits could result in any of the four possibilities associated with the four basis vectors.

By analogy to classical logical gates such as NOT and AND, a basic operation on a qubit or system of qubits is called a gate, which mathematically is a unitary transformation U. In contrast to classical gates, unitaries are reversible and hence the number of input qubits always equals the number of output qubits. The gates mentioned during this dissertation and their algebraic representation follow in section A.3.

A quantum algorithm using n-qubits can be graphically represented as a quantum circuit with n horizontal lines, each representing a qubit. Here, quantum gates are represented by boxes over one (single qubit) or two lines (CNOT). Operations on the qubits are ordered from left to right.

A quantum gate $\hat{U}$ acting on qubit q_1 of an n-qubit register $|q_1 \cdots q_n\rangle$ can be mathematically expressed as:

$$\hat{U} |q_1\rangle \otimes |q_2 \cdots q_n\rangle = (\hat{X} \otimes \hat{U} \otimes \cdots \otimes \hat{I}) |q_1 q_2 \cdots q_n\rangle \tag{47}$$

where $\hat{I}$ is simply the identity operation (i.e. leaving the qubit unchanged). In this work, the algebraic expression representing a quantum gate $\hat{U}$ operating on qubit q_j is written as $\hat{U}_j$ where:

$$\hat{I}_1 \otimes \cdots \otimes \hat{U}_j \otimes \cdots \otimes \hat{I}_n \tag{48}$$

represents the full algebraic expression for the operation. This allows expressions representing the action of a string of quantum gates over a register to adopt a more readable form such as demonstrated in equation 36 and throughout section 4.2:

$$\hat{U}_{prep} = \hat{X}_1 \,.\, \hat{CX}_{12} \,.\, \hat{X}_1 \,.\, \hat{H}_1 \tag{49}$$

which can be written in its expanded form as $\hat{U}_{prep} = (\hat{X}_1 \otimes \hat{I}_2) \,.\, \hat{CX}_{12} \,.\, (\hat{X}_1 \otimes \hat{I}_2) \,.\, (\hat{H}_1 \otimes \hat{I}_2)$.

A.3 Quantum gates

This section describes the matrix representation of all the quantum gates referenced throughout this work. In QISKit, the most general single qubit gate is the unitary U_3 gate:

$$\hat{U}_3(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+i\phi}\sin(\theta/2) \end{pmatrix}$$

The software also allows for more restricted versions of this unitary:

$$\hat{U}_2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i\lambda+i\phi} \end{pmatrix} = \hat{U}_3(\pi/2, \phi, \lambda)$$

$$\hat{U}_1(\lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} = \hat{U}_3(0, 0, \lambda)$$

Here, $U_1(\lambda)$ is equivalent to a quantum phase gate, P_λ. Other referenced single qubit gates are:

$$\hat{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; \qquad \hat{H} = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix};$$

$$\hat{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \hat{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \hat{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$\hat{R}_X(\theta) = \begin{pmatrix} \cos\theta/2 & -i\sin\theta/2 \\ -i\sin\theta/2 & \cos\theta/2 \end{pmatrix} \quad \hat{R}_Y(\theta) = \begin{pmatrix} \cos\theta/2 & -\sin\theta/2 \\ \sin\theta/2 & \cos\theta/2 \end{pmatrix} \quad \hat{R}_Z(\phi) = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}$$

For controlled-NOT (or controlled-X) operations acting on qubits $|q_1 q_2\rangle$ the gate $CNOT_{12}$, with qubit 1 as control and qubit 2 as target, has a different representation than $CNOT_{21}$, with qubit 2 as control and qubit 1 as target:

$$\hat{C}X_{12} = \hat{C}NOT_{12} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}; \quad \hat{C}X_{21} = \hat{C}NOT_{21} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

The controlled-phase rotation, $CU_1(\lambda)$ or $CP_{(\lambda}$), has the same representation independently of which qubit is the target, and which is the control:

$$\hat{CU}_1(\lambda)_{12} = CP_{(\lambda)12} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\lambda} \end{pmatrix}$$

B

QISKIT IMPLEMENTATION

The experiments were run using the QISKit Terra SDK, version 0.5.7, which requires Python 3.5 or later. The code presented can be executed interactively with Jupyter Notebook. The alternative compiler (Zulehner and Wille, 2018) is available in http://iic.jku.at/eda/research/ibm_qx_mapping/.

This software should be independent of operating system, and theoretically, there are no specific architecture requirements as long as the software dependencies are satisfied. Here. all results were obtained using a machine with a 2.5GHz Intel Core i5 processor and 6GB of DDR3 memory, running Windows 10 64-bit.

Executing the experiments in IBM's quantum devices requires the registering of a private token associated with a (free) account, which may be created in https://quantumexperience.ng.bluemix.net/qx/experience. Before executing the scripts below, the string `'TOKEN'` should be replaced with a string containing a valid token.

B.1 2-QUBIT ALGORITHMS

To run the simulation with the different parameters presented in this work, some initial variables need to be changed.

$\phi =$		$\lvert\psi(x_m, \Delta t)\rangle =$		phi=		idealvec=	
	0		$\frac{1}{\sqrt{2}}(\lvert 01\rangle + \lvert 10\rangle)$		0		`[0,1/sqrt(2),1/sqrt(2),0]`
	$\pi/2$		$\frac{1}{2}(e^{i\pi/4}\lvert 00\rangle + e^{-i\pi/4}\lvert 01\rangle$ $+e^{i\pi/4}\lvert 10\rangle + e^{-i\pi/4}\lvert 11\rangle)$		pi/2		`(1/sqrt(8))[1+j,1-j,1-j,1+j]`
	π		$\frac{1}{\sqrt{2}}(\lvert 00\rangle + \lvert 01\rangle)$		pi		`[1/sqrt(2),0,0,1/sqrt(2)]`

Table 6.: Characteristic phase shift and associated desired final state for each implementation (left); corresponding variables to be modified (right).

To change between devices, the string variable `backend` should be set to either `'ibmqx4'` or `'ibmq_20_tokyo'`. For the alternative compiler, the mapping has to be set explicitly.

For *ibmqx4*, `mapping = [[1, 0], [2, 0], [2, 1], [3, 2], [3, 4], [4, 2]]`.

For *ibmq20*, `mapping = [[0, 1], [0, 5], [1, 0], [1, 2], [1, 6], [1, 7], [2, 1], [2, 3], [2, 6], [3, 2], [3, 8], [3, 9], [4, 8], [4, 9], [5, 0], [5, 6], [5, 10], [5, 11], [6, 1], [6, 2], [6, 5], [6, 7], [6, 10], [6, 11], [7, 1], [7, 6], [7, 8], [7, 12], [7, 13], [8, 3], [8, 4], [8, 7], [8, 9], [8, 12], [8, 13], [9, 3], [9, 4], [9, 8], [10, 5], [10, 6], [10, 11], [10, 15], [11, 5], [11, 6], [11, 10], [11, 12], [11, 16], [11, 17], [12, 7], [12, 8], [12, 11], [12, 13], [12, 16], [13, 7], [13, 8], [13, 12], [13, 14], [13, 18], [13, 19], [14, 13], [15, 10], [15, 16], [16, 11], [16, 12], [16, 15], [16, 17], [17, 11], [17, 16], [18, 13], [19, 13]]`.

```
# # Simulation of the Schrodinger equation
#
#  This is a quantum simulation of the schrodinger equation for a free (V(x)=0) 1D particle in a
 ↪  4-point grid, using 2 qubits.

# In[11]:


# Import the QuantumProgram and our configuration
from math import pi, sqrt
from pprint import pprint
import time
import numpy as np
import qiskit

from qiskit import QuantumProgram #QuantumProgram is being deprecated
from qiskit import ClassicalRegister, QuantumRegister
from qiskit import QuantumCircuit,  available_backends, execute, register, get_backend, compile

# Import basic plotting tools
from qiskit.tools.visualization import plot_histogram, circuit_drawer, plot_state
from qiskit.tools.visualization import matplotlib_circuit_drawer as drawer, qx_color_scheme
get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('config', "InlineBackend.figure_format = 'svg'")
my_style = {'cregbundle': True, 'compress': True, 'usepiformat': True, 'latexdrawerstyle': False,
 ↪  'showindex': True}

# Import tomography tools
import qiskit.tools.qcvv.tomography as tomo

# Aditional packages
from qiskit.tools.qi.qi import *

# Compiler function

from qiskit.dagcircuit import DAGCircuit
import pyximportcpp; pyximportcpp.install()
import a_star_mapper_challenge
import pre_processing
import post_mapping_optimization

import copy
import sys, os, traceback

GLOBAL_TIMEOUT = 3600
ERROR_LIMIT = 1e-10

from qiskit.unroll import Unroller, DAGBackend
from qiskit._openquantumcompiler import dag2json
from multiprocessing import Pool
```

```
49  from qiskit.mapper._mappererror import MapperError
50
51
52  # Register token
53
54  try:
55      register('TOKEN',
56          "https://quantumexperience.ng.bluemix.net/api")
57      print('Available backends:\n')
58      print(available_backends({'simulator':False}))
59      print('Available simulators:')
60      print(available_backends({'simulator':True}))
61
62  except:
63      print('No valid token registered. Proceeding with available simulators.\n')
64      #print(available_backends())
65
66
67  # Set variables for the simulation
68  #Device
69  backend = 'ibmqx4'
70  #Characteristic phase shift
71  phi = 0
72  #Desired final state
73  idealvec = [0, 1/sqrt(2), 1/sqrt(2), 0]
74  #Mapping list: ibmqx4m or ibmq20m
75  mapping = [[1, 0], [2, 0], [2, 1], [3, 2], [3, 4], [4, 2]]
76
77
78  # ## Ideal simulation
79  #
80  # QISKit provides the option to use a local, classical simulator of a quantum device according to
     ↪  mathematical models. The results of the simulator should replicate those of an ideal quantum
     ↪  simulator, i.e. without decoherence or errors.
81
82  # In[25]:
83
84
85  #Define number of Qubits and bits of the circuit
86  qnum = 5
87  bnum = 2
88
89  #Qubit numbering scheme
90
91  q0 = 0
92  q1 = 1
93  q2 = 2
94
95  # Creating Programs
96  qp = QuantumProgram()
97  q = qp.create_quantum_register('q', qnum)
98  c = qp.create_classical_register('c', bnum)
99  qc = qp.create_circuit('Circuit', [q], [c])
100
101
102 #State preparation
103 qc.h(q[q0])
104 qc.x(q[q0])
105 qc.cx(q[q0], q[q1])
106 qc.x(q[q0])
107
108 #Direct fast fourier transform (QFT)
109 qc.h(q[q0])
110 qc.cu1(pi/2, q[q1], q[q0])
111 qc.h(q[q1])
112
```

```
#NOTE: Swapping gate (at the end of QFT)
#eliminated by simply changing qubit references

#Momentum centering
qc.x(q[q1])

#Phase transformations
qc.u1(2*phi, q[q1])
qc.u1(phi, q[q0])

qc.cx(q[q0], q[q2])
qc.cx(q[q1], q[q2])
qc.u1(2*phi, q[q2])
qc.cx(q[q1], q[q2])
qc.cx(q[q0], q[q2])

#Momentum (de)centering
qc.x(q[q1])

#Inverse QFT
qc.h(q[q1])
qc.cu1(-pi/2, q[q1], q[q0])
qc.h(q[q0])

#Measurement
#qc.measure(q[q0], c[0])
#qc.measure(q[q1], c[1])

#Get the qasm file
original_str = qp.get_qasm("Circuit")
#print(original_str)

#Draw the circuit
drawer(qc, style=my_style)


# In[19]:


#Using the state vector simulator, we can check if the algorithm produces the desired state

#Desired state (after delta t)
idealvec = [0, 1/sqrt(2), 1/sqrt(2), 0]

job_sv = execute(qc, backend='local_statevector_simulator')
statevector = job_sv.result().get_statevector(qc)

#The statevector describes the state of all 5 qubits; we can extract the 2-qubit state for simplicity
simvec = statevector[0:4]

#The Fidelity function can compare the desired state to the ideal output of the circuit:
F_fit = state_fidelity(simvec, idealvec)
print('Fidelity =', F_fit)


# In[26]:


#After state vector simulation, we can add the measurement gates:
qc.measure(q[q0], c[0])
qc.measure(q[q1], c[1])


# In[27]:


```

```
#Its possible to check the compiled circuit for the ideal simulator:
ideal_comp = compile(qc, backend='local_qasm_simulator');
ideal_qasm = qp.get_compiled_qasm(ideal_comp, 'Circuit');
ideal_circ = qiskit.load_qasm_string(ideal_qasm);

drawer(ideal_circ, style=my_style)


# In[29]:


#We can also run the simulation and check for the expected results:
job_ideal = execute(qc, 'local_qasm_simulator', shots=1000, max_credits=3)

lapse = 0
interval = 5
while not job_ideal.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(job_ideal.status)
    time.sleep(interval)
    lapse += 1
print(job_ideal.status)

print(job_ideal.result().get_counts(qc))
plot_histogram(job_ideal.result().get_counts(qc))


# ## Simulation using QISKit's optimization algorithms
#
#
# The simulation has to obey a predetermined gate set. For IBMQX4, it is composed of all gates
 ↪  belonging to SU(2), and the CNOT gate. The simulation also has to observe the specific coupling
 ↪  map for the quantum device. This coupling map determines which pairs of qubits can be used for the
 ↪  direct implementation of a CNOT gate. In the case of ibmqx4, the coupling map is:
#
# <img src="../images/ibmqx4-connections.png" alt="Note: In order for images to show up in this jupyter
 ↪  notebook you need to select File => Trusted Notebook" width="500 px" align="center">
#
#
# QISKit provides automated tools for the compiling of quantum algorithms into device-compliant
 ↪  circuits.

# In[31]:


#We can first check the circuit compiled for ibmqx4
qx4_comp = compile(qc, backend=backend);
qx4_qasm = qp.get_compiled_qasm(qx4_comp, 'Circuit');
qx4_circ = qiskit.load_qasm_string(qx4_qasm);

drawer(qx4_circ, style=my_style)


# In[33]:


#Checking device availability
backendx = get_backend(backend);
pprint(backendx.status)


# In[34]:


#Run results on ibmqx4
job_qx4 = execute(qc, 'ibmqx4', shots=1000, max_credits=3)
```

```
240
241 lapse = 0
242 interval = 30
243 while not job_qx4.done:
244     print('Status @ {} seconds'.format(interval * lapse))
245     print(job_qx4.status)
246     time.sleep(interval)
247     lapse += 1
248 print(job_qx4.status)
249
250 print(job_qx4.result().get_counts(qc))
251 plot_histogram(job_qx4.result().get_counts(qc))
252
253
254
255
256 # ## Running optimization algorithms
257 #
258 # Very recently, mapping algorithms have been developed which claim to have better efficiency than
     ↪  QISKit's. Such one is described in:
259 #
260 # Compiling SU(4) Quantum Circuits to IBM QX Architectures, by Zulehner, Alwin and Wille, Robert.
261 # http://iic.jku.at/files/eda/2018_arxiv_developer_challenge.pdf
262 #
263 # The optimization algorithm was run according to the provided tools, adapted for the simulation
     ↪  circuit. The circuit first has to be compiled into the gate set {u1, u2, u3, cx, id}
264
265 # In[40]:
266
267
268 #Optimization function
269
270 def qasm_to_dag_circuit(qasm_string, basis_gates='u1,u2,u3,cx,id'):
271     """
272     Convert an OPENQASM text string to a DAGCircuit.
273
274     Args:
275         qasm_string (str): OPENQASM2.0 circuit string.
276         basis_gates (str): QASM gates to unroll circuit to.
277
278     Returns:
279         A DAGCircuit object of the unrolled QASM circuit.
280     """
281     program_node_circuit = qiskit.qasm.Qasm(data=qasm_string).parse()
282     dag_circuit = Unroller(program_node_circuit,
283                            DAGBackend(basis_gates.split(","))).execute()
284     return dag_circuit
285
286
287
288 def compiler_function(dag_circuit, coupling_map=None, gate_costs=None):
289     """
290     Modify a DAGCircuit based on a gate cost function.
291
292     Instructions:
293         Your submission involves filling in the implementation
294         of this function. The function takes as input a DAGCircuit
295         object, which can be generated from a QASM file by using the
296         function 'qasm_to_dag_circuit' from the included
297         'submission_evaluation.py' module. For more information
298         on the DAGCircuit object see the or QISKit documentation
299         (eg. 'help(DAGCircuit)').
300
301     Args:
302         dag_circuit (DAGCircuit): DAGCircuit object to be compiled.
303         coupling_circuit (list): Coupling map for device topology.
```

```
                                A coupling map of None corresponds an
                                all-to-all connected topology.
        gate_costs (dict) : dictionary of gate names and costs.

    Returns:
        A modified DAGCircuit object that satisfies an input coupling_map
        and has as low a gate_cost as possible.
    """


    ######################
    # Put your code here
    ######################

    import copy
    from qiskit.mapper import Coupling, coupling_list2dict
    from qiskit import qasm, unroll
    import networkx as nx

    if gate_costs == None:
        gate_costs = {'id': 0, 'u1': 0, 'measure': 0, 'reset': 0, 'barrier': 0, 'u2': 1, 'u3': 1, 'U':
         ↪  1, 'cx': 10, 'CX': 10}

    compiled_dag = copy.deepcopy(dag_circuit)

    # temporary circuit to add all used gates to the available gate set
    tmp_qasm = "OPENQASM 2.0;\n" +                    "gate cx c,t { CX c,t; }\n" +
     ↪  "gate u3(theta,phi,lambda) q { U(theta,phi,lambda) q; }\n" +                    "gate
     ↪  u2(phi,lambda) q { U(pi/2,phi,lambda) q; }\n" +                    "gate u1(lambda) q {
     ↪  U(0,0,lambda) q; }\n" +                    "qreg q[2];\n" +                    "cx q[0],
     ↪  q[1];\n" +                    "u3(0.1,0.4,0.7) q[0];\n" +                    "u2(0.1,0.4)
     ↪  q[0]\n;" +                    "u1(0.1) q[0];\n"
    u = unroll.Unroller(qasm.Qasm(data=tmp_qasm).parse(),
                        unroll.DAGBackend(["cx", "u3", "u2", "u1"]))
    tmp_circuit = u.execute()

    # prepare empty circuit for the result
    empty_dag = DAGCircuit()

    coupling = Coupling(coupling_list2dict(mapping))
    empty_dag.add_qreg('q', coupling.size())

    for k, v in sorted(compiled_dag.cregs.items()):
        empty_dag.add_creg(k, v)

    empty_dag.basis = compiled_dag._make_union_basis(tmp_circuit)
    empty_dag.gates = compiled_dag._make_union_gates(tmp_circuit)

    # pre processing: group gates
    grouped_gates = pre_processing.group_gates(compiled_dag)

    # call mapper (based on an A* search) to satisfy the constraints for CNOTs given by the
     ↪  coupling_map
    compiled_dag = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
     ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
    grouped_gates_compiled = pre_processing.group_gates(compiled_dag)

    # estimate the cost of the mapped circuit: the number of groups as well as the cost regarding to
     ↪  gate_costs
    min_groups = grouped_gates_compiled.order()
    min_cost = 0
    for op, count in compiled_dag.count_ops().items():
        min_cost += count * gate_costs[op]

    # Repeat the mapping procedure 9 times and take the result with minimum groups/cost. Each call may
     ↪  yield a different result, since the mapper is implemented with a certain non-determinism. In
     ↪  fact, in the priority queue used for implementing the A* algorithm, the entries are a pair of
     ↪  the priority and a pointer to an object holding th mapping infomation (as second criterion).
     ↪  Thus, it is uncertain which node is expanded first if two nodes have the same priority (it
     ↪  depends on the value of the pointer). However, this non-determinism allows to find different
     ↪  solution by repeatedly calling the mapper.
```

```
    for i in range(9):
        result = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
         ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
        grouped_gates_result = pre_processing.group_gates(result)

        groups = grouped_gates_result.order()
        cost = 0
        for op, count in result.count_ops().items():
            cost += count * gate_costs[op]
        # take the solution with fewer groups (fewer cost if the number of groups is equal)
        if groups < min_groups or (groups == min_groups and cost < min_cost):
            min_groups = groups
            min_cost = cost
            compiled_dag = result
            grouped_gates_compiled = grouped_gates_result

    # post-mapping optimization: build 4x4 matrix for gate groups and decompose them using KAK
     ↪  decomposition.
    # Moreover, subsequent single qubit gates are optimized
    compiled_dag = post_mapping_optimization.optimize_gate_groups(grouped_gates_compiled,
     ↪  coupling.get_edges(), copy.deepcopy(empty_dag), gate_costs)

    return compiled_dag


# In[45]:



#Get the optimized circuit qasm, load it into a circuit, and visualize it
gateset_comp = compile(qc, backend='local_qasm_simulator', basis_gates='u1,u2,u3,cx,id');
gateset_str = qp.get_compiled_qasm(gateset_comp, 'Circuit');
opti_str=compiler_function(qasm_to_dag_circuit(gateset_str)).qasm();
opti_circ = qiskit.load_qasm_string(opti_str, name = 'Circuit');
drawer(opti_circ, style=my_style)


# In[50]:


#We can check if the compiler provides further optimization
opti_qx4_comp = compile(opti_circ, backend=backend);
opti_qx4_qasm = qp.get_compiled_qasm(opti_qx4_comp, 'Circuit');
opti_qx4_circ = qiskit.load_qasm_string(opti_qx4_qasm, name='Circuit');

drawer(opti_qx4_circ, style=my_style)


# In[52]:


#We now run the optimized circuit, and check the results
opti_qx4 = execute(opti_qx4_circ, backend=backend, shots=1000, max_credits=3)

lapse = 0
interval = 30
while not opti_qx4.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(opti_qx4.status)
    time.sleep(interval)
    lapse += 1
print(opti_qx4.status)

print(opti_qx4.result().get_counts(qc))
plot_histogram(opti_qx4.result().get_counts(qc))

```

```
423
424 # ## Device parameters
425
426 # In[53]:
427
428
429 backendx = get_backend(backend);
430 pprint(backendx.status)
431 pprint(backendx.configuration)
432 pprint(backendx.calibration)
433 pprint(backendx.parameters)
```

The script for quantum state tomography over the 2-qubit simulation follows.

```
# # Simulation of the Schrodinger equation
#
#  This is a quantum simulation of the schrodinger equation for a free (V(x)=0) 1D particle in a
 ↪  4-point grid, using 2 qubits.

# In[1]:


# Import the QuantumProgram and our configuration
from math import pi, sqrt
from pprint import pprint
import time
import numpy as np
import qiskit

from qiskit import QuantumProgram #QuantumProgram is being deprecated
from qiskit import ClassicalRegister, QuantumRegister
from qiskit import QuantumCircuit,  available_backends, execute, register, get_backend, compile

# Import basic plotting tools
from qiskit.tools.visualization import plot_histogram, circuit_drawer, plot_state
from qiskit.tools.visualization import matplotlib_circuit_drawer as drawer, qx_color_scheme
get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('config', "InlineBackend.figure_format = 'svg'")
my_style = {'cregbundle': True, 'compress': True, 'usepiformat': True, 'latexdrawerstyle': False,
 ↪  'showindex': True}

# Import tomography tools
import qiskit.tools.qcvv.tomography as tomo

# Aditional packages
from qiskit.tools.qi.qi import *

# Compiler function

from qiskit.dagcircuit import DAGCircuit
import pyximportcpp; pyximportcpp.install()
import a_star_mapper_challenge
import pre_processing
import post_mapping_optimization

import copy
import sys, os, traceback

GLOBAL_TIMEOUT = 3600
ERROR_LIMIT = 1e-10

from qiskit.unroll import Unroller, DAGBackend
from qiskit._openquantumcompiler import dag2json
from multiprocessing import Pool
from qiskit.mapper._mappererror import MapperError


# Register token

try:
    register('TOKEN',
        "https://quantumexperience.ng.bluemix.net/api")
    print('Available backends:\n')
    print(available_backends({'simulator':False}))
    print('Available simulators:')
    print(available_backends({'simulator':True}))

```

```
except:
    print('No valid token registered. Proceeding with available simulators.\n')
    #print(available_backends())


# Set variables for the simulation
#Device
backend = 'ibmqx4'
#Characteristic phase shift
phi = 0
#Desired final state
idealvec = [0, 1/sqrt(2), 1/sqrt(2), 0]
#Mapping list
mapping = [[1, 0], [2, 0], [2, 1], [3, 2], [3, 4], [4, 2]]

# ## Ideal simulation
#
# QISKit provides the option to use a local, classical simulator of a quantum device according to
 ↪  mathematical models. The results of the simulator should replicate those of an ideal quantum
 ↪  simulator, i.e. without decoherence or errors.

# In[2]:


#Define number of Qubits and bits of the circuit
qnum = 5
bnum = 2


#Qubit numbering scheme

q0 = 0
q1 = 1
q2 = 2

# Creating Programs
qp = QuantumProgram()
q = qp.create_quantum_register('q', qnum)
c = qp.create_classical_register('c', bnum)
qc = qp.create_circuit('Circuit', [q], [c])


#State preparation
qc.h(q[q0])
qc.x(q[q0])
qc.cx(q[q0], q[q1])
qc.x(q[q0])

#Direct fast fourier transform (QFT)
qc.h(q[q0])
qc.cu1(pi/2, q[q1], q[q0])
qc.h(q[q1])

#NOTE: Swapping gate (at the end of QFT)
# eliminated by simply changing qubit references

#Momentum centering
qc.x(q[q1])

#Phase transformations
qc.u1(2*phi, q[q1])
qc.u1(phi, q[q0])

qc.cx(q[q0], q[q2])
qc.cx(q[q1], q[q2])
qc.u1(2*phi, q[q2])
```

```
126 qc.cx(q[q1], q[q2])
127 qc.cx(q[q0], q[q2])
128
129 #Momentum (de)centering
130 qc.x(q[q1])
131
132 #Inverse QFT
133 qc.h(q[q1])
134 qc.cu1(-pi/2, q[q1], q[q0])
135 qc.h(q[q0])
136
137 #Measurement
138 #qc.measure(q[q0], c[0])
139 #qc.measure(q[q1], c[1])
140
141 #Get the qasm file
142 original_str = qp.get_qasm('Circuit')
143 #print(original_str)
144
145 #Draw the circuit
146 drawer(qc, style=my_style)
147
148
149 # In[3]:
150
151
152 #Using the state vector simulator, we can check if the algorithm produces the desired state
153
154 job_sv = execute(qc, backend='local_statevector_simulator')
155 statevector = job_sv.result().get_statevector(qc)
156
157 #The statevector describes the state of all 5 qubits; we can extract the 2-qubit state for simplicity
158 simvec = statevector[0:4]
159 print('State vector = ', simvec)
160
161
162 #The Fidelity function can compare the desired state to the ideal output of the circuit:
163 F_fit = state_fidelity(simvec, idealvec)
164 print('Fidelity =', F_fit)
165
166 #Create density matrix of desired state
167 ideal_rho = outer(simvec)
168 plot_state(ideal_rho)
169
170
171 # In[4]:
172
173
174 # Construct state tomography set for measurement of qubits [q0, q1] in the Pauli basis
175 qc_tomo_set = tomo.state_tomography_set([q0, q1])
176
177 # Add the state tomography measurement circuits to the Quantum Program
178 qc_tomo_circuit_names = tomo.create_tomography_circuits(qp, 'Circuit', q, c, qc_tomo_set)
179
180 circuit_list = [];
181
182 print('Created State tomography circuits:')
183 for name in qc_tomo_circuit_names:
184     circuit_list.append(qp.get_circuit(name))
185     print(name)
186
187
188 # In[6]:
189
190
191 # Test results on local simulator
```

```
backend = 'local_qasm_simulator'

# Define number of shots for each measurement basis
shots = 100

# Run the simulation

qc_tomo_job = execute(circuit_list, backend=backend, shots=shots, max_credits=3)

lapse = 0
interval = 1
while not qc_tomo_job.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(qc_tomo_job.status)
    time.sleep(interval)
    lapse += 1
print(qc_tomo_job.status)

qc_tomo_result = qc_tomo_job.result()
print(qc_tomo_result)

# Extract tomography data from results
qc_tomo_data = tomo.tomography_data(qc_tomo_result, 'Circuit', qc_tomo_set)

#Reconstruct the state from count data
rho_fit = tomo.fit_tomography_data(qc_tomo_data)

print('Vector = ', rho_fit)

# calculate fidelity, concurrence and purity of fitted state
F_fit = state_fidelity(rho_fit, simvec)
con = concurrence(rho_fit)
pur = purity(rho_fit)

# plot
plot_state(rho_fit,)
plot_state(rho_fit, 'paulivec')
print('Fidelity =', F_fit)
print('concurrence = ', str(con))
print('purity = ', str(pur))


# In[5]:


#Checking device availability
backendx = get_backend(backend);
pprint(backend.status)


# In[6]:

# Define number of shots for each measurement basis
shots = 100

# Run the simulation

qc_tomo_job_qx4 = execute(circuit_list, backend=backend, shots=shots, max_credits=3)

lapse = 0
interval = 30
while not qc_tomo_job_qx4.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(qc_tomo_job_qx4.status)
    time.sleep(interval)
    lapse += 1
```

```
258 print(qc_tomo_job_qx4.status)
259
260 qc_tomo_result_qx4 = qc_tomo_job_qx4.result()
261 print(qc_tomo_result_qx4)
262
263 # Extract tomography data from results
264 qc_tomo_data = tomo.tomography_data(qc_tomo_result_qx4, 'Circuit', qc_tomo_set)
265
266 #Reconstruct the state from count data
267 rho_fit = tomo.fit_tomography_data(qc_tomo_data)
268
269 print('Vector = ', rho_fit)
270
271 # calculate fidelity, concurrence and purity of fitted state
272 F_fit = state_fidelity(rho_fit, simvec)
273 con = concurrence(rho_fit)
274 pur = purity(rho_fit)
275
276 # plot
277 plot_state(rho_fit,)
278 plot_state(rho_fit, 'paulivec')
279 print('Fidelity =', F_fit)
280 print('concurrence = ', str(con))
281 print('purity = ', str(pur))
282
283
284
285 # # Running optimization algorithms
286
287 # In[7]:
288
289
290 #Optimization function
291
292 def qasm_to_dag_circuit(qasm_string, basis_gates='u1,u2,u3,cx,id'):
293     """
294     Convert an OPENQASM text string to a DAGCircuit.
295
296     Args:
297         qasm_string (str): OPENQASM2.0 circuit string.
298         basis_gates (str): QASM gates to unroll circuit to.
299
300     Returns:
301         A DAGCircuit object of the unrolled QASM circuit.
302     """
303     program_node_circuit = qiskit.qasm.Qasm(data=qasm_string).parse()
304     dag_circuit = Unroller(program_node_circuit,
305                            DAGBackend(basis_gates.split(","))).execute()
306     return dag_circuit
307
308
309
310 def compiler_function(dag_circuit, coupling_map=None, gate_costs=None):
311     """
312     Modify a DAGCircuit based on a gate cost function.
313
314     Instructions:
315         Your submission involves filling in the implementation
316         of this function. The function takes as input a DAGCircuit
317         object, which can be generated from a QASM file by using the
318         function 'qasm_to_dag_circuit' from the included
319         'submission_evaluation.py' module. For more information
320         on the DAGCircuit object see the or QISKit documentation
321         (eg. 'help(DAGCircuit)').
322
323     Args:
```

```
        dag_circuit (DAGCircuit): DAGCircuit object to be compiled.
        coupling_circuit (list): Coupling map for device topology.
                                 A coupling map of None corresponds an
                                 all-to-all connected topology.
        gate_costs (dict) : dictionary of gate names and costs.

    Returns:
        A modified DAGCircuit object that satisfies an input coupling_map
        and has as low a gate_cost as possible.
    """


    ######################
    # Put your code here
    ######################

    import copy
    from qiskit.mapper import Coupling, coupling_list2dict
    from qiskit import qasm, unroll
    import networkx as nx

    if gate_costs == None:
        gate_costs = {'id': 0, 'u1': 0, 'measure': 0, 'reset': 0, 'barrier': 0, 'u2': 1, 'u3': 1, 'U':
         ↪  1, 'cx': 10, 'CX': 10}

    compiled_dag = copy.deepcopy(dag_circuit)

    # temporary circuit to add all used gates to the available gate set
    tmp_qasm = "OPENQASM 2.0;\n" +                    "gate cx c,t { CX c,t; }\n" +
     ↪  "gate u3(theta,phi,lambda) q { U(theta,phi,lambda) q; }\n" +                     "gate
     ↪  u2(phi,lambda) q { U(pi/2,phi,lambda) q; }\n" +                  "gate u1(lambda) q {
     ↪  U(0,0,lambda) q; }\n" +                      "qreg q[2];\n" +                     "cx q[0],
     ↪  q[1];\n" +                    "u3(0.1,0.4,0.7) q[0];\n" +                    "u2(0.1,0.4)
     ↪  q[0]\n;" +                    "u1(0.1) q[0];\n"
    u = unroll.Unroller(qasm.Qasm(data=tmp_qasm).parse(),
                        unroll.DAGBackend(["cx", "u3", "u2", "u1"]))
    tmp_circuit = u.execute()

    # prepare empty circuit for the result
    empty_dag = DAGCircuit()

    coupling = Coupling(coupling_list2dict(mapping))
    empty_dag.add_qreg('q', coupling.size())

    for k, v in sorted(compiled_dag.cregs.items()):
        empty_dag.add_creg(k, v)

    empty_dag.basis = compiled_dag._make_union_basis(tmp_circuit)
    empty_dag.gates = compiled_dag._make_union_gates(tmp_circuit)

    # pre processing: group gates
    grouped_gates = pre_processing.group_gates(compiled_dag)

    # call mapper (based on an A* search) to satisfy the constraints for CNOTs given by the
     ↪  coupling_map
    compiled_dag = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
     ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
    grouped_gates_compiled = pre_processing.group_gates(compiled_dag)

    # estimate the cost of the mapped circuit: the number of groups as well as the cost regarding to
     ↪  gate_costs
    min_groups = grouped_gates_compiled.order()
    min_cost = 0
    for op, count in compiled_dag.count_ops().items():
        min_cost += count * gate_costs[op]

```

```
    # Repeat the mapping procedure 9 times and take the result with minimum groups/cost. Each call may
     ↪  yield a different result, since the mapper is implemented with a certain non-determinism. In
     ↪  fact, in the priority queue used for implementing the A* algorithm, the entries are a pair of
     ↪  the priority and a pointer to an object holding th mapping infomation (as second criterion).
     ↪  Thus, it is uncertain which node is expanded first if two nodes have the same priority (it
     ↪  depends on the value of the pointer). However, this non-determinism allows to find different
     ↪  solution by repeatedly calling the mapper.
    for i in range(9):
        result = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
         ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
        grouped_gates_result = pre_processing.group_gates(result)

        groups = grouped_gates_result.order()
        cost = 0
        for op, count in result.count_ops().items():
            cost += count * gate_costs[op]
        # take the solution with fewer groups (fewer cost if the number of groups is equal)
        if groups < min_groups or (groups == min_groups and cost < min_cost):
            min_groups = groups
            min_cost = cost
            compiled_dag = result
            grouped_gates_compiled = grouped_gates_result

    # post-mapping optimization: build 4x4 matrix for gate groups and decompose them using KAK
     ↪  decomposition.
    # Moreover, subsequent single qubit gates are optimized
    compiled_dag = post_mapping_optimization.optimize_gate_groups(grouped_gates_compiled,
     ↪  coupling.get_edges(), copy.deepcopy(empty_dag), gate_costs)

    return compiled_dag


# In[8]:


#Get the optimized circuit qasm, load it into a circuit, and visualize it
gateset_comp = compile(qc, backend='local_qasm_simulator', basis_gates='u1,u2,u3,cx,id');
gateset_str = qp.get_compiled_qasm(gateset_comp, 'Circuit');
opti_str=compiler_function(qasm_to_dag_circuit(gateset_str)).qasm();
opti_circ = qiskit.load_qasm_string(opti_str, name = 'Circuit');
drawer(opti_circ, style=my_style)


# In[9]:


# Construct state tomography set for measurement of qubits [q0, q1] in the Pauli basis
qc_tomo_set = tomo.state_tomography_set([q0, q1])

# Add the state tomography measurement circuits to the Quantum Program
qc_tomo_circuit_names = tomo.create_tomography_circuits(qp, 'Circuit', q, c, qc_tomo_set)

circuit_list = [];

print('Created State tomography circuits:')
for name in qc_tomo_circuit_names:
    circuit_list.append(qp.get_circuit(name))
    print(name)


# In[10]:


# Define number of shots for each measurement basis
shots = 100

```

```
# Run the simulation

qc_tomo_job_qx4 = execute(circuit_list, backend=backend, shots=shots, max_credits=3)

lapse = 0
interval = 30
while not qc_tomo_job_qx4.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(qc_tomo_job_qx4.status)
    time.sleep(interval)
    lapse += 1
print(qc_tomo_job_qx4.status)

qc_tomo_result_qx4 = qc_tomo_job_qx4.result()
print(qc_tomo_result_qx4)

# Extract tomography data from results
qc_tomo_data = tomo.tomography_data(qc_tomo_result_qx4, 'Circuit', qc_tomo_set)

#Reconstruct the state from count data
rho_fit = tomo.fit_tomography_data(qc_tomo_data)

print('Vector = ', rho_fit)

# calculate fidelity, concurrence and purity of fitted state
F_fit = state_fidelity(rho_fit, simvec)
con = concurrence(rho_fit)
pur = purity(rho_fit)

# plot
plot_state(rho_fit,)
plot_state(rho_fit, 'paulivec')
print('Fidelity =', F_fit)
print('concurrence = ', str(con))
print('purity = ', str(pur))


# ## Device parameters

# In[12]:


backendx = get_backend(backend);
pprint(backendx.status)
pprint(backendx.configuration)
pprint(backendx.calibration)
pprint(backendx.parameters)
```

B.2 3-qubit algorithms

To change between devices, the string variable `backend` should be set to either `'ibmqx4'` or `'ibmq_20_tokyo'`. For the alternative compiler, the mapping has to be set explicitly.

For *ibmqx4*, `mapping = [[1, 0], [2, 0], [2, 1], [3, 2], [3, 4], [4, 2]]`.

For *ibmq20*, `mapping = [[0, 1], [0, 5], [1, 0], [1, 2], [1, 6], [1, 7], [2, 1], [2, 3], [2, 6], [3, 2], [3, 8], [3, 9], [4, 8], [4, 9], [5, 0], [5, 6], [5, 10], [5, 11], [6, 1], [6, 2], [6, 5], [6, 7], [6, 10], [6, 11], [7, 1], [7, 6], [7,`

8], [7, 12], [7, 13], [8, 3], [8, 4], [8, 7], [8, 9], [8, 12], [8, 13], [9, 3], [9, 4], [9, 8], [10, 5], [10, 6], [10, 11], [10, 15], [11, 5], [11, 6], [11, 10], [11, 12], [11, 16], [11, 17], [12, 7], [12, 8], [12, 11], [12, 13], [12, 16], [13, 7], [13, 8], [13, 12], [13, 14], [13, 18], [13, 19], [14, 13], [15, 10], [15, 16], [16, 11], [16, 12], [16, 15], [16, 17], [17, 11], [17, 16], [18, 13], [19, 13]].

```
# coding: utf-8

# # Simulation of the Schrödinger equation
#
#  This is a quantum simulation of the schrodinger equation for a free (V(x)=0) 1D particle in a 4-point grid, using 2 qubits.

# In[1]:


# Import the QuantumProgram and our configuration
from math import pi, sqrt
from pprint import pprint
import time
import numpy as np
import qiskit

from qiskit import QuantumProgram #QuantumProgram is being deprecated
from qiskit import ClassicalRegister, QuantumRegister
from qiskit import QuantumCircuit,  available_backends, execute, register, get_backend, compile

# Import basic plotting tools
from qiskit.tools.visualization import plot_histogram, circuit_drawer, plot_state
from qiskit.tools.visualization import matplotlib_circuit_drawer as drawer, qx_color_scheme
get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('config', "InlineBackend.figure_format = 'svg'")
my_style = {'cregbundle': True, 'compress': True, 'usepiformat': True, 'latexdrawerstyle': False, 'showindex': True}

# Import tomography tools
import qiskit.tools.qcvv.tomography as tomo

# Aditional packages
from qiskit.tools.qi.qi import *

# Compiler function

from qiskit.dagcircuit import DAGCircuit
import pyximportcpp; pyximportcpp.install()
import a_star_mapper_challenge
import pre_processing
import post_mapping_optimization

import copy
import sys, os, traceback

GLOBAL_TIMEOUT = 3600
ERROR_LIMIT = 1e-10

from qiskit.unroll import Unroller, DAGBackend
from qiskit._openquantumcompiler import dag2json
from multiprocessing import Pool
from qiskit.mapper._mappererror import MapperError


```

```
54  # Register token
55
56  try:
57      register('TOKEN',
58          "https://quantumexperience.ng.bluemix.net/api")
59      print('Available backends:\n')
60      print(available_backends({'simulator':False}))
61      print('Available simulators:')
62      print(available_backends({'simulator':True}))
63
64  except:
65      print('No valid token registered. Proceeding with available simulators.\n')
66      print(available_backends())
67
68
69  # Set variables for the simulation
70  #Device
71  backend = 'ibmqx4'
72  #Characteristic phase shift
73  phi = 0
74  #Desired final state
75  idealvec = [0, 1/sqrt(2), 1/sqrt(2), 0]
76  #Mapping list
77  mapping = [[1, 0], [2, 0], [2, 1], [3, 2], [3, 4], [4, 2]]
78
79
80  # ## Ideal simulation
81  #
82  # QISKit provides the option to use a local, classical simulator of a quantum device according to
     ↪  mathematical models. The results of the simulator should replicate those of an ideal quantum
     ↪  simulator, i.e. without decoherence or errors.
83
84  # In[22]:
85
86
87  #Define number of Qubits and bits of the circuit
88  qnum = 5
89  bnum = 3
90
91  #Qubit numbering scheme
92
93  q0 = 0
94  q1 = 1
95  q2 = 2
96  q3 = 3 #ancilla qubit
97
98  # Creating Programs
99  qp = QuantumProgram()
100 q = qp.create_quantum_register('q', qnum)
101 c = qp.create_classical_register('c', bnum)
102 qc = qp.create_circuit('Circuit', [q], [c])
103
104
105 #State preparation
106 psi0 = [0, 0, 1/2, 1/2, 1/2, 1/2, 0, 0]
107 qc.initialize(psi0, [q[q0],q[q1],q[q2]])
108
109 #Direct fast fourier transform (QFT)
110 qc.h(q[q0])
111 qc.cu1(pi/(2**(1)), q[q0], q[q1]);
112 qc.cu1(pi/(2**(2)), q[q0], q[q2]);
113 qc.h(q[q1])
114 qc.cu1(pi/(2**(1)), q[q1], q[q2]);
115 qc.h(q[q2])
116
117
```

```
#NOTE: Swapping gate (at the end of QFT)
# eliminated by simply changing qubit references

#Momentum centering
qc.x(q[q2])


#Phase transformations
qc.u1(phi/4, q[q0])
qc.u1(phi/2, q[q1])
qc.u1(phi, q[q2])

qc.cx(q[q2], q[q3])
qc.cx(q[q1], q[q3])
qc.u1(2*phi, q[q3])
qc.cx(q[q1], q[q3])
qc.cx(q[q2], q[q3])

qc.cx(q[q2], q[q3])
qc.cx(q[q0], q[q3])
qc.u1(phi, q[q3])
qc.cx(q[q0], q[q3])
qc.cx(q[q2], q[q3])

qc.cx(q[q1], q[q3])
qc.cx(q[q0], q[q3])
qc.u1(phi/2, q[q3])
qc.cx(q[q0], q[q3])
qc.cx(q[q1], q[q3])

#Momentum (de)centering
qc.x(q[q2])

#Inverse QFT (swapped input)
qc.h(q[q2])
qc.cu1(-pi/(2**(1)), q[q1], q[q2]);
qc.h(q[q1])
qc.cu1(-pi/(2**(2)), q[q0], q[q2]);
qc.cu1(-pi/(2**(1)), q[q0], q[q1]);
qc.h(q[q0])



#Measurement
#qc.measure(q[q0], c[0])
#qc.measure(q[q1], c[1])
#qc.measure(q[q2], c[2])

#Draw the circuit
drawer(qc, style=my_style, scale=0.6)
#print(qc.qasm())


# In[23]:


#Using the state vector simulator, we can check if the algorithm produces the desired state

#Desired state (after delta t)
idealvec = [0, 0, 1/2, 1/2, 1/2, 1/2, 0, 0]

job_sv = execute(qc, backend='local_statevector_simulator')
statevector = job_sv.result().get_statevector(qc)

#The statevector describes the state of all 5 qubits; we can extract the 2-qubit state for simplicity
simvec = statevector[0:8]
```

```
184 print('State vector = ', simvec)
185
186
187 #The Fidelity function can compare the desired state to the ideal output of the circuit:
188 F_fit = state_fidelity(simvec, idealvec)
189 print('Fidelity =', F_fit)
190
191 #Create density matrix of desired state
192 ideal_rho = outer(simvec)
193 plot_state(ideal_rho)
194
195
196 # In[24]:
197
198
199 # Construct state tomography set for measurement of qubits [q0, q1] in the Pauli basis
200 qc_tomo_set = tomo.state_tomography_set([q0, q1, q2])
201
202 qp.add_circuit('Circuit', qc)
203
204 # Add the state tomography measurement circuits to the Quantum Program
205 qc_tomo_circuit_names = tomo.create_tomography_circuits(qp, 'Circuit', q, c, qc_tomo_set)
206
207 circuit_list = [];
208
209 print('Created State tomography circuits:')
210 for name in qc_tomo_circuit_names:
211     circuit_list.append(qp.get_circuit(name))
212     print(name)
213
214 drawer(circuit_list[0], style=my_style)
215
216
217 # In[7]:
218
219
220 # Check tomography circuit after compilation
221 qx4_comp = compile(qp.get_circuit('Circuit_meas_X(0)X(1)X(2)'), backend=backend);
222 qp.get_execution_list(qx4_comp)
223 qx4_qasm = qp.get_compiled_qasm(qx4_comp, 'circuit3');
224 qx4_circ = qiskit.load_qasm_string(qx4_qasm);
225
226 drawer(qx4_circ, style=my_style)
227
228
229 # In[25]:
230
231
232 # Test results on local simulator
233 backend = 'local_qasm_simulator'
234
235 # Define number of shots for each measurement basis
236 shots = 100
237
238 # Run the simulation
239
240 qc_tomo_job = execute(circuit_list, backend=backend, shots=shots, max_credits=3)
241
242 lapse = 0
243 interval = 1
244 while not qc_tomo_job.done:
245     print('Status @ {} seconds'.format(interval * lapse))
246     print(qc_tomo_job.status)
247     time.sleep(interval)
248     lapse += 1
249 print(qc_tomo_job.status)
```

```
250
251 qc_tomo_result = qc_tomo_job.result()
252 print(qc_tomo_result)
253
254 # Extract tomography data from results
255 qc_tomo_data = tomo.tomography_data(qc_tomo_result, 'Circuit', qc_tomo_set)
256
257 #Reconstruct the state from count data
258 rho_fit = tomo.fit_tomography_data(qc_tomo_data)
259
260 print('Matrix = ', rho_fit)
261
262 # calculate fidelity, concurrence and purity of fitted state
263 F_fit = state_fidelity(rho_fit, simvec)
264 pur = purity(rho_fit)
265
266 # plot
267 plot_state(rho_fit,)
268 plot_state(rho_fit, 'paulivec')
269 print('Fidelity =', F_fit)
270 print('purity = ', str(pur))
271
272
273 # # Simulation using QISKit's optimization algortithms
274
275 # In[26]:
276
277
278 backendx = get_backend(backend);
279 pprint(backendx.status)
280
281
282 # In[27]:
283
284
285 # Define number of shots for each measurement basis
286 shots = 100
287
288 # Run the simulation
289
290 qc_tomo_job_qx4 = execute(circuit_list, backend=backend, shots=shots, max_credits=3)
291
292 lapse = 0
293 interval = 30
294 while not qc_tomo_job_qx4.done:
295     print('Status @ {} seconds'.format(interval * lapse))
296     print(qc_tomo_job_qx4.status)
297     time.sleep(interval)
298     lapse += 1
299 print(qc_tomo_job_qx4.status)
300
301 qc_tomo_result_qx4 = qc_tomo_job_qx4.result()
302 print(qc_tomo_result_qx4)
303
304 # Extract tomography data from results
305 qc_tomo_data = tomo.tomography_data(qc_tomo_result_qx4, 'Circuit', qc_tomo_set)
306
307 #Reconstruct the state from count data
308 rho_fit = tomo.fit_tomography_data(qc_tomo_data)
309
310 print('Matrix = ', rho_fit)
311
312 # calculate fidelity, concurrence and purity of fitted state
313 F_fit = state_fidelity(rho_fit, simvec)
314 pur = purity(rho_fit)
315
```

```
316 # plot
317 plot_state(rho_fit,)
318 plot_state(rho_fit, 'paulivec')
319 print('Fidelity =', F_fit)
320 print('purity = ', str(pur))
321
322
323 # In[ ]:
324
325
326 # calculate fidelity, concurrence and purity of fitted state
327 F_fit = state_fidelity(rho_fit, simvec)
328 pur = purity(rho_fit)
329
330 # plot
331 plot_state(rho_fit,)
332 plot_state(rho_fit, 'paulivec')
333 print('Fidelity =', F_fit)
334 print('purity = ', str(pur))
335
336
337 # ## Running optimization algorithms
338 #
339 # Very recently, mapping algorithms have been developed which claim to have better efficiency than
     ↪  QISKit's. Such one is described in:
340 #
341 # Compiling SU(4) Quantum Circuits to IBM QX Architectures, by Zulehner, Alwin and Wille, Robert.
342 # http://iic.jku.at/files/eda/2018_arxiv_developer_challenge.pdf
343 #
344 # The optimization algorithm was run according to the provided tools, adapted for the simulation
     ↪  circuit. The circuit first has to be compiled into the gate set {u1, u2, u3, cx, id}
345
346 # In[17]:
347
348
349 #Optimization function
350
351 def qasm_to_dag_circuit(qasm_string, basis_gates='u1,u2,u3,cx,id'):
352     """
353     Convert an OPENQASM text string to a DAGCircuit.
354
355     Args:
356         qasm_string (str): OPENQASM2.0 circuit string.
357         basis_gates (str): QASM gates to unroll circuit to.
358
359     Returns:
360         A DAGCircuit object of the unrolled QASM circuit.
361     """
362     program_node_circuit = qiskit.qasm.Qasm(data=qasm_string).parse()
363     dag_circuit = Unroller(program_node_circuit,
364                            DAGBackend(basis_gates.split(","))).execute()
365     return dag_circuit
366
367
368
369 def compiler_function(dag_circuit, coupling_map=None, gate_costs=None):
370     """
371     Modify a DAGCircuit based on a gate cost function.
372
373     Instructions:
374         Your submission involves filling in the implementation
375         of this function. The function takes as input a DAGCircuit
376         object, which can be generated from a QASM file by using the
377         function 'qasm_to_dag_circuit' from the included
378         'submission_evaluation.py' module. For more information
379         on the DAGCircuit object see the or QISKit documentation
```

```
380         (eg. 'help(DAGCircuit)').
381
382     Args:
383         dag_circuit (DAGCircuit): DAGCircuit object to be compiled.
384         coupling_circuit (list): Coupling map for device topology.
385                                  A coupling map of None corresponds an
386                                  all-to-all connected topology.
387         gate_costs (dict) : dictionary of gate names and costs.
388
389     Returns:
390         A modified DAGCircuit object that satisfies an input coupling_map
391         and has as low a gate_cost as possible.
392     """
393
394
395     ######################
396     # Put your code here
397     ######################
398
399     import copy
400     from qiskit.mapper import Coupling, coupling_list2dict
401     from qiskit import qasm, unroll
402     import networkx as nx
403
404     if gate_costs == None:
405         gate_costs = {'id': 0, 'u1': 0, 'measure': 0, 'reset': 0, 'barrier': 0, 'u2': 1, 'u3': 1, 'U':
          ↪  1, 'cx': 10, 'CX': 10}
406
407     compiled_dag = copy.deepcopy(dag_circuit)
408
409     # temporary circuit to add all used gates to the available gate set
410     tmp_qasm = "OPENQASM 2.0;\n" +                    "gate cx c,t { CX c,t; }\n" +
         ↪  "gate u3(theta,phi,lambda) q { U(theta,phi,lambda) q; }\n" +                    "gate
         ↪  u2(phi,lambda) q { U(pi/2,phi,lambda) q; }\n" +                  "gate u1(lambda) q {
         ↪  U(0,0,lambda) q; }\n" +                    "qreg q[2];\n" +                    "cx q[0],
         ↪  q[1];\n" +                  "u3(0.1,0.4,0.7) q[0];\n" +                   "u2(0.1,0.4)
         ↪  q[0]\n;" +                  "u1(0.1) q[0];\n"
411     u = unroll.Unroller(qasm.Qasm(data=tmp_qasm).parse(),
412                         unroll.DAGBackend(["cx", "u3", "u2", "u1"]))
413     tmp_circuit = u.execute()
414
415     # prepare empty circuit for the result
416     empty_dag = DAGCircuit()
417
418     coupling = Coupling(coupling_list2dict(mapping))
419     empty_dag.add_qreg('q', coupling.size())
420
421     for k, v in sorted(compiled_dag.cregs.items()):
422         empty_dag.add_creg(k, v)
423
424     empty_dag.basis = compiled_dag._make_union_basis(tmp_circuit)
425     empty_dag.gates = compiled_dag._make_union_gates(tmp_circuit)
426
427     # pre processing: group gates
428     grouped_gates = pre_processing.group_gates(compiled_dag)
429
430     # call mapper (based on an A* search) to satisfy the constraints for CNOTs given by the
         ↪  coupling_map
431     compiled_dag = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
         ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
432     grouped_gates_compiled = pre_processing.group_gates(compiled_dag)
433
434     # estimate the cost of the mapped circuit: the number of groups as well as the cost regarding to
         ↪  gate_costs
435     min_groups = grouped_gates_compiled.order()
436     min_cost = 0
```

```
437     for op, count in compiled_dag.count_ops().items():
438         min_cost += count * gate_costs[op]
439 
440     # Repeat the mapping procedure 9 times and take the result with minimum groups/cost. Each call may
        ↪  yield a different result, since the mapper is implemented with a certain non-determinism. In
        ↪  fact, in the priority queue used for implementing the A* algorithm, the entries are a pair of
        ↪  the priority and a pointer to an object holding th mapping infomation (as second criterion).
        ↪  Thus, it is uncertain which node is expanded first if two nodes have the same priority (it
        ↪  depends on the value of the pointer). However, this non-determinism allows to find different
        ↪  solution by repeatedly calling the mapper.
441     for i in range(9):
442         result = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
            ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
443         grouped_gates_result = pre_processing.group_gates(result)
444 
445         groups = grouped_gates_result.order()
446         cost = 0
447         for op, count in result.count_ops().items():
448             cost += count * gate_costs[op]
449         # take the solution with fewer groups (fewer cost if the number of groups is equal)
450         if groups < min_groups or (groups == min_groups and cost < min_cost):
451             min_groups = groups
452             min_cost = cost
453             compiled_dag = result
454             grouped_gates_compiled = grouped_gates_result
455 
456     # post-mapping optimization: build 4x4 matrix for gate groups and decompose them using KAK
        ↪  decomposition.
457     # Moreover, subsequent single qubit gates are optimized
458     compiled_dag = post_mapping_optimization.optimize_gate_groups(grouped_gates_compiled,
        ↪  coupling.get_edges(), copy.deepcopy(empty_dag), gate_costs)
459 
460     return compiled_dag
461 
462 
463 # In[18]:
464 
465 #Get the optimized circuit qasm, load it into a circuit, and visualize it
466 gateset_comp = compile(qc, backend='local_qasm_simulator', basis_gates='u1,u2,u3,cx,id');
467 gateset_str = qp.get_compiled_qasm(gateset_comp, 'Circuit');
468 opti_str=compiler_function(qasm_to_dag_circuit(gateset_str)).qasm();
469 opti_circ = qiskit.load_qasm_string(opti_str, name = 'Circuit');
470 drawer(opti_circ, style=my_style)
471 
472 
473 # In[19]:
474 
475 
476 #Using the state vector simulator, we can check if the algorithm produces the desired state
477 
478 #Desired state (after delta t, phi=0)
479 idealvec = [0, 0, 1/2, 1/2, 1/2, 1/2, 0, 0]
480 
481 job_sv = execute(opti_circ, backend='local_statevector_simulator')
482 statevector = job_sv.result().get_statevector(opti_circ)
483 
484 #The statevector describes the state of all 5 qubits; we can extract the 2-qubit state for simplicity
485 simvec = statevector[0:8]
486 print('State vector = ', simvec)
487 
488 
489 #The Fidelity function can compare the desired state to the ideal output of the circuit:
490 F_fit = state_fidelity(simvec, idealvec)
491 print('Fidelity =', F_fit)
492 
493 #Create density matrix of desired state
```

```
ideal_rho = outer(simvec)
plot_state(ideal_rho)


# In[20]:


# Construct state tomography set for measurement of qubits [q0, q1] in the Pauli basis
qc_tomo_set = tomo.state_tomography_set([q0, q1, q2])


# Reset quantum program
# Creating Programs
qp = QuantumProgram()
q = qp.create_quantum_register('q', qnum)
c = qp.create_classical_register('c', bnum)
qc = qp.create_circuit('Circuit', [q], [c])


qp.add_circuit('Circuit', opti_circ)

# Add the state tomography measurement circuits to the Quantum Program
qc_tomo_circuit_names = tomo.create_tomography_circuits(qp, 'Circuit', q, c, qc_tomo_set)

circuit_list = [];

print('Created State tomography circuits:')
for name in qc_tomo_circuit_names:
    circuit_list.append(qp.get_circuit(name))
    print(name)

drawer(circuit_list[0], style=my_style)


# In[21]:


# Define number of shots for each measurement basis
shots = 100

# Run the simulation

qc_tomo_job_qx4 = execute(circuit_list, backend=backend, shots=shots, max_credits=3)

lapse = 0
interval = 30
while not qc_tomo_job_qx4.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(qc_tomo_job_qx4.status)
    time.sleep(interval)
    lapse += 1
print(qc_tomo_job_qx4.status)

qc_tomo_result_qx4 = qc_tomo_job_qx4.result()
print(qc_tomo_result_qx4)

# Extract tomography data from results
qc_tomo_data = tomo.tomography_data(qc_tomo_result_qx4, 'Circuit', qc_tomo_set)

#Reconstruct the state from count data
rho_fit = tomo.fit_tomography_data(qc_tomo_data)

print('Vector = ', rho_fit)

# calculate fidelity, concurrence and purity of fitted state
F_fit = state_fidelity(rho_fit, simvec)
```

```
pur = purity(rho_fit)

# plot
plot_state(rho_fit,)
plot_state(rho_fit, 'paulivec')
print('Fidelity =', F_fit)
print('purity = ', str(pur))


# ## Device parameters

# In[28]:


backendx = get_backend(backend);
pprint(backendx.status)
pprint(backendx.configuration)
pprint(backendx.calibration)
pprint(backendx.parameters)
```

The script for quantum state tomography over the 3-qubit simulation follows.

```
# coding: utf-8

# # Simulation of the Schrödinger equation
#
#  This is a quantum simulation of the schrodinger equation for a free (V(x)=0) 1D particle in a 4-point grid, using 2 qubits.

# In[1]:


# Import the QuantumProgram and our configuration
from math import pi, sqrt
from pprint import pprint
import time
import numpy as np
import qiskit

from qiskit import QuantumProgram #QuantumProgram is being deprecated
from qiskit import ClassicalRegister, QuantumRegister
from qiskit import QuantumCircuit,  available_backends, execute, register, get_backend, compile

# Import basic plotting tools
from qiskit.tools.visualization import plot_histogram, circuit_drawer, plot_state
from qiskit.tools.visualization import matplotlib_circuit_drawer as drawer, qx_color_scheme
get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('config', "InlineBackend.figure_format = 'svg'")
my_style = {'cregbundle': True, 'compress': True, 'usepiformat': True, 'latexdrawerstyle': False, 'showindex': True}

# Import tomography tools
import qiskit.tools.qcvv.tomography as tomo

# Aditional packages
from qiskit.tools.qi.qi import *

# Compiler function

from qiskit.dagcircuit import DAGCircuit
import pyximportcpp; pyximportcpp.install()
import a_star_mapper_challenge
import pre_processing
import post_mapping_optimization

import copy
import sys, os, traceback

GLOBAL_TIMEOUT = 3600
ERROR_LIMIT = 1e-10

from qiskit.unroll import Unroller, DAGBackend
from qiskit._openquantumcompiler import dag2json
from multiprocessing import Pool
from qiskit.mapper._mappererror import MapperError


# Register token

try:
    register('TOKEN',
        "https://quantumexperience.ng.bluemix.net/api")
    print('Available backends:\n')
    print(available_backends({'simulator':False}))
    print('Available simulators:')
```

```
62      print(available_backends({'simulator':True}))
63
64  except:
65      print('No valid token registered. Proceeding with available simulators.\n')
66      print(available_backends())
67
68
69  # Set variables for the simulation
70  #Device
71  backend = 'ibmqx4'
72  #Characteristic phase shift
73  phi = 0
74  #Desired final state
75  idealvec = [0, 1/sqrt(2), 1/sqrt(2), 0]
76  #Mapping list
77  mapping = [[1, 0], [2, 0], [2, 1], [3, 2], [3, 4], [4, 2]]
78
79
80  # ## Ideal simulation
81  #
82  # QISKit provides the option to use a local, classical simulator of a quantum device according to
     ↪  mathematical models. The results of the simulator should replicate those of an ideal quantum
     ↪  simulator, i.e. without decoherence or errors.
83
84  # In[22]:
85
86
87  #Define number of Qubits and bits of the circuit
88  qnum = 5
89  bnum = 3
90
91  #Qubit numbering scheme
92
93  q0 = 0
94  q1 = 1
95  q2 = 2
96  q3 = 3 #ancilla qubit
97
98  # Creating Programs
99  qp = QuantumProgram()
100 q = qp.create_quantum_register('q', qnum)
101 c = qp.create_classical_register('c', bnum)
102 qc = qp.create_circuit('Circuit', [q], [c])
103
104
105 #State preparation
106 psi0 = [0, 0, 1/2, 1/2, 1/2, 1/2, 0, 0]
107 qc.initialize(psi0, [q[q0],q[q1],q[q2]])
108
109 #Direct fast fourier transform (QFT)
110 qc.h(q[q0])
111 qc.cu1(pi/(2**(1)), q[q0], q[q1]);
112 qc.cu1(pi/(2**(2)), q[q0], q[q2]);
113 qc.h(q[q1])
114 qc.cu1(pi/(2**(1)), q[q1], q[q2]);
115 qc.h(q[q2])
116
117
118 #NOTE: Swapping gate (at the end of QFT)
119 # eliminated by simply changing qubit references
120
121 #Momentum centering
122 qc.x(q[q2])
123
124
125 #Phase transformations
```

```
126 qc.u1(phi/4, q[q0])
127 qc.u1(phi/2, q[q1])
128 qc.u1(phi, q[q2])
129
130 qc.cx(q[q2], q[q3])
131 qc.cx(q[q1], q[q3])
132 qc.u1(2*phi, q[q3])
133 qc.cx(q[q1], q[q3])
134 qc.cx(q[q2], q[q3])
135
136 qc.cx(q[q2], q[q3])
137 qc.cx(q[q0], q[q3])
138 qc.u1(phi, q[q3])
139 qc.cx(q[q0], q[q3])
140 qc.cx(q[q2], q[q3])
141
142 qc.cx(q[q1], q[q3])
143 qc.cx(q[q0], q[q3])
144 qc.u1(phi/2, q[q3])
145 qc.cx(q[q0], q[q3])
146 qc.cx(q[q1], q[q3])
147
148 #Momentum (de)centering
149 qc.x(q[q2])
150
151 #Inverse QFT (swapped input)
152 qc.h(q[q2])
153 qc.cu1(-pi/(2**(1)), q[q1], q[q2]);
154 qc.h(q[q1])
155 qc.cu1(-pi/(2**(2)), q[q0], q[q2]);
156 qc.cu1(-pi/(2**(1)), q[q0], q[q1]);
157 qc.h(q[q0])
158
159
160
161 #Measurement
162 #qc.measure(q[q0], c[0])
163 #qc.measure(q[q1], c[1])
164 #qc.measure(q[q2], c[2])
165
166 #Draw the circuit
167 drawer(qc, style=my_style, scale=0.6)
168 #print(qc.qasm())
169
170
171 # In[23]:
172
173
174 #Using the state vector simulator, we can check if the algorithm produces the desired state
175
176 #Desired state (after delta t)
177 idealvec = [0, 0, 1/2, 1/2, 1/2, 1/2, 0, 0]
178
179 job_sv = execute(qc, backend='local_statevector_simulator')
180 statevector = job_sv.result().get_statevector(qc)
181
182 #The statevector describes the state of all 5 qubits; we can extract the 2-qubit state for simplicity
183 simvec = statevector[0:8]
184 print('State vector = ', simvec)
185
186
187 #The Fidelity function can compare the desired state to the ideal output of the circuit:
188 F_fit = state_fidelity(simvec, idealvec)
189 print('Fidelity =', F_fit)
190
191 #Create density matrix of desired state
```

```
ideal_rho = outer(simvec)
plot_state(ideal_rho)


# In[24]:


# Construct state tomography set for measurement of qubits [q0, q1] in the Pauli basis
qc_tomo_set = tomo.state_tomography_set([q0, q1, q2])

qp.add_circuit('Circuit', qc)

# Add the state tomography measurement circuits to the Quantum Program
qc_tomo_circuit_names = tomo.create_tomography_circuits(qp, 'Circuit', q, c, qc_tomo_set)

circuit_list = [];

print('Created State tomography circuits:')
for name in qc_tomo_circuit_names:
    circuit_list.append(qp.get_circuit(name))
    print(name)

drawer(circuit_list[0], style=my_style)


# In[7]:


# Check tomography circuit after compilation
qx4_comp = compile(qp.get_circuit('Circuit_meas_X(0)X(1)X(2)'), backend=backend);
qp.get_execution_list(qx4_comp)
qx4_qasm = qp.get_compiled_qasm(qx4_comp, 'circuit3');
qx4_circ = qiskit.load_qasm_string(qx4_qasm);

drawer(qx4_circ, style=my_style)


# In[25]:


# Test results on local simulator
backend = 'local_qasm_simulator'

# Define number of shots for each measurement basis
shots = 100

# Run the simulation

qc_tomo_job = execute(circuit_list, backend=backend, shots=shots, max_credits=3)

lapse = 0
interval = 1
while not qc_tomo_job.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(qc_tomo_job.status)
    time.sleep(interval)
    lapse += 1
print(qc_tomo_job.status)

qc_tomo_result = qc_tomo_job.result()
print(qc_tomo_result)

# Extract tomography data from results
qc_tomo_data = tomo.tomography_data(qc_tomo_result, 'Circuit', qc_tomo_set)

#Reconstruct the state from count data
```

```
rho_fit = tomo.fit_tomography_data(qc_tomo_data)

print('Matrix = ', rho_fit)

# calculate fidelity, concurrence and purity of fitted state
F_fit = state_fidelity(rho_fit, simvec)
pur = purity(rho_fit)

# plot
plot_state(rho_fit,)
plot_state(rho_fit, 'paulivec')
print('Fidelity =', F_fit)
print('purity = ', str(pur))


# # Simulation using QISKit's optimization algortithms

# In[26]:


backendx = get_backend(backend);
pprint(backendx.status)


# In[27]:


# Define number of shots for each measurement basis
shots = 100

# Run the simulation

qc_tomo_job_qx4 = execute(circuit_list, backend=backend, shots=shots, max_credits=3)

lapse = 0
interval = 30
while not qc_tomo_job_qx4.done:
    print('Status @ {} seconds'.format(interval * lapse))
    print(qc_tomo_job_qx4.status)
    time.sleep(interval)
    lapse += 1
print(qc_tomo_job_qx4.status)

qc_tomo_result_qx4 = qc_tomo_job_qx4.result()
print(qc_tomo_result_qx4)

# Extract tomography data from results
qc_tomo_data = tomo.tomography_data(qc_tomo_result_qx4, 'Circuit', qc_tomo_set)

#Reconstruct the state from count data
rho_fit = tomo.fit_tomography_data(qc_tomo_data)

print('Matrix = ', rho_fit)

# calculate fidelity, concurrence and purity of fitted state
F_fit = state_fidelity(rho_fit, simvec)
pur = purity(rho_fit)

# plot
plot_state(rho_fit,)
plot_state(rho_fit, 'paulivec')
print('Fidelity =', F_fit)
print('purity = ', str(pur))


# In[ ]:
```

```
324
325
326 # calculate fidelity, concurrence and purity of fitted state
327 F_fit = state_fidelity(rho_fit, simvec)
328 pur = purity(rho_fit)
329
330 # plot
331 plot_state(rho_fit,)
332 plot_state(rho_fit, 'paulivec')
333 print('Fidelity =', F_fit)
334 print('purity = ', str(pur))
335
336
337 # ## Running optimization algorithms
338 #
339 # Very recently, mapping algorithms have been developed which claim to have better efficiency than
     ↪  QISKit's. Such one is described in:
340 #
341 # Compiling SU(4) Quantum Circuits to IBM QX Architectures, by Zulehner, Alwin and Wille, Robert.
342 # http://iic.jku.at/files/eda/2018_arxiv_developer_challenge.pdf
343 #
344 # The optimization algorithm was run according to the provided tools, adapted for the simulation
     ↪  circuit. The circuit first has to be compiled into the gate set {u1, u2, u3, cx, id}
345
346 # In[17]:
347
348
349 #Optimization function
350
351 def qasm_to_dag_circuit(qasm_string, basis_gates='u1,u2,u3,cx,id'):
352     """
353     Convert an OPENQASM text string to a DAGCircuit.
354
355     Args:
356         qasm_string (str): OPENQASM2.0 circuit string.
357         basis_gates (str): QASM gates to unroll circuit to.
358
359     Returns:
360         A DAGCircuit object of the unrolled QASM circuit.
361     """
362     program_node_circuit = qiskit.qasm.Qasm(data=qasm_string).parse()
363     dag_circuit = Unroller(program_node_circuit,
364                            DAGBackend(basis_gates.split(","))).execute()
365     return dag_circuit
366
367
368
369 def compiler_function(dag_circuit, coupling_map=None, gate_costs=None):
370     """
371     Modify a DAGCircuit based on a gate cost function.
372
373     Instructions:
374         Your submission involves filling in the implementation
375         of this function. The function takes as input a DAGCircuit
376         object, which can be generated from a QASM file by using the
377         function 'qasm_to_dag_circuit' from the included
378         'submission_evaluation.py' module. For more information
379         on the DAGCircuit object see the or QISKit documentation
380         (eg. 'help(DAGCircuit)').
381
382     Args:
383         dag_circuit (DAGCircuit): DAGCircuit object to be compiled.
384         coupling_circuit (list): Coupling map for device topology.
385                                  A coupling map of None corresponds an
386                                  all-to-all connected topology.
387         gate_costs (dict) : dictionary of gate names and costs.
```

```
388
389     Returns:
390         A modified DAGCircuit object that satisfies an input coupling_map
391         and has as low a gate_cost as possible.
392     """
393
394
395     ######################
396     # Put your code here
397     ######################
398
399     import copy
400     from qiskit.mapper import Coupling, coupling_list2dict
401     from qiskit import qasm, unroll
402     import networkx as nx
403
404     if gate_costs == None:
405         gate_costs = {'id': 0, 'u1': 0, 'measure': 0, 'reset': 0, 'barrier': 0, 'u2': 1, 'u3': 1, 'U':
            ↪  1, 'cx': 10, 'CX': 10}
406
407     compiled_dag = copy.deepcopy(dag_circuit)
408
409     # temporary circuit to add all used gates to the available gate set
410     tmp_qasm = "OPENQASM 2.0;\n" +                    "gate cx c,t { CX c,t; }\n" +
        ↪  "gate u3(theta,phi,lambda) q { U(theta,phi,lambda) q; }\n" +                    "gate
        ↪  u2(phi,lambda) q { U(pi/2,phi,lambda) q; }\n" +                  "gate u1(lambda) q {
        ↪  U(0,0,lambda) q; }\n" +                    "qreg q[2];\n" +                    "cx q[0],
        ↪  q[1];\n" +                   "u3(0.1,0.4,0.7) q[0];\n" +                   "u2(0.1,0.4)
        ↪  q[0]\n;" +                   "u1(0.1) q[0];\n"
411     u = unroll.Unroller(qasm.Qasm(data=tmp_qasm).parse(),
412                         unroll.DAGBackend(["cx", "u3", "u2", "u1"]))
413     tmp_circuit = u.execute()
414
415     # prepare empty circuit for the result
416     empty_dag = DAGCircuit()
417
418     coupling = Coupling(coupling_list2dict(mapping))
419     empty_dag.add_qreg('q', coupling.size())
420
421     for k, v in sorted(compiled_dag.cregs.items()):
422         empty_dag.add_creg(k, v)
423
424     empty_dag.basis = compiled_dag._make_union_basis(tmp_circuit)
425     empty_dag.gates = compiled_dag._make_union_gates(tmp_circuit)
426
427     # pre processing: group gates
428     grouped_gates = pre_processing.group_gates(compiled_dag)
429
430     # call mapper (based on an A* search) to satisfy the constraints for CNOTs given by the
        ↪  coupling_map
431     compiled_dag = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
        ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
432     grouped_gates_compiled = pre_processing.group_gates(compiled_dag)
433
434     # estimate the cost of the mapped circuit: the number of groups as well as the cost regarding to
        ↪  gate_costs
435     min_groups = grouped_gates_compiled.order()
436     min_cost = 0
437     for op, count in compiled_dag.count_ops().items():
438         min_cost += count * gate_costs[op]
439
440     # Repeat the mapping procedure 9 times and take the result with minimum groups/cost. Each call may
        ↪  yield a different result, since the mapper is implemented with a certain non-determinism. In
        ↪  fact, in the priority queue used for implementing the A* algorithm, the entries are a pair of
        ↪  the priority and a pointer to an object holding th mapping infomation (as second criterion).
        ↪  Thus, it is uncertain which node is expanded first if two nodes have the same priority (it
        ↪  depends on the value of the pointer). However, this non-determinism allows to find different
        ↪  solution by repeatedly calling the mapper.
```

```
    for i in range(9):
        result = a_star_mapper_challenge.a_star_mapper(copy.deepcopy(grouped_gates),
         ↪  coupling_list2dict(mapping), coupling.size(), copy.deepcopy(empty_dag))
        grouped_gates_result = pre_processing.group_gates(result)

        groups = grouped_gates_result.order()
        cost = 0
        for op, count in result.count_ops().items():
            cost += count * gate_costs[op]
        # take the solution with fewer groups (fewer cost if the number of groups is equal)
        if groups < min_groups or (groups == min_groups and cost < min_cost):
            min_groups = groups
            min_cost = cost
            compiled_dag = result
            grouped_gates_compiled = grouped_gates_result

    # post-mapping optimization: build 4x4 matrix for gate groups and decompose them using KAK
     ↪  decomposition.
    # Moreover, subsequent single qubit gates are optimized
    compiled_dag = post_mapping_optimization.optimize_gate_groups(grouped_gates_compiled,
     ↪  coupling.get_edges(), copy.deepcopy(empty_dag), gate_costs)

    return compiled_dag


# In[18]:

#Get the optimized circuit qasm, load it into a circuit, and visualize it
gateset_comp = compile(qc, backend='local_qasm_simulator', basis_gates='u1,u2,u3,cx,id');
gateset_str = qp.get_compiled_qasm(gateset_comp, 'Circuit');
opti_str=compiler_function(qasm_to_dag_circuit(gateset_str)).qasm();
opti_circ = qiskit.load_qasm_string(opti_str, name = 'Circuit');
drawer(opti_circ, style=my_style)


# In[19]:


#Using the state vector simulator, we can check if the algorithm produces the desired state

#Desired state (after delta t, phi=0)
idealvec = [0, 0, 1/2, 1/2, 1/2, 1/2, 0, 0]

job_sv = execute(opti_circ, backend='local_statevector_simulator')
statevector = job_sv.result().get_statevector(opti_circ)

#The statevector describes the state of all 5 qubits; we can extract the 2-qubit state for simplicity
simvec = statevector[0:8]
print('State vector = ', simvec)


#The Fidelity function can compare the desired state to the ideal output of the circuit:
F_fit = state_fidelity(simvec, idealvec)
print('Fidelity =', F_fit)

#Create density matrix of desired state
ideal_rho = outer(simvec)
plot_state(ideal_rho)


# In[20]:


# Construct state tomography set for measurement of qubits [q0, q1] in the Pauli basis
qc_tomo_set = tomo.state_tomography_set([q0, q1, q2])

```

```
504
505 # Reset quantum program
506 # Creating Programs
507 qp = QuantumProgram()
508 q = qp.create_quantum_register('q', qnum)
509 c = qp.create_classical_register('c', bnum)
510 qc = qp.create_circuit('Circuit', [q], [c])
511
512
513 qp.add_circuit('Circuit', opti_circ)
514
515 # Add the state tomography measurement circuits to the Quantum Program
516 qc_tomo_circuit_names = tomo.create_tomography_circuits(qp, 'Circuit', q, c, qc_tomo_set)
517
518 circuit_list = [];
519
520 print('Created State tomography circuits:')
521 for name in qc_tomo_circuit_names:
522     circuit_list.append(qp.get_circuit(name))
523     print(name)
524
525 drawer(circuit_list[0], style=my_style)
526
527
528 # In[21]:
529
530
531 # Define number of shots for each measurement basis
532 shots = 100
533
534 # Run the simulation
535
536 qc_tomo_job_qx4 = execute(circuit_list, backend=backend, shots=shots, max_credits=3)
537
538 lapse = 0
539 interval = 30
540 while not qc_tomo_job_qx4.done:
541     print('Status @ {} seconds'.format(interval * lapse))
542     print(qc_tomo_job_qx4.status)
543     time.sleep(interval)
544     lapse += 1
545 print(qc_tomo_job_qx4.status)
546
547 qc_tomo_result_qx4 = qc_tomo_job_qx4.result()
548 print(qc_tomo_result_qx4)
549
550 # Extract tomography data from results
551 qc_tomo_data = tomo.tomography_data(qc_tomo_result_qx4, 'Circuit', qc_tomo_set)
552
553 #Reconstruct the state from count data
554 rho_fit = tomo.fit_tomography_data(qc_tomo_data)
555
556 print('Vector = ', rho_fit)
557
558 # calculate fidelity, concurrence and purity of fitted state
559 F_fit = state_fidelity(rho_fit, simvec)
560 pur = purity(rho_fit)
561
562 # plot
563 plot_state(rho_fit,)
564 plot_state(rho_fit, 'paulivec')
565 print('Fidelity =', F_fit)
566 print('purity = ', str(pur))
567
568
569 # ## Device parameters
```

```
570
571 # In[28]:
572
573
574 backendx = get_backend(backend);
575 pprint(backendx.status)
576 pprint(backendx.configuration)
577 pprint(backendx.calibration)
578 pprint(backendx.parameters)
```

C

QISKIT RESULTS

C.1 DEVICE PARAMETERS

The raw device parameters were obtained from the execution of the last cell of code, for each of the scripts in section B.

ibmqx4 - Tenerife

```
{'pending_jobs': 15, 'name': 'ibmq_5_tenerife', 'operational': True}
{'allow_q_object': False,
'basis_gates': 'u1,u2,u3,cx,id',
'chip_name': 'Raven',
'coupling_map': [[1, 0], [2, 0], [2, 1], [3, 2], [3, 4], [4, 2]],
'deleted': False,
'description': '5 qubit transmon bowtie chip 3',
'gate_set': 'SU2+CNOT',
'internal_id': '5ae875670f020500393162b3',
'local': False,
'n_qubits': 5,
'name': 'ibmq_5_tenerife',
'online_date': '2017-09-18T00:00:00.000Z',
'simulator': False,
'url': 'https://ibm.biz/qiskit-ibmqx4',
'version': '1.2.0'}
{'backend': 'ibmq_5_tenerife',
'last_update_date': '2018-09-18T09:56:34.000Z',
'multi_qubit_gates': [{'gateError': {'date': '2018-09-18T09:56:34Z',
                                    'value': 0.03139759925594232},
        'name': 'CX1_0',
        'qubits': [1, 0],
        'type': 'CX'},
   {'gateError': {'date': '2018-09-18T09:56:34Z',
                                    'value': 0.022638367856000235},
        'name': 'CX2_0',
        'qubits': [2, 0],
        'type': 'CX'},
   {'gateError': {'date': '2018-09-18T09:56:34Z',
                                    'value': 0.041650384934039136},
        'name': 'CX2_1',
        'qubits': [2, 1],
        'type': 'CX'},
   {'gateError': {'date': '2018-09-18T09:56:34Z',
                                    'value': 0.08569671400725232},
```

```
        'name': 'CX3_2',
        'qubits': [3, 2],
        'type': 'CX'},
   {'gateError': {'date': '2018-09-18T09:56:34Z',
                                'value': 0.04158178794909698},
        'name': 'CX3_4',
        'qubits': [3, 4],
        'type': 'CX'},
   {'gateError': {'date': '2018-09-18T09:56:34Z',
                                'value': 0.04942339516006977},
        'name': 'CX4_2',
        'qubits': [4, 2],
        'type': 'CX'}],
 'qubits': [{'gateError': {'date': '2018-09-18T09:56:34Z',
            'value': 0.0010302181416348977},
 'name': 'Q0',
 'readoutError': {'date': '2018-09-18T09:56:34Z', 'value': 0.064}},
{'gateError': {'date': '2018-09-18T09:56:34Z',
            'value': 0.003091708656797032},
 'name': 'Q1',
 'readoutError': {'date': '2018-09-18T09:56:34Z', 'value': 0.064}},
{'gateError': {'date': '2018-09-18T09:56:34Z',
            'value': 0.0011160854878761173},
 'name': 'Q2',
 'readoutError': {'date': '2018-09-18T09:56:34Z', 'value': 0.026}},
{'gateError': {'date': '2018-09-18T09:56:34Z',
            'value': 0.001803112096824766},
 'name': 'Q3',
 'readoutError': {'date': '2018-09-18T09:56:34Z', 'value': 0.025}},
{'gateError': {'date': '2018-09-18T09:56:34Z',
            'value': 0.001545458810288558},
 'name': 'Q4',
 'readoutError': {'date': '2018-09-18T09:56:34Z', 'value': 0.065}}]}
{'backend': 'ibmq_5_tenerife',
 'fridge_parameters': {'Temperature': {'date': '-', 'unit': '-', 'value': []},
    'cooldownDate': '2017-09-07'},
 'last_update_date': '2018-09-18T09:56:34.000Z',
 'qubits': [{'T1': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 50.9},
 'T2': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 40.8},
 'buffer': {'date': '2018-09-18T09:56:34Z',
        'unit': 'ns',
        'value': 10},
 'frequency': {'date': '2018-09-18T09:56:34Z',
            'unit': 'GHz',
            'value': 5.24984},
 'gateTime': {'date': '2018-09-18T09:56:34Z',
           'unit': 'ns',
           'value': 60},
 'name': 'Q0'},
{'T1': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 49.8},
 'T2': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 17.1},
 'buffer': {'date': '2018-09-18T09:56:34Z',
        'unit': 'ns',
        'value': 10},
 'frequency': {'date': '2018-09-18T09:56:34Z',
            'unit': 'GHz',
            'value': 5.29578},
 'gateTime': {'date': '2018-09-18T09:56:34Z',
```

```
          'unit': 'ns',
          'value': 60},
 'name': 'Q1'},
{'T1': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 49.2},
 'T2': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 37.3},
 'buffer': {'date': '2018-09-18T09:56:34Z',
        'unit': 'ns',
        'value': 10},
 'frequency': {'date': '2018-09-18T09:56:34Z',
           'unit': 'GHz',
           'value': 5.35321},
 'gateTime': {'date': '2018-09-18T09:56:34Z',
          'unit': 'ns',
          'value': 60},
 'name': 'Q2'},
{'T1': {'date': '2018-09-18T09:56:34Z', 'unit': 'µs', 'value': 53},
 'T2': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 17.7},
 'buffer': {'date': '2018-09-18T09:56:34Z',
        'unit': 'ns',
        'value': 10},
 'frequency': {'date': '2018-09-18T09:56:34Z',
           'unit': 'GHz',
           'value': 5.43497},
 'gateTime': {'date': '2018-09-18T09:56:34Z',
          'unit': 'ns',
          'value': 60},
 'name': 'Q3'},
{'T1': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 46.1},
 'T2': {'date': '2018-09-18T09:56:34Z',
 'unit': 'µs',
 'value': 11.1},
 'buffer': {'date': '2018-09-18T09:56:34Z',
        'unit': 'ns',
        'value': 10},
 'frequency': {'date': '2018-09-18T09:56:34Z',
           'unit': 'GHz',
           'value': 5.17582},
 'gateTime': {'date': '2018-09-18T09:56:34Z',
          'unit': 'ns',
          'value': 60},
 'name': 'Q4'}]}
```


ibmq20 - Tokyo


```
{'pending_jobs': -1, 'name': 'ibmq_20_tokyo', 'operational': True}
{'allow_q_object': False,
 'basis_gates': 'u1,u2,u3,cx,id',
 'chip_name': 'Qubert',
 'coupling_map': [[0, 1],
   [0, 5],
   [1, 0],
```

```
    [1, 2],
    [1, 6],
    [1, 7],
    [2, 1],
    [2, 3],
    [2, 6],
    [3, 2],
    [3, 8],
    [3, 9],
    [4, 8],
    [4, 9],
    [5, 0],
    [5, 6],
    [5, 10],
    [5, 11],
    [6, 1],
    [6, 2],
    [6, 5],
    [6, 7],
    [6, 10],
    [6, 11],
    [7, 1],
    [7, 6],
    [7, 8],
    [7, 12],
    [7, 13],
    [8, 3],
    [8, 4],
    [8, 7],
    [8, 9],
    [8, 12],
    [8, 13],
    [9, 3],
    [9, 4],
    [9, 8],
    [10, 5],
    [10, 6],
    [10, 11],
    [10, 15],
    [11, 5],
    [11, 6],
    [11, 10],
    [11, 12],
    [11, 16],
    [11, 17],
    [12, 7],
    [12, 8],
    [12, 11],
    [12, 13],
    [12, 16],
    [13, 7],
    [13, 8],
    [13, 12],
    [13, 14],
    [13, 18],
    [13, 19],
    [14, 13],
    [15, 10],
    [15, 16],
    [16, 11],
    [16, 12],
    [16, 15],
    [16, 17],
    [17, 11],
    [17, 16],
    [18, 13],
```

```
  [19, 13]],
 'description': '20 qubit device Tokyo',
 'gate_set': 'SU2+CNOT',
 'local': False,
 'n_qubits': 20,
 'name': 'ibmq_20_tokyo',
 'online_date': '2018-05-10T00:00:00.000Z',
 'simulator': False,
 'url': 'None',
 'version': '1'}
{'backend': 'ibmq_20_tokyo',
 'last_update_date': '2018-09-20T04:04:36.000Z',
 'multi_qubit_gates': [{'gateError': {'date': '2018-09-20T01:31:11Z',
                                      'value': 0.03746604186552352},
                'name': 'CX0_1',
                'qubits': [0, 1],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T01:35:35Z',
                                      'value': 0.039147142673431334},
                'name': 'CX0_5',
                'qubits': [0, 5],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T01:31:11Z',
                                      'value': 0.03746604186552352},
                'name': 'CX1_0',
                'qubits': [1, 0],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T01:44:44Z',
                                      'value': 0.01871490928609615},
                'name': 'CX1_2',
                'qubits': [1, 2],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T01:40:01Z',
                                      'value': 0.042803396283799394},
                'name': 'CX1_6',
                'qubits': [1, 6],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T02:17:06Z',
                                      'value': 0.03028888326801335},
                'name': 'CX1_7',
                'qubits': [1, 7],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T01:44:44Z',
                                      'value': 0.01871490928609615},
                'name': 'CX2_1',
                'qubits': [2, 1],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T01:49:08Z',
                                      'value': 0.028262063629773237},
                'name': 'CX2_3',
                'qubits': [2, 3],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T02:07:56Z',
                                      'value': 0.03110789820736304},
                'name': 'CX2_6',
                'qubits': [2, 6],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T01:49:08Z',
                                      'value': 0.028262063629773237},
                'name': 'CX3_2',
                'qubits': [3, 2],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T02:35:43Z',
                                      'value': 0.038716648346250854},
                'name': 'CX3_8',
                'qubits': [3, 8],
```

```
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:50:08Z',
                                   'value': 0.024967535547755604},
                'name': 'CX3_9',
                'qubits': [3, 9],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T01:53:35Z',
                                   'value': 0.01883545014443025},
                'name': 'CX4_8',
                'qubits': [4, 8],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:54:32Z',
                                   'value': 0.014304302256721302},
                'name': 'CX4_9',
                'qubits': [4, 9],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T01:35:35Z',
                                   'value': 0.039147142673431334},
                'name': 'CX5_0',
                'qubits': [5, 0],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T01:58:04Z',
                                   'value': 0.044379251681177095},
                'name': 'CX5_6',
                'qubits': [5, 6],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:59:17Z',
                                   'value': 0.018922477678219668},
                'name': 'CX5_10',
                'qubits': [5, 10],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:03:09Z',
                                   'value': 0.015035509397939212},
                'name': 'CX5_11',
                'qubits': [5, 11],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T01:40:01Z',
                                   'value': 0.042803396283799394},
                'name': 'CX6_1',
                'qubits': [6, 1],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:07:56Z',
                                   'value': 0.03110789820736304},
                'name': 'CX6_2',
                'qubits': [6, 2],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T01:58:04Z',
                                   'value': 0.044379251681177095},
                'name': 'CX6_5',
                'qubits': [6, 5],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:21:30Z',
                                   'value': 0.028482857829085106},
                'name': 'CX6_7',
                'qubits': [6, 7],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:03:43Z',
                                   'value': 0.017131627529771293},
                'name': 'CX6_10',
                'qubits': [6, 10],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:12:42Z',
                                   'value': 0.01615766101357008},
                'name': 'CX6_11',
                'qubits': [6, 11],
                'type': 'CX'},
```

```
206             {'gateError': {'date': '2018-09-20T02:17:06Z',
207                                    'value': 0.03028888326801335},
208              'name': 'CX7_1',
209              'qubits': [7, 1],
210              'type': 'CX'},
211             {'gateError': {'date': '2018-09-20T02:21:30Z',
212                                    'value': 0.028482857829085106},
213              'name': 'CX7_6',
214              'qubits': [7, 6],
215              'type': 'CX'},
216             {'gateError': {'date': '2018-09-20T02:31:17Z',
217                                    'value': 0.019257533255399778},
218              'name': 'CX7_8',
219              'qubits': [7, 8],
220              'type': 'CX'},
221             {'gateError': {'date': '2018-09-20T02:26:52Z',
222                                    'value': 0.019179485415230096},
223              'name': 'CX7_12',
224              'qubits': [7, 12],
225              'type': 'CX'},
226             {'gateError': {'date': '2018-09-20T03:37:51Z',
227                                    'value': 0.03684749138915164},
228              'name': 'CX7_13',
229              'qubits': [7, 13],
230              'type': 'CX'},
231             {'gateError': {'date': '2018-09-20T02:35:43Z',
232                                    'value': 0.038716648346250854},
233              'name': 'CX8_3',
234              'qubits': [8, 3],
235              'type': 'CX'},
236             {'gateError': {'date': '2018-09-20T01:53:35Z',
237                                    'value': 0.01883545014443025},
238              'name': 'CX8_4',
239              'qubits': [8, 4],
240              'type': 'CX'},
241             {'gateError': {'date': '2018-09-20T02:31:17Z',
242                                    'value': 0.019257533255399778},
243              'name': 'CX8_7',
244              'qubits': [8, 7],
245              'type': 'CX'},
246             {'gateError': {'date': '2018-09-20T02:40:12Z',
247                                    'value': 0.052583034333770595},
248              'name': 'CX8_9',
249              'qubits': [8, 9],
250              'type': 'CX'},
251             {'gateError': {'date': '2018-09-20T02:45:46Z',
252                                    'value': 0.08501285782708357},
253              'name': 'CX8_12',
254              'qubits': [8, 12],
255              'type': 'CX'},
256             {'gateError': {'date': '2018-09-20T03:42:34Z',
257                                    'value': 0.018538232112185332},
258              'name': 'CX8_13',
259              'qubits': [8, 13],
260              'type': 'CX'},
261             {'gateError': {'date': '2018-09-20T02:50:08Z',
262                                    'value': 0.024967535547755604},
263              'name': 'CX9_3',
264              'qubits': [9, 3],
265              'type': 'CX'},
266             {'gateError': {'date': '2018-09-20T02:54:32Z',
267                                    'value': 0.014304302256721302},
268              'name': 'CX9_4',
269              'qubits': [9, 4],
270              'type': 'CX'},
271             {'gateError': {'date': '2018-09-20T02:40:12Z',
```

```
                                        'value': 0.052583034333770595},
             'name': 'CX9_8',
             'qubits': [9, 8],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:59:17Z',
                                        'value': 0.018922477678219668},
             'name': 'CX10_5',
             'qubits': [10, 5],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:03:43Z',
                                        'value': 0.017131627529771293},
             'name': 'CX10_6',
             'qubits': [10, 6],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:12:33Z',
                                        'value': 0.03967657481091055},
             'name': 'CX10_11',
             'qubits': [10, 11],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:08:06Z',
                                        'value': 0.022865430819129895},
             'name': 'CX10_15',
             'qubits': [10, 15],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:03:09Z',
                                        'value': 0.015035509397939212},
             'name': 'CX11_5',
             'qubits': [11, 5],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:12:42Z',
                                        'value': 0.01615766101357008},
             'name': 'CX11_6',
             'qubits': [11, 6],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:12:33Z',
                                        'value': 0.03967657481091055},
             'name': 'CX11_10',
             'qubits': [11, 10],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:16:58Z',
                                        'value': 0.03430588126330322},
             'name': 'CX11_12',
             'qubits': [11, 12],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:21:25Z',
                                        'value': 0.016329362868310077},
             'name': 'CX11_16',
             'qubits': [11, 16],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:25:50Z',
                                        'value': 0.029505149636642664},
             'name': 'CX11_17',
             'qubits': [11, 17],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T02:26:52Z',
                                        'value': 0.019179485415230096},
             'name': 'CX12_7',
             'qubits': [12, 7],
             'type': 'CX'},
            {'gateError': {'date': '2018-08-23T13:52:35Z',
                                        'value': 0.1224402730634041},
             'name': 'CX12_8',
             'qubits': [12, 8],
             'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:16:58Z',
                                        'value': 0.03430588126330322},
```

```
                'name': 'CX12_11',
                'qubits': [12, 11],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:30:17Z',
                                   'value': 0.030815164528812028},
                'name': 'CX12_13',
                'qubits': [12, 13],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-19T17:32:37Z',
                                   'value': 0.032560723669238734},
                'name': 'CX12_16',
                'qubits': [12, 16],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:37:51Z',
                                   'value': 0.03684749138915164},
                'name': 'CX13_7',
                'qubits': [13, 7],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:42:34Z',
                                   'value': 0.018538232112185332},
                'name': 'CX13_8',
                'qubits': [13, 8],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:30:17Z',
                                   'value': 0.030815164528812028},
                'name': 'CX13_12',
                'qubits': [13, 12],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:47:01Z',
                                   'value': 0.028453741315738634},
                'name': 'CX13_14',
                'qubits': [13, 14],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:51:27Z',
                                   'value': 0.025184827360353296},
                'name': 'CX13_18',
                'qubits': [13, 18],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T04:04:36Z',
                                   'value': 0.0466878447771896},
                'name': 'CX13_19',
                'qubits': [13, 19],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:47:01Z',
                                   'value': 0.028453741315738634},
                'name': 'CX14_13',
                'qubits': [14, 13],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:08:06Z',
                                   'value': 0.022865430819129895},
                'name': 'CX15_10',
                'qubits': [15, 10],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:55:57Z',
                                   'value': 0.015578993080560988},
                'name': 'CX15_16',
                'qubits': [15, 16],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-20T03:21:25Z',
                                   'value': 0.016329362868310077},
                'name': 'CX16_11',
                'qubits': [16, 11],
                'type': 'CX'},
            {'gateError': {'date': '2018-09-19T17:32:37Z',
                                   'value': 0.032560723669238734},
                'name': 'CX16_12',
```

```
                'qubits': [16, 12],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T03:55:57Z',
                                  'value': 0.015578993080560988},
                'name': 'CX16_15',
                'qubits': [16, 15],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T04:00:24Z',
                                  'value': 0.023981683549585908},
                'name': 'CX16_17',
                'qubits': [16, 17],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T03:25:50Z',
                                  'value': 0.029505149636642664},
                'name': 'CX17_11',
                'qubits': [17, 11],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T04:00:24Z',
                                  'value': 0.023981683549585908},
                'name': 'CX17_16',
                'qubits': [17, 16],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T03:51:27Z',
                                  'value': 0.025184827360353296},
                'name': 'CX18_13',
                'qubits': [18, 13],
                'type': 'CX'},
           {'gateError': {'date': '2018-09-20T04:04:36Z',
                                  'value': 0.0466878447771896},
                'name': 'CX19_13',
                'qubits': [19, 13],
                'type': 'CX'}],
 'qubits': [{'gateError': {'date': '2018-09-20T01:02:15Z',
                           'value': 0.0006991930204756636},
             'name': 'Q0',
             'readoutError': {'date': '2018-09-19T20:55:52',
                              'value': 0.05600000000000005}},
            {'gateError': {'date': '2018-09-20T01:03:22Z',
                           'value': 0.00078899914883145721},
             'name': 'Q1',
             'readoutError': {'date': '2018-09-19T20:55:52',
                              'value': 0.04699999999999993}},
            {'gateError': {'date': '2018-09-20T01:02:15Z',
                           'value': 0.0009886300613452526},
             'name': 'Q2',
             'readoutError': {'date': '2018-09-19T20:55:52',
                              'value': 0.05899999999999994}},
            {'gateError': {'date': '2018-09-20T01:03:22Z',
                           'value': 0.0025731445216121696},
             'name': 'Q3',
             'readoutError': {'date': '2018-09-19T20:55:52',
                              'value': 0.08899999999999997}},
            {'gateError': {'date': '2018-09-20T01:02:15Z',
                           'value': 0.002036104022737961},
             'name': 'Q4',
             'readoutError': {'date': '2018-09-19T20:55:52',
                              'value': 0.04699999999999993}},
            {'gateError': {'date': '2018-09-20T01:03:22Z',
                           'value': 0.0006173096953862034},
             'name': 'Q5',
             'readoutError': {'date': '2018-09-19T20:55:52',
                              'value': 0.02200000000000002}},
            {'gateError': {'date': '2018-09-20T01:04:32Z',
                           'value': 0.0007059812261480114},
             'name': 'Q6',
             'readoutError': {'date': '2018-09-19T20:55:52',
```

```
                           'value': 0.03600000000000003}},
 {'gateError': {'date': '2018-09-20T01:05:42Z',
                        'value': 0.0008735386069325668},
  'name': 'Q7',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.09000000000000008}},
 {'gateError': {'date': '2018-09-20T01:04:32Z',
                        'value': 0.0006520883818727508},
  'name': 'Q8',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.03600000000000003}},
 {'gateError': {'date': '2018-09-20T01:05:42Z',
                        'value': 0.0009465413727616223},
  'name': 'Q9',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.02300000000000002}},
 {'gateError': {'date': '2018-09-20T01:01:10Z',
                        'value': 0.0010770656544301094},
  'name': 'Q10',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.02600000000000023}},
 {'gateError': {'date': '2018-09-20T01:01:10Z',
                        'value': 0.0007899349344439033},
  'name': 'Q11',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.02600000000000023}},
 {'gateError': {'date': '2018-09-20T01:02:15Z',
                        'value': 0.0011547320170097741},
  'name': 'Q12',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.31400000000000006}},
 {'gateError': {'date': '2018-09-20T01:03:22Z',
                        'value': 0.0008256323822705691},
  'name': 'Q13',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.03200000000000003}},
 {'gateError': {'date': '2018-09-20T01:02:15Z',
                        'value': 0.004003386392515573},
  'name': 'Q14',
  'readoutError': {'date': '2018-09-19T20:55:52', 'value': 0.136}},
 {'gateError': {'date': '2018-09-20T01:01:10Z',
                        'value': 0.001511509737380956},
  'name': 'Q15',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.02500000000000022}},
 {'gateError': {'date': '2018-09-20T01:01:10Z',
                        'value': 0.000783947763673909},
  'name': 'Q16',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.02100000000000002}},
 {'gateError': {'date': '2018-09-20T01:03:22Z',
                        'value': 0.0009847877397772264},
  'name': 'Q17',
  'readoutError': {'date': '2018-09-19T20:55:52', 'value': 0.124}},
 {'gateError': {'date': '2018-09-20T01:04:32Z',
                        'value': 0.0029553308978162995},
  'name': 'Q18',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.08799999999999997}},
 {'gateError': {'date': '2018-09-20T01:05:42Z',
                        'value': 0.004024421168266401},
  'name': 'Q19',
  'readoutError': {'date': '2018-09-19T20:55:52',
                           'value': 0.21599999999999997}}]}

{'backend': 'ibmq_20_tokyo',
```

```
'last_update_date': '2018-09-20T04:04:36.000Z',
'qubits': [{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 73.97981251076645},
'T2': {'date': '2018-09-20T00:57:52Z',
        'unit': 'µs',
        'value': 57.45185513429501},
'frequency': {'date': '2018-09-20T04:04:36Z',
                  'unit': 'GHz',
                  'value': 4.490730818760002},
'name': 'Q0'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 57.90009001517608},
'T2': {'date': '2018-09-20T00:58:44Z',
        'unit': 'µs',
        'value': 49.76375027812928},
'frequency': {'date': '2018-09-20T04:04:36Z',
                  'unit': 'GHz',
                  'value': 5.074677796000907},
'name': 'Q1'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 81.80700499358136},
'T2': {'date': '2018-09-20T00:57:52Z',
        'unit': 'µs',
        'value': 57.24147410606729},
'frequency': {'date': '2018-09-20T04:04:36Z',
                  'unit': 'GHz',
                  'value': 4.984768292626855},
'name': 'Q2'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 60.998308964297735},
'T2': {'date': '2018-09-20T00:58:44Z',
        'unit': 'µs',
        'value': 38.60104384267681},
'frequency': {'date': '2018-09-20T04:04:36Z',
                  'unit': 'GHz',
                  'value': 5.1094796207543975},
'name': 'Q3'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 94.03263643897567},
'T2': {'date': '2018-09-20T00:57:52Z',
        'unit': 'µs',
        'value': 58.01405979931684},
'frequency': {'date': '2018-09-20T04:04:36Z',
                  'unit': 'GHz',
                  'value': 5.125198514600377},
'name': 'Q4'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 110.68589325313599},
'T2': {'date': '2018-09-20T00:58:44Z',
        'unit': 'µs',
        'value': 59.66365678384753},
'frequency': {'date': '2018-09-20T04:04:36Z',
                  'unit': 'GHz',
                  'value': 4.959163147824194},
'name': 'Q5'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 88.85140461675599},
'T2': {'date': '2018-09-20T00:59:34Z',
        'unit': 'µs',
```

```
        'value': 65.53941040812195},
 'frequency': {'date': '2018-09-20T04:04:36Z',
               'unit': 'GHz',
               'value': 5.229591051536364},
 'name': 'Q6'},
{'T1': {'date': '2018-09-18T03:51:02Z',
        'unit': 'µs',
        'value': 127.71208127230781},
 'T2': {'date': '2018-09-20T01:00:22Z',
        'unit': 'µs',
        'value': 69.67014805739245},
 'frequency': {'date': '2018-09-20T04:04:36Z',
               'unit': 'GHz',
               'value': 4.662443378664702},
 'name': 'Q7'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 113.70288906142393},
 'T2': {'date': '2018-09-20T00:59:34Z',
        'unit': 'µs',
        'value': 62.244369026050414},
 'frequency': {'date': '2018-09-20T04:04:36Z',
               'unit': 'GHz',
               'value': 4.898170007936852},
 'name': 'Q8'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 58.5076847544152},
 'T2': {'date': '2018-09-20T01:00:22Z',
        'unit': 'µs',
        'value': 41.67305899678406},
 'frequency': {'date': '2018-09-20T04:04:36Z',
               'unit': 'GHz',
               'value': 5.23356320846535},
 'name': 'Q9'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 79.30371805842574},
 'T2': {'date': '2018-09-20T00:57:52Z',
        'unit': 'µs',
        'value': 49.01314546901237},
 'frequency': {'date': '2018-09-20T04:04:36Z',
               'unit': 'GHz',
               'value': 5.143066844328144},
 'name': 'Q10'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 82.05509127425432},
 'T2': {'date': '2018-09-20T01:00:22Z',
        'unit': 'µs',
        'value': 69.69627881631212},
 'frequency': {'date': '2018-09-20T04:04:36Z',
               'unit': 'GHz',
               'value': 5.033452505246013},
 'name': 'Q11'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'µs',
        'value': 155.472440371174},
 'T2': {'date': '2018-09-20T00:57:52Z',
        'unit': 'µs',
        'value': 82.66992944021662},
 'frequency': {'date': '2018-09-20T04:04:36Z',
               'unit': 'GHz',
               'value': 4.457739048688047},
 'name': 'Q12'},
{'T1': {'date': '2018-09-20T00:56:33Z',
```

```
        'unit': 'μs',
        'value': 98.82957987929765},
 'T2': {'date': '2018-09-20T00:58:44Z',
        'unit': 'μs',
        'value': 65.40292977993548},
 'frequency': {'date': '2018-09-20T04:04:36Z',
                   'unit': 'GHz',
                   'value': 5.019179601871802},
 'name': 'Q13'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'μs',
        'value': 54.07355005973775},
 'T2': {'date': '2018-09-20T00:57:52Z',
        'unit': 'μs',
        'value': 49.29794199258292},
 'frequency': {'date': '2018-09-20T04:04:36Z',
                   'unit': 'GHz',
                   'value': 5.080117334592717},
 'name': 'Q14'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'μs',
        'value': 82.91385036748657},
 'T2': {'date': '2018-09-20T00:58:44Z',
        'unit': 'μs',
        'value': 61.5228915646018},
 'frequency': {'date': '2018-09-20T04:04:36Z',
                   'unit': 'GHz',
                   'value': 5.020233434629288},
 'name': 'Q15'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'μs',
        'value': 85.67928720189813},
 'T2': {'date': '2018-09-20T00:59:34Z',
        'unit': 'μs',
        'value': 58.63905527468041},
 'frequency': {'date': '2018-09-20T04:04:36Z',
                   'unit': 'GHz',
                   'value': 4.903254256329702},
 'name': 'Q16'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'μs',
        'value': 74.44205335344174},
 'T2': {'date': '2018-09-20T00:58:44Z',
        'unit': 'μs',
        'value': 53.824983990499895},
 'frequency': {'date': '2018-09-20T04:04:36Z',
                   'unit': 'GHz',
                   'value': 4.778395760893867},
 'name': 'Q17'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'μs',
        'value': 77.84556028168906},
 'T2': {'date': '2018-09-20T00:59:34Z',
        'unit': 'μs',
        'value': 42.45041778142437},
 'frequency': {'date': '2018-09-20T04:04:36Z',
                   'unit': 'GHz',
                   'value': 5.094016718022559},
 'name': 'Q18'},
{'T1': {'date': '2018-09-20T00:56:33Z',
        'unit': 'μs',
        'value': 33.54235712417796},
 'T2': {'date': '2018-09-20T01:00:22Z',
        'unit': 'μs',
        'value': 7.413384481068095},
 'frequency': {'date': '2018-09-20T04:04:36Z',
```

```
                    'unit': 'GHz',
                    'value': 5.0591896753481045},
'name': 'Q19'}]}
```

C.2 OUTPUT QUANTUM CIRCUITS

C.2.1 *2-qubit implementation*

$\phi = 0$

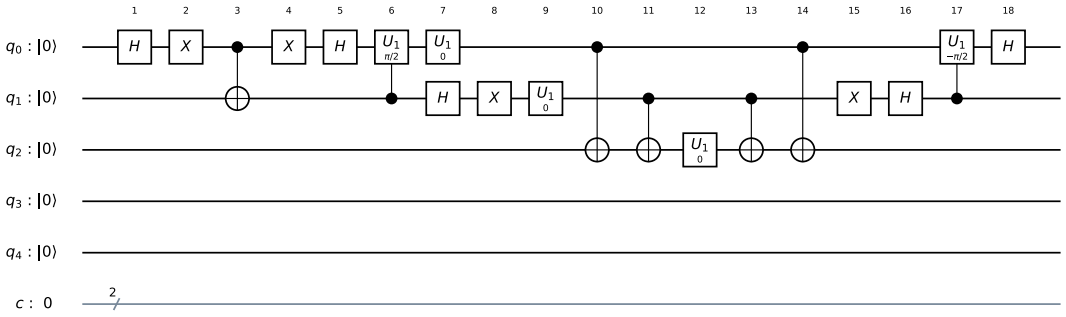


Figure 29.: Circuit representation of the 2-qubit simulation algorithm, for $\phi = 0$.

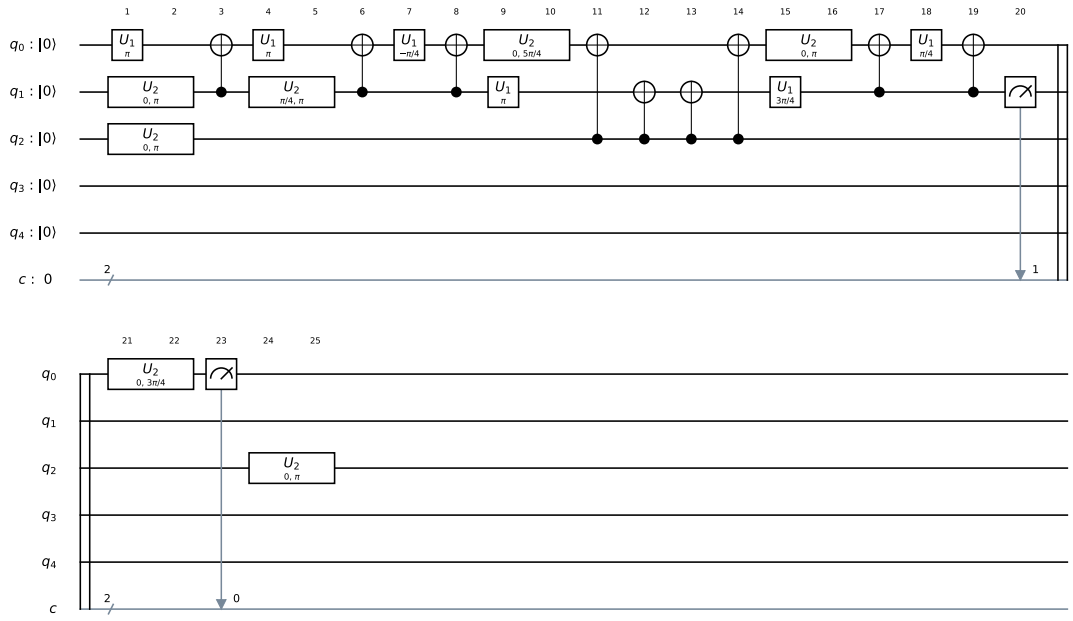


Figure 30.: Circuit representation of the 2-qubit simulation algorithm implemented using QISKit's compiler in *ibmqx4*, for $\phi = 0$.

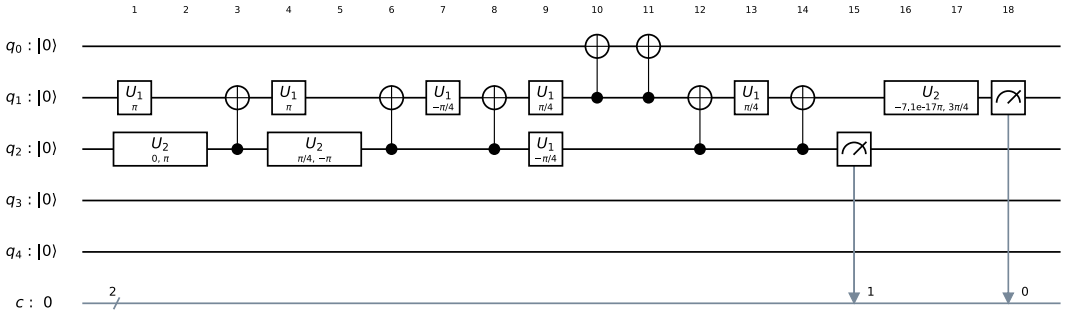


Figure 31.: Circuit representation of the 2-qubit simulation algorithm implemented using the alternative compiler in *ibmqx4*, for $\phi = 0$.

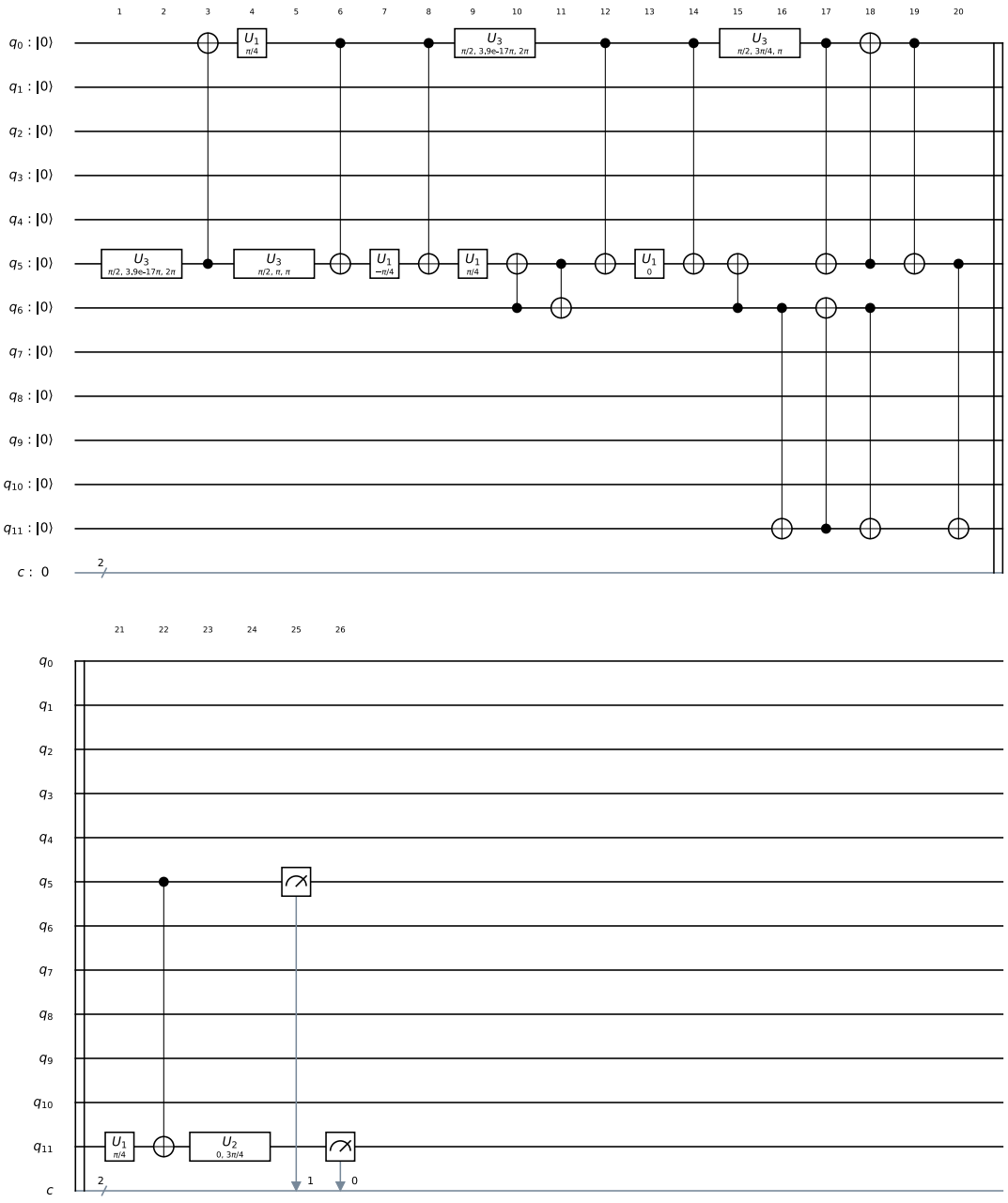


Figure 32.: Circuit representation of the 2-qubit simulation algorithm implemented using QISKit's compiler in *ibmq20*, for $\phi = 0$.

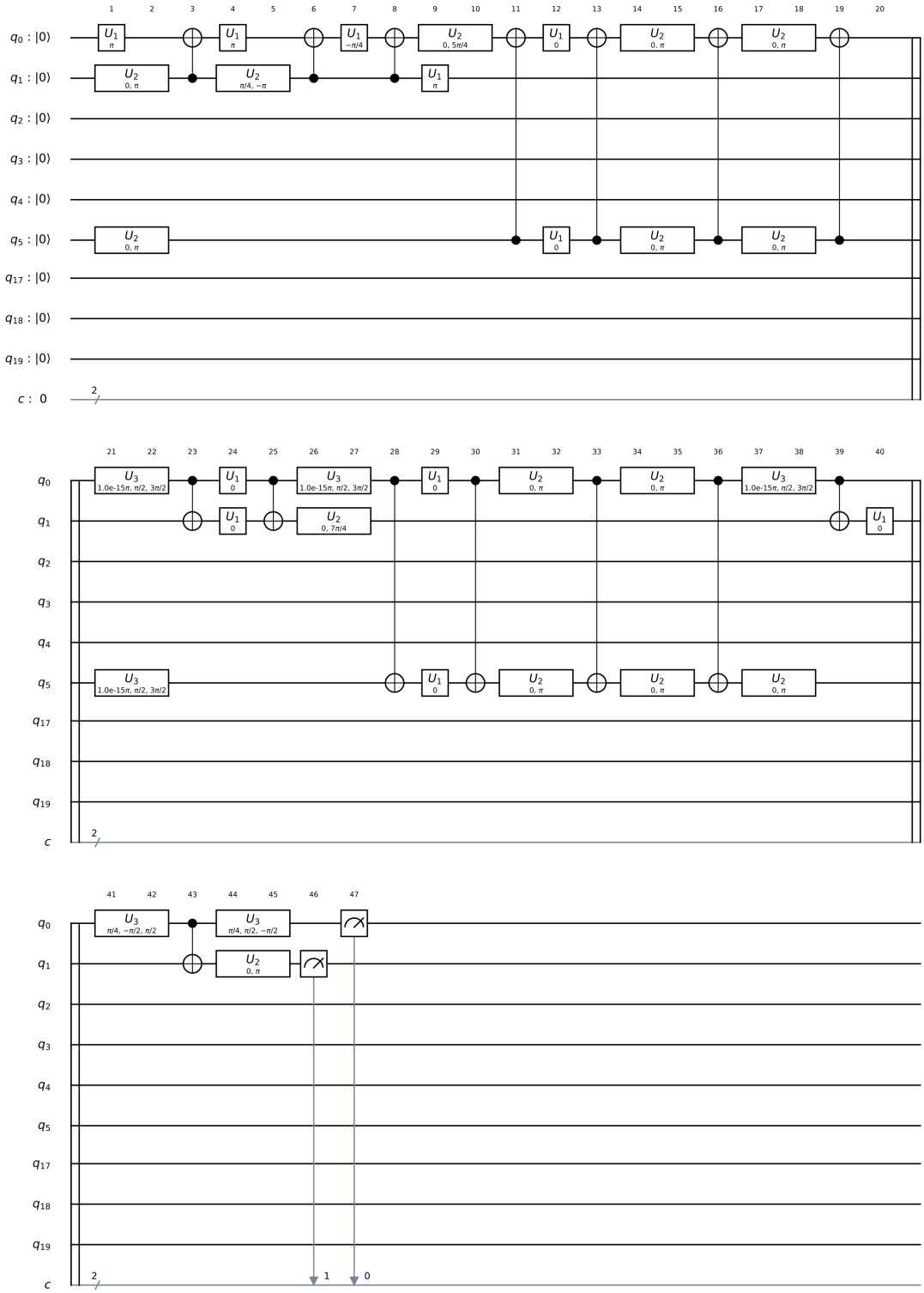


Figure 33.: Circuit representation of the 2-qubit simulation algorithm implemented using the alternative compiler in *ibmq20*, for $\phi = 0$.

$\phi = \pi/2$

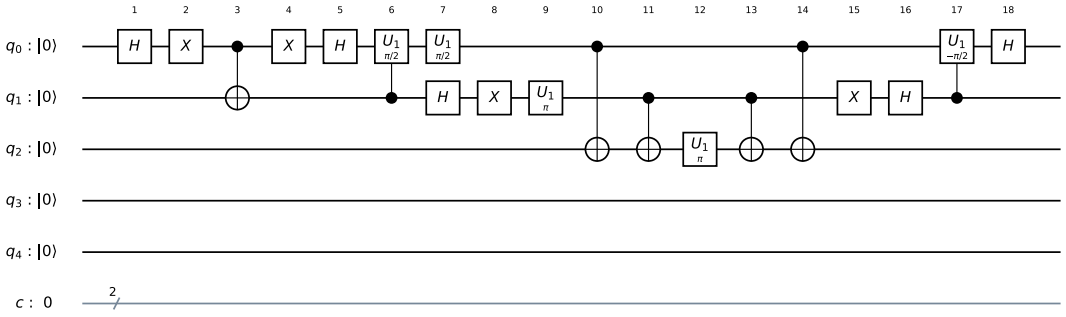


Figure 34.: Circuit representation of the 2-qubit simulation algorithm, for $\phi = \pi/2$.

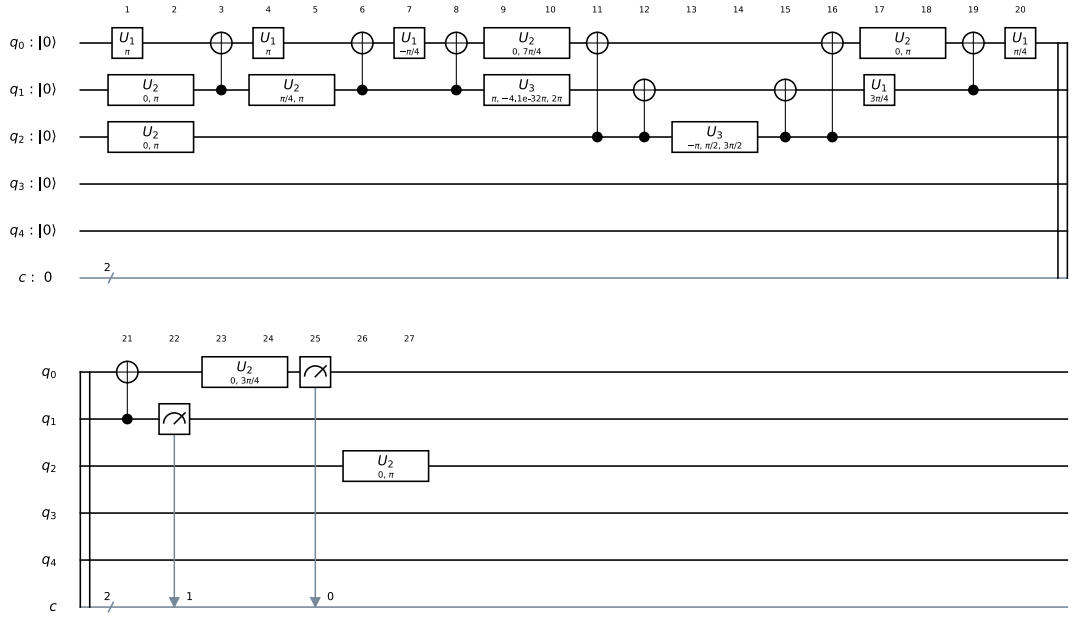


Figure 35.: Circuit representation of the 2-qubit simulation algorithm implemented using QISKit's compiler in *ibmqx4*, for $\phi = \pi/2$.

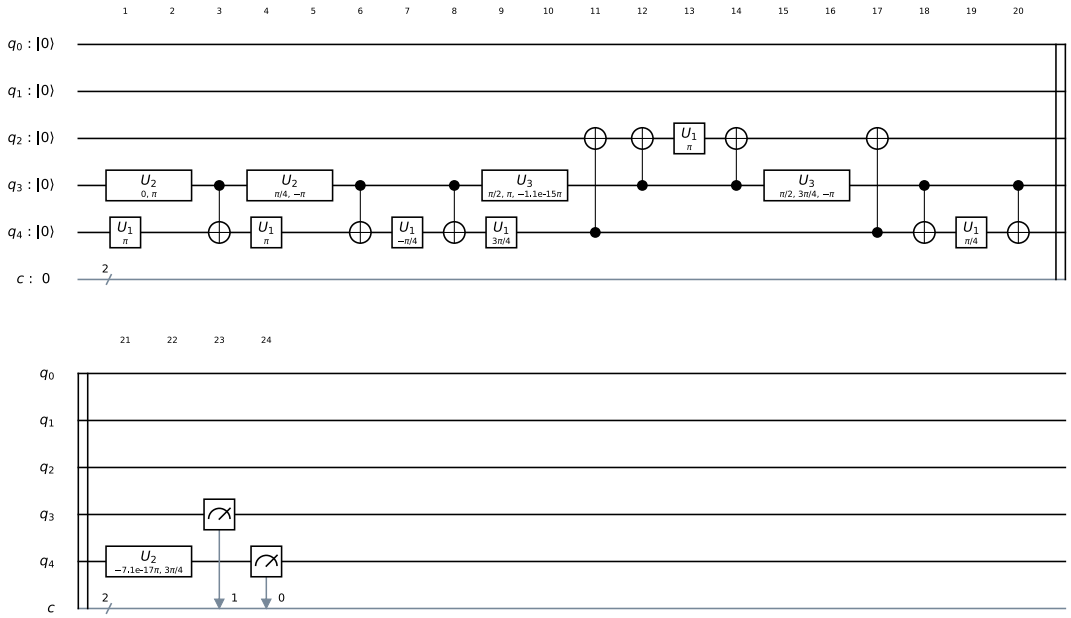


Figure 36.: Circuit representation of the 2-qubit simulation algorithm implemented using the alternative compiler in *ibmqx4*, for $\phi = \pi/2$.

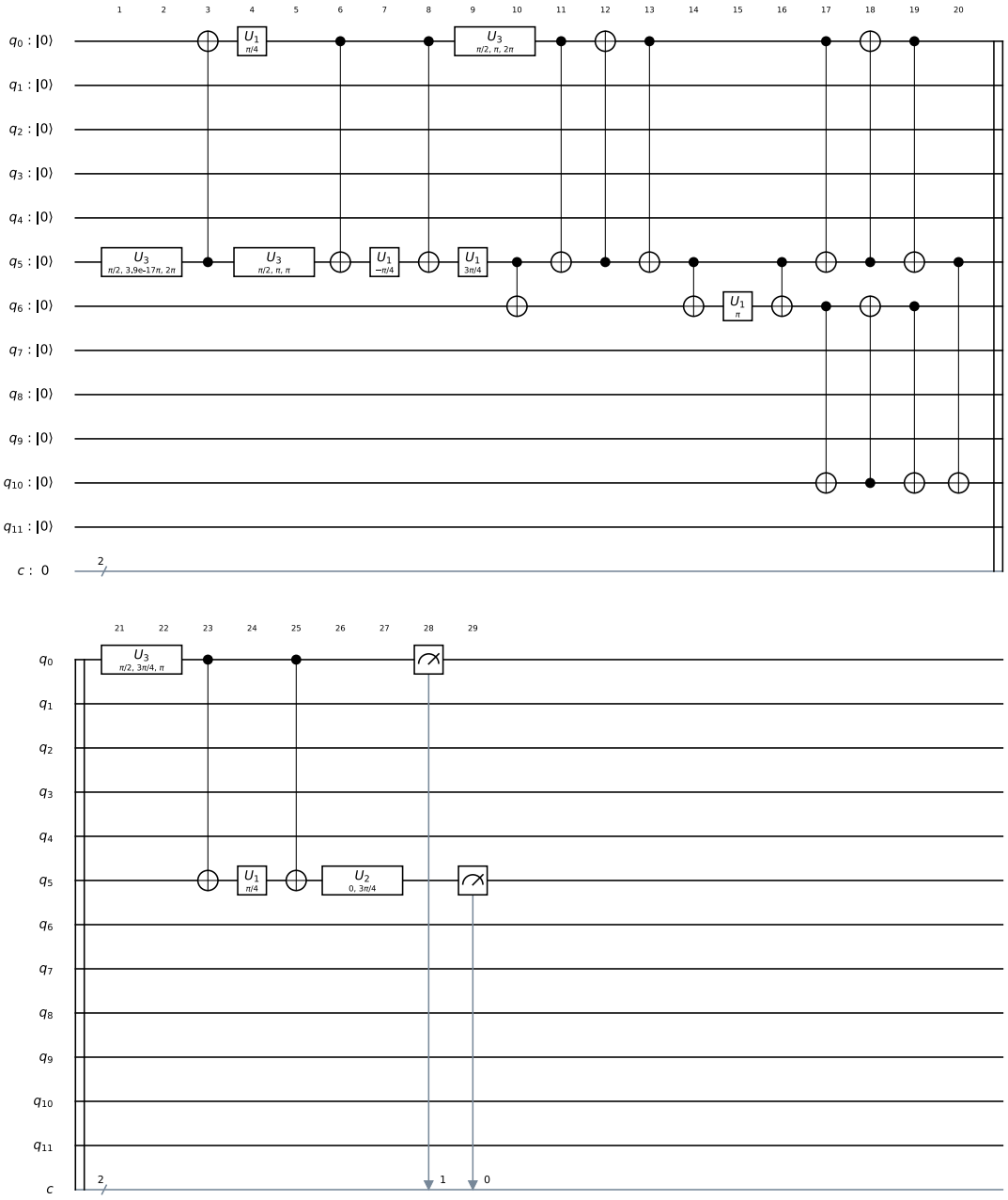


Figure 37.: Circuit representation of the 2-qubit simulation algorithm implemented using QISKit's compiler in *ibmq20*, for $\phi = \pi/2$.

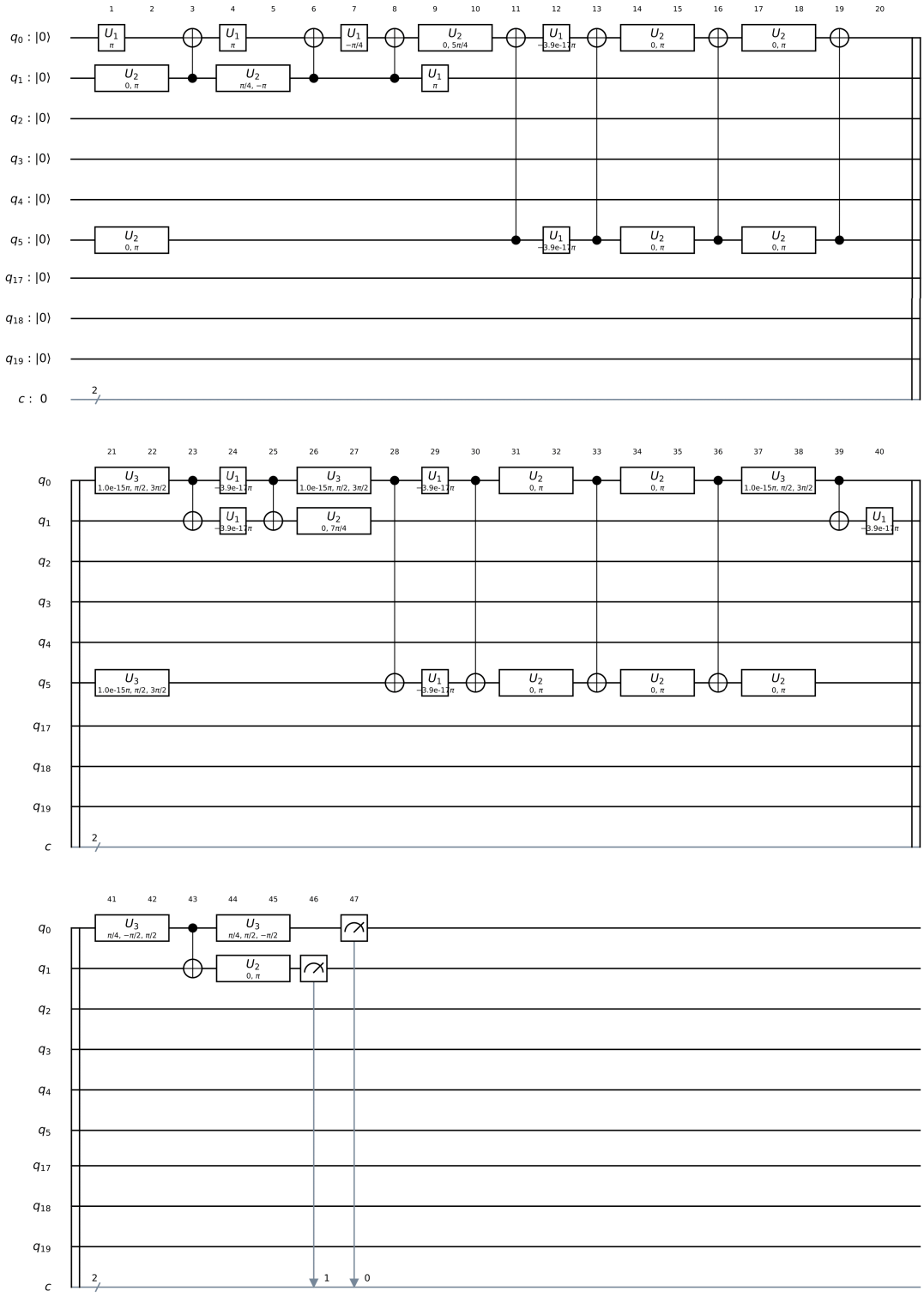


Figure 38.: Circuit representation of the 2-qubit simulation algorithm implemented using the alternative compiler in *ibmq20*, for $\phi = \pi/2$.

$\phi = \pi$

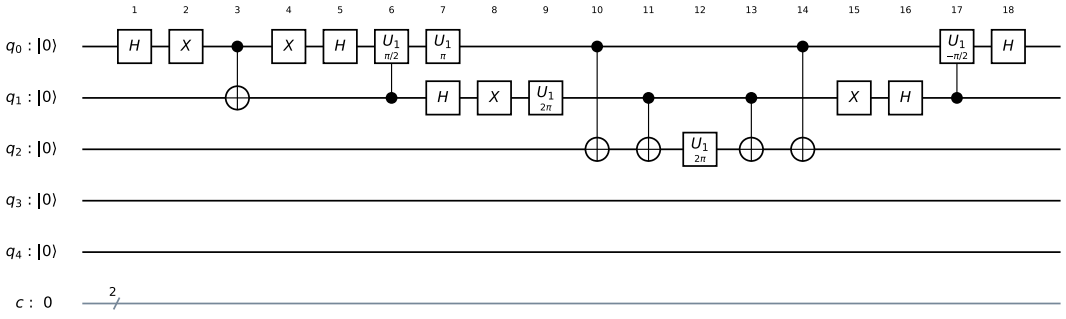


Figure 39.: Circuit representation of the 2-qubit simulation algorithm, for $\phi = \pi$.

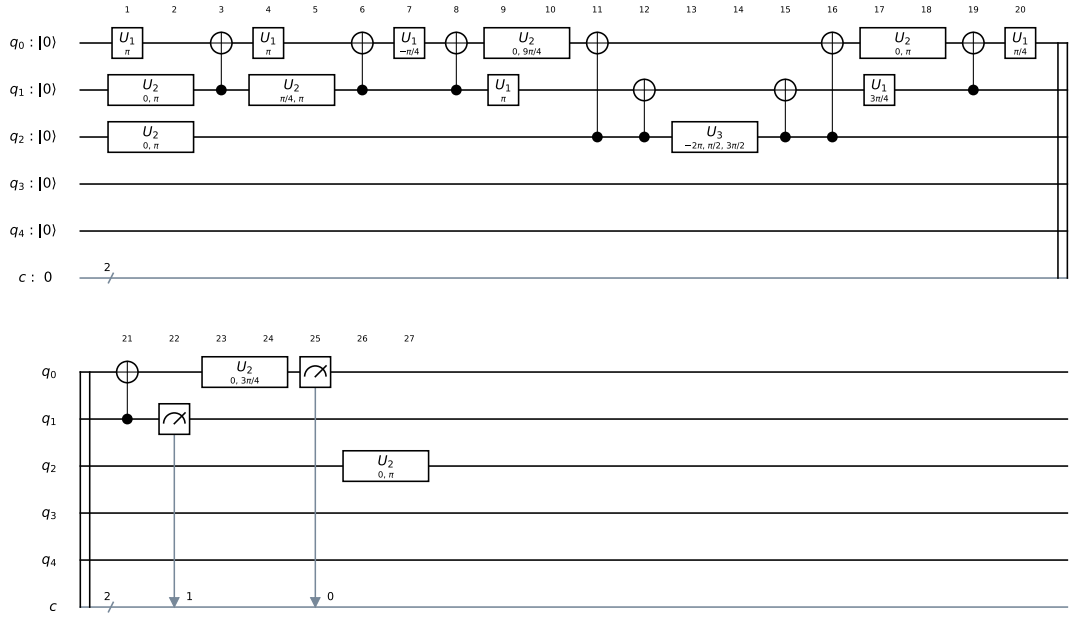


Figure 40.: Circuit representation of the 2-qubit simulation algorithm implemented using QISKit's compiler in *ibmqx4*, for $\phi = \pi$.

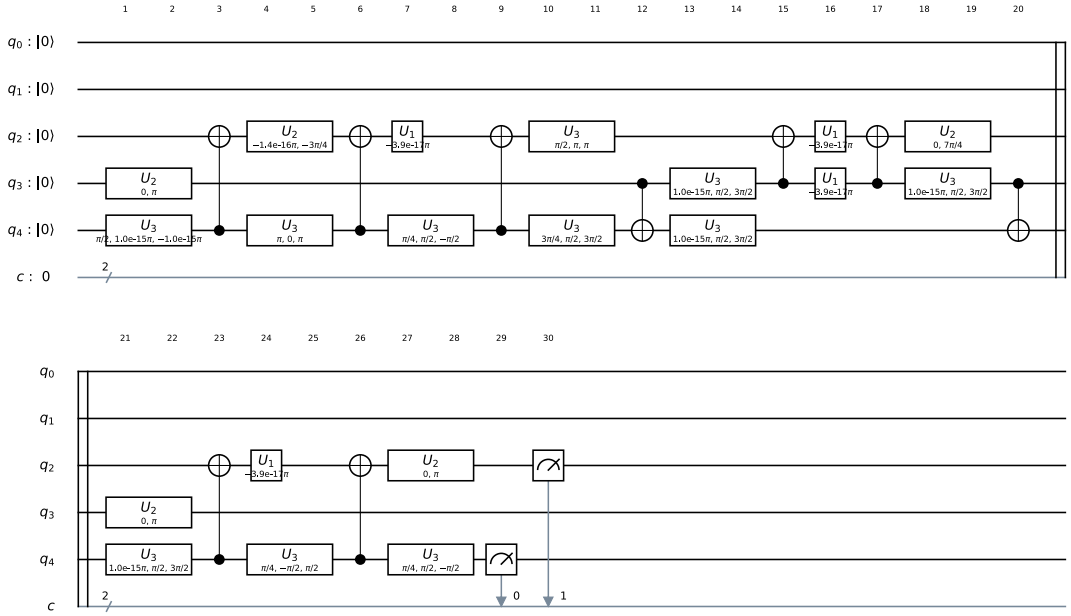


Figure 41.: Circuit representation of the 2-qubit simulation algorithm implemented using the alternative compiler in *ibmqx4*, for $\phi = \pi$.

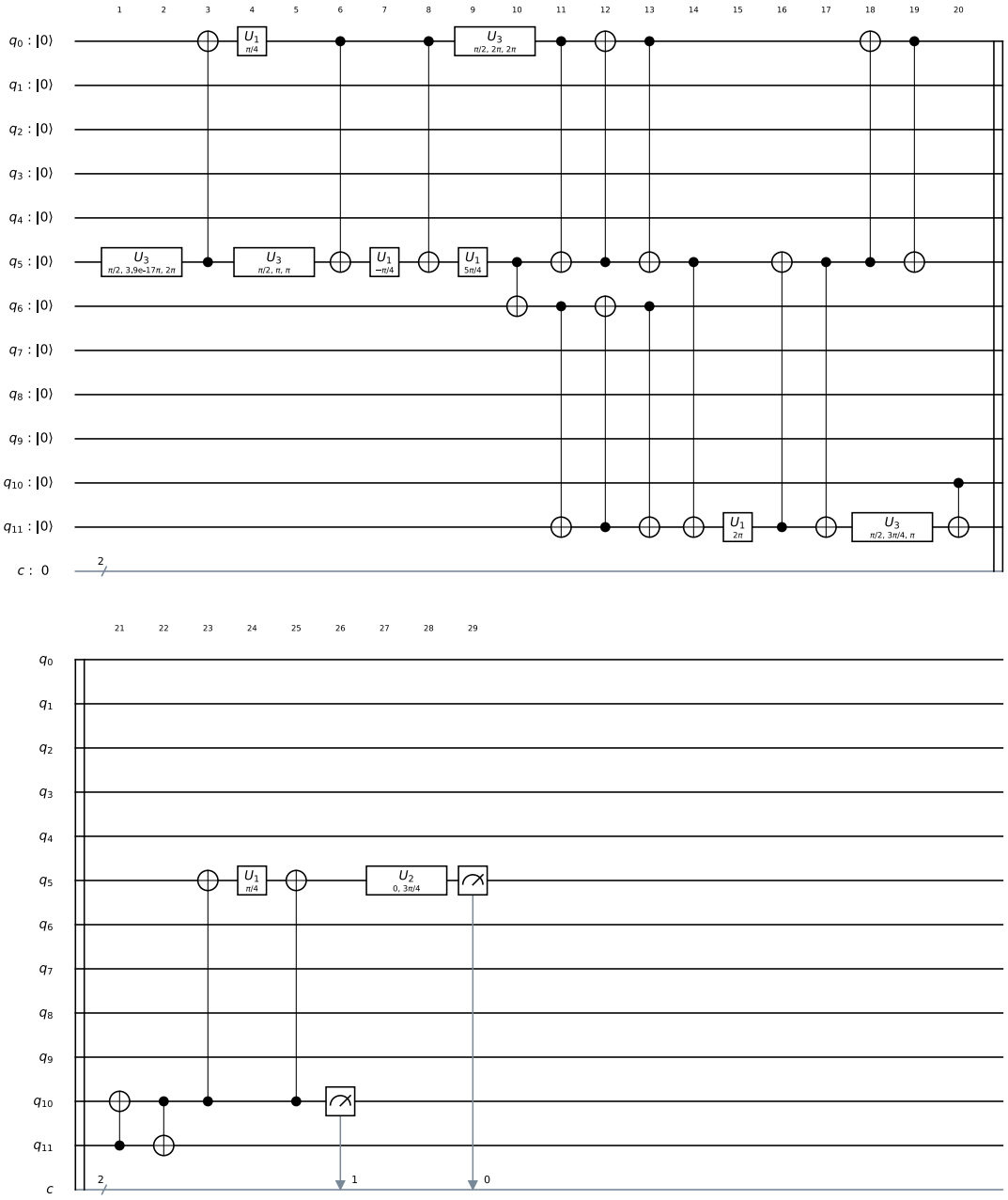


Figure 42.: Circuit representation of the 2-qubit simulation algorithm implemented using QISKit's compiler in *ibmq20*, for $\phi = \pi$.

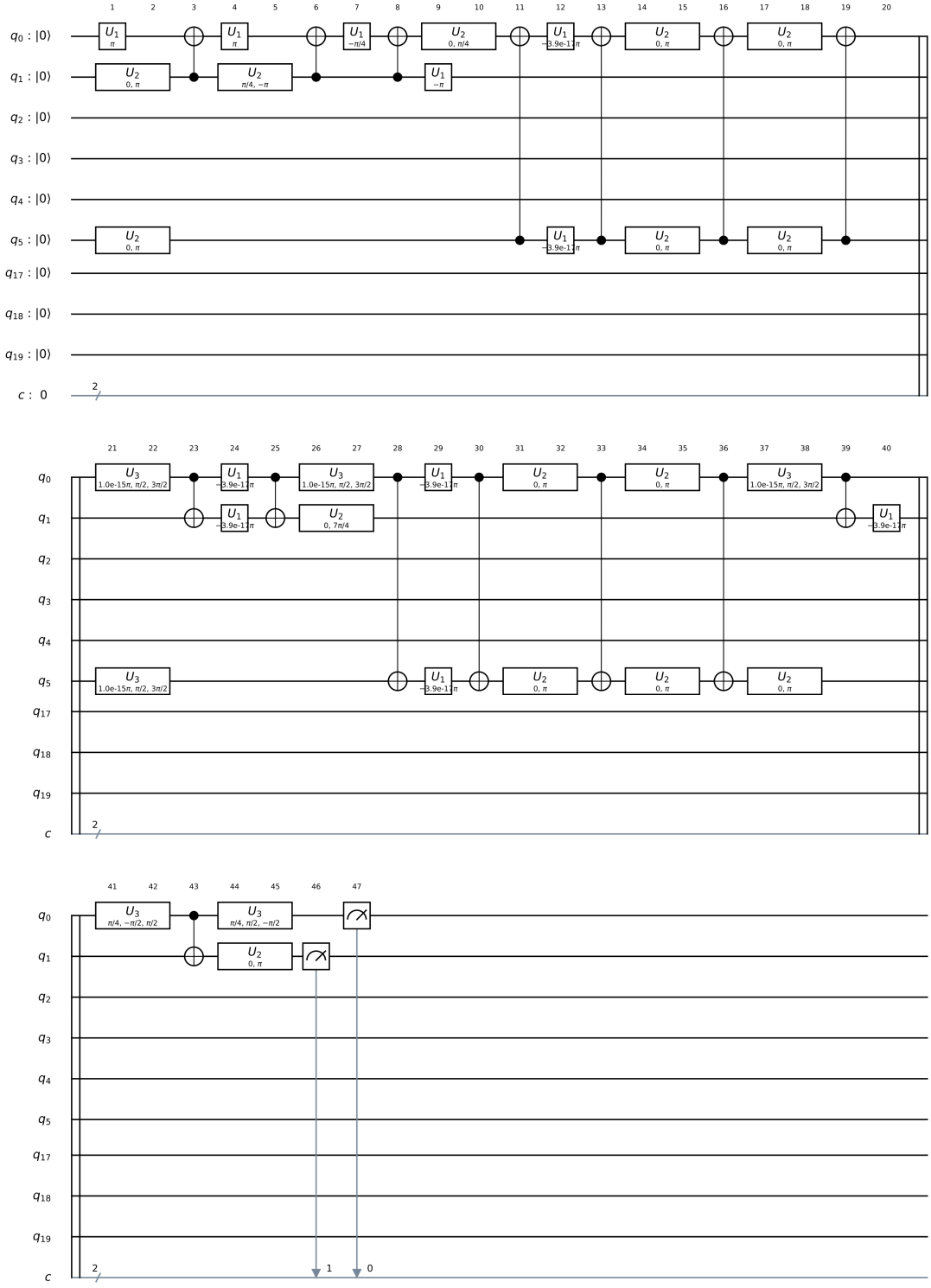


Figure 43.: Circuit representation of the 2-qubit simulation algorithm implemented using the alternative compiler in *ibmq20*, for $\phi = \pi$.

C.2.2 *3-qubit implementation*

$\phi = 0$

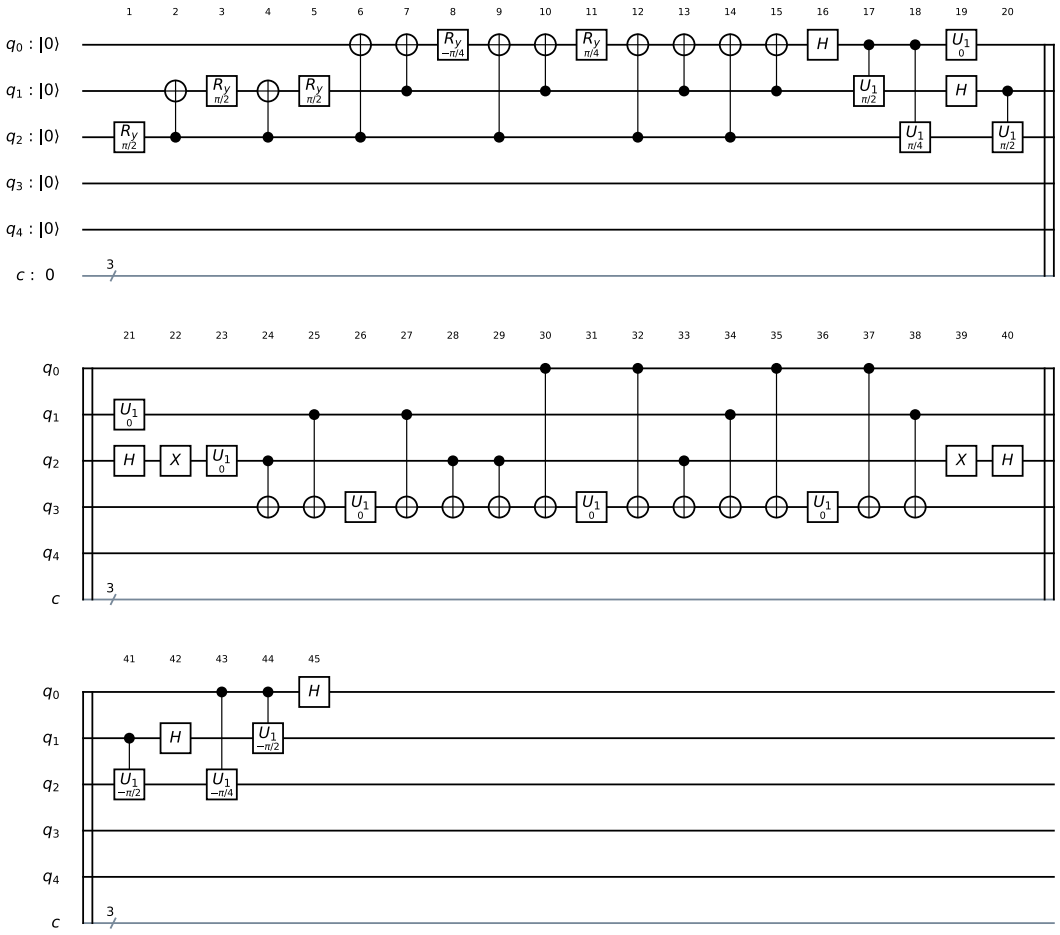


Figure 44.: Circuit representation of the 3-qubit simulation algorithm, for $\phi = 0$.

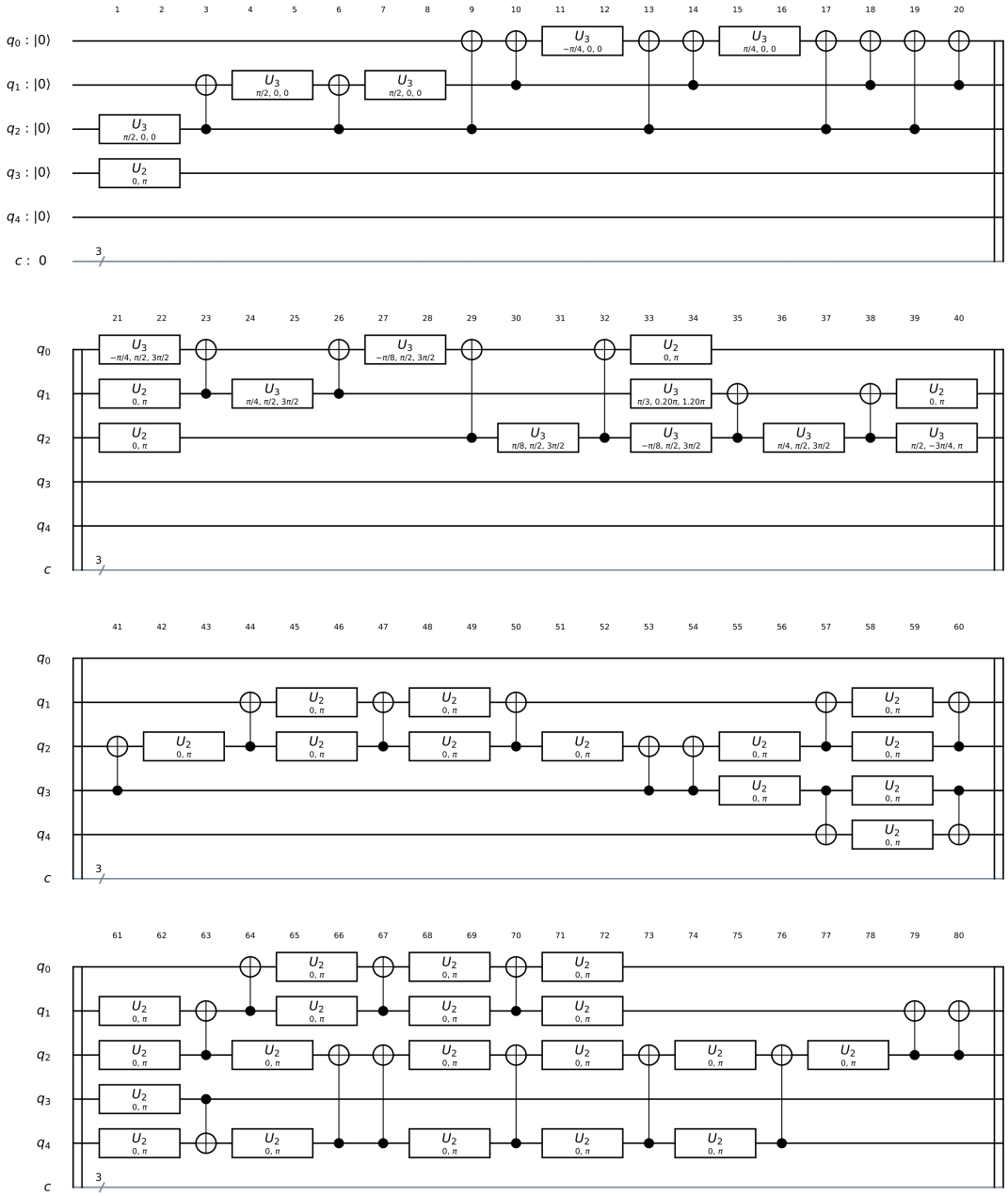


Figure 45.: Circuit representation of the 3-qubit simulation algorithm implemented using QISKit's compiler in *ibmqx4*, for $\phi = 0$ (section 1).

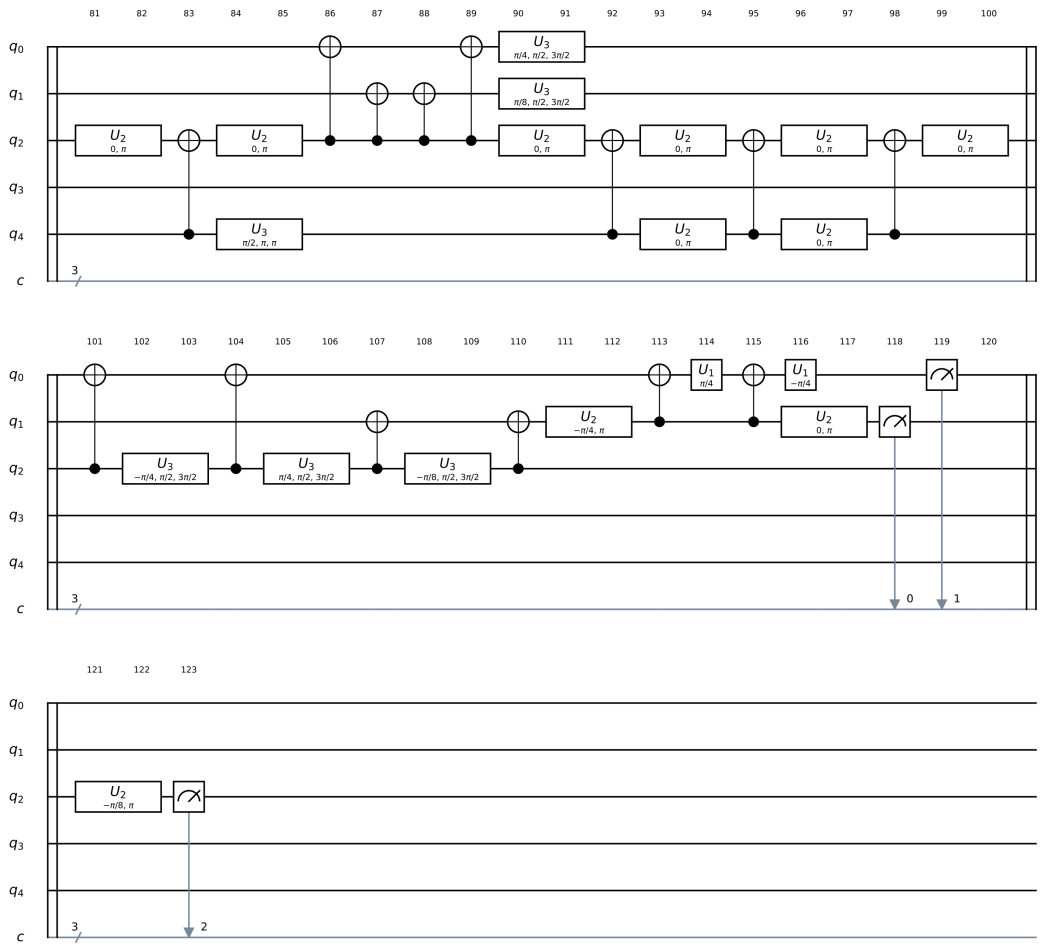


Figure 46.: Circuit representation of the 3-qubit simulation algorithm implemented using QISKit's compiler in *ibmqx4*, for $\phi = 0$ (section 2).

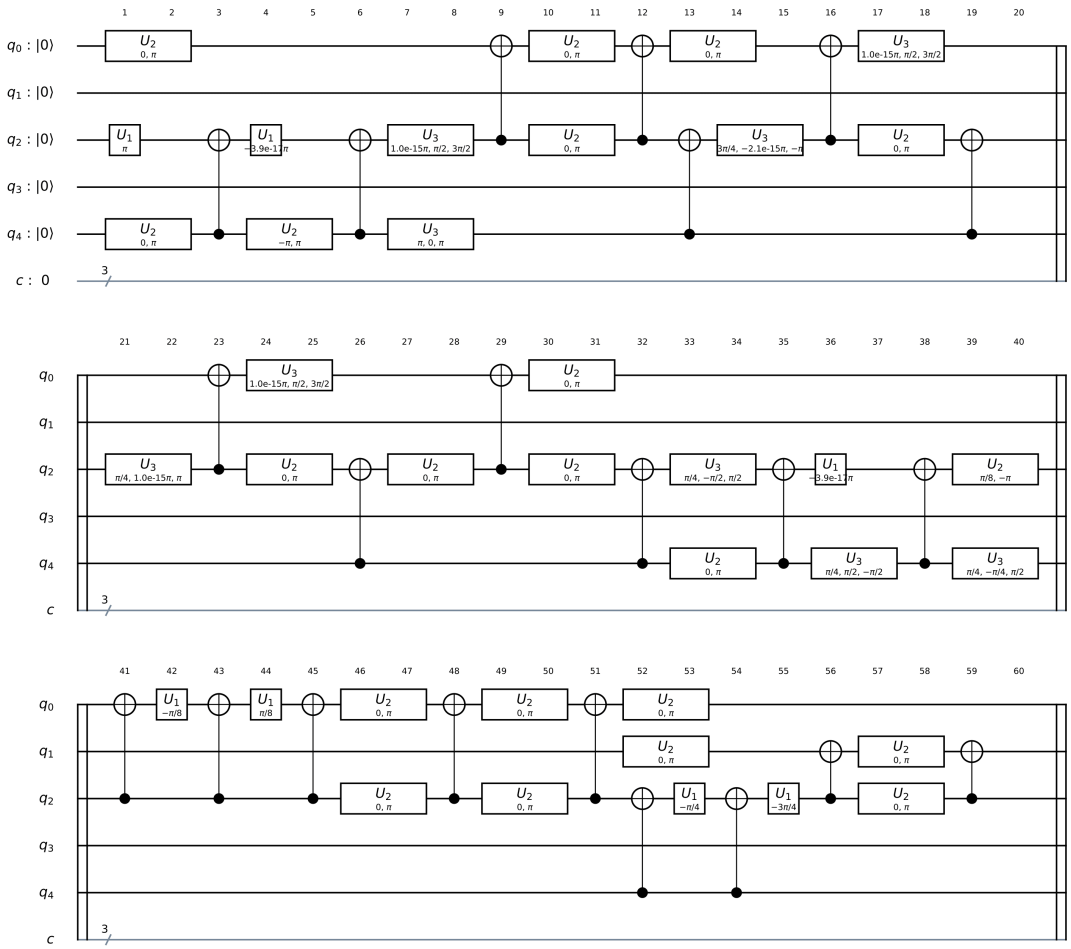


Figure 47.: Circuit representation of the 3-qubit simulation algorithm implemented using the alternative compiler in *ibmqx4*, for $\phi = 0$ (section 1).

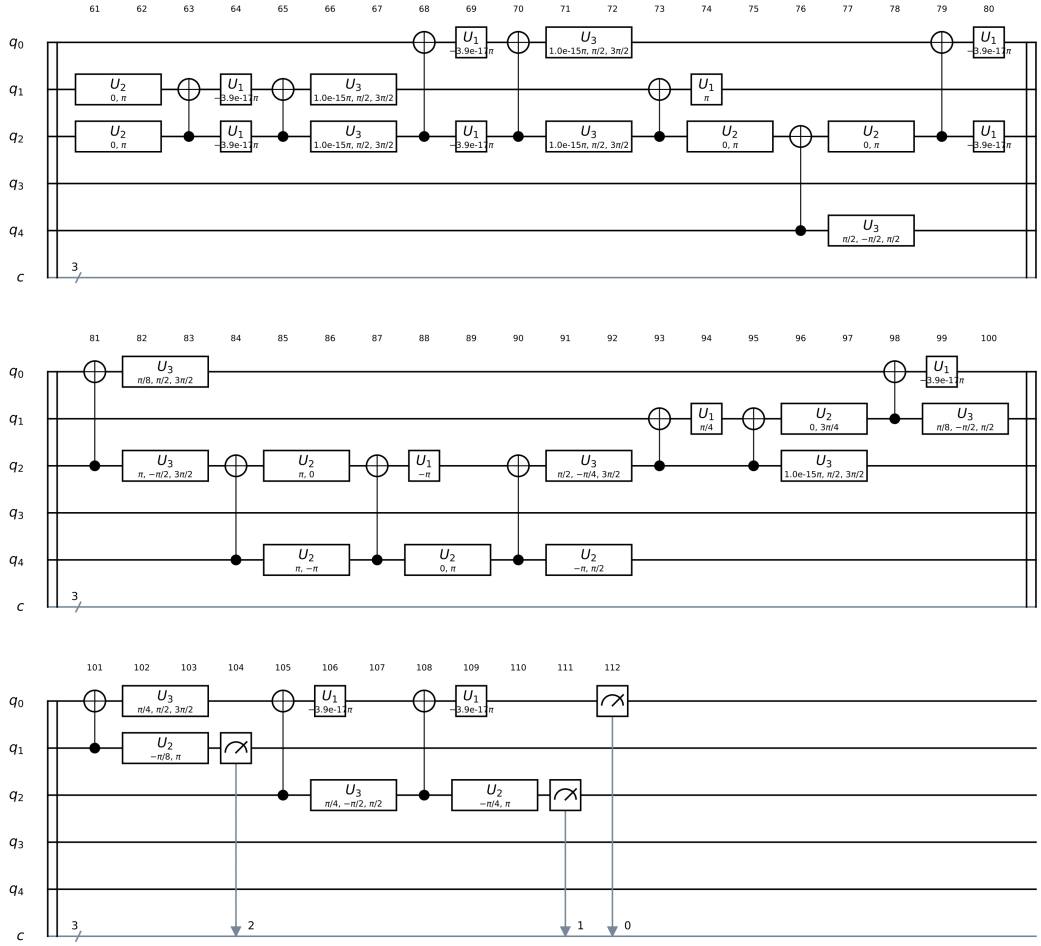


Figure 48.: Circuit representation of the 3-qubit simulation algorithm implemented using the alternative compiler in *ibmqx4*, for $\phi = 0$ (section 2).

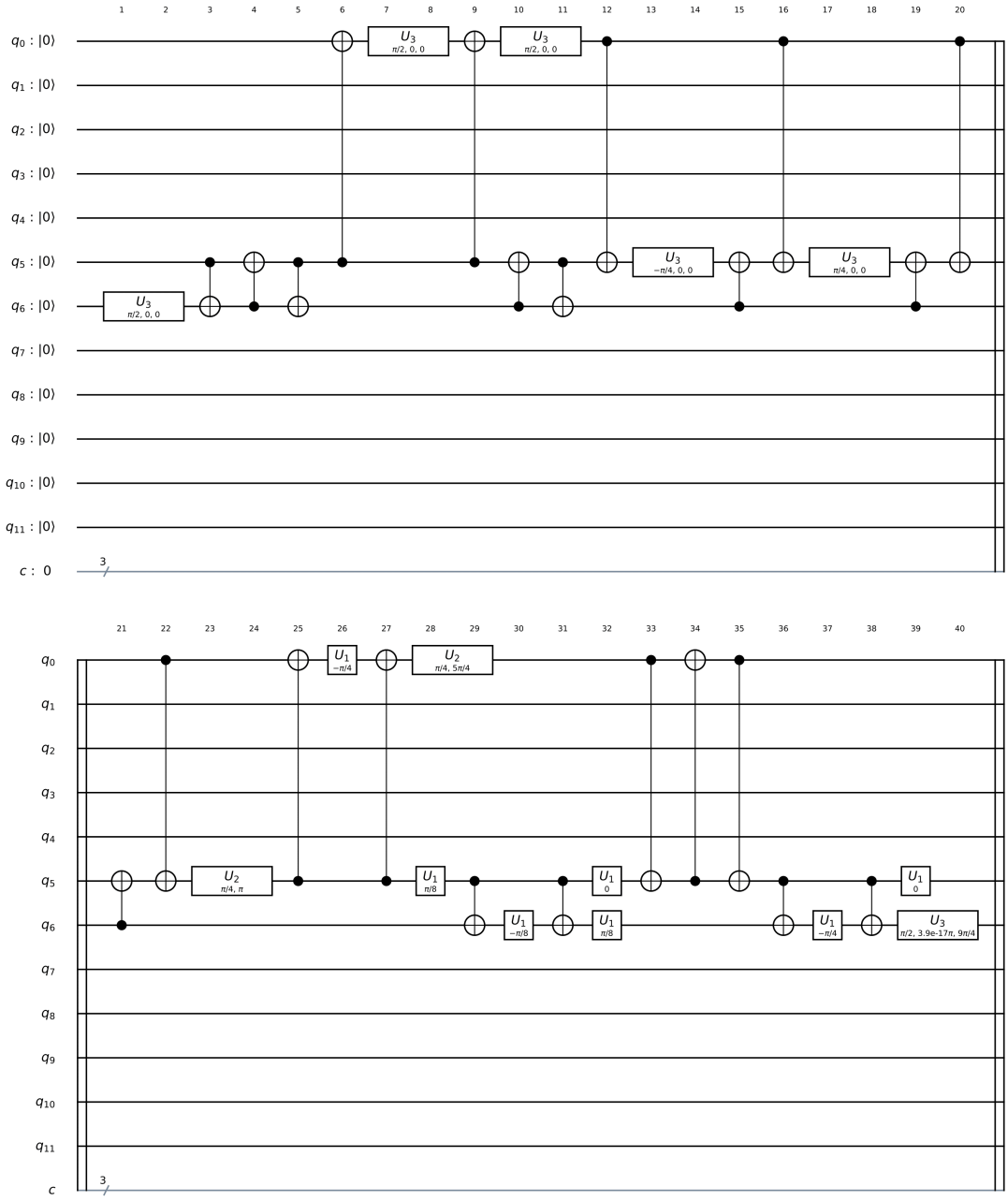


Figure 49.: Circuit representation of the 3-qubit simulation algorithm implemented using QISKit's compiler in *ibmq20*, for $\phi = 0$ (section 1).

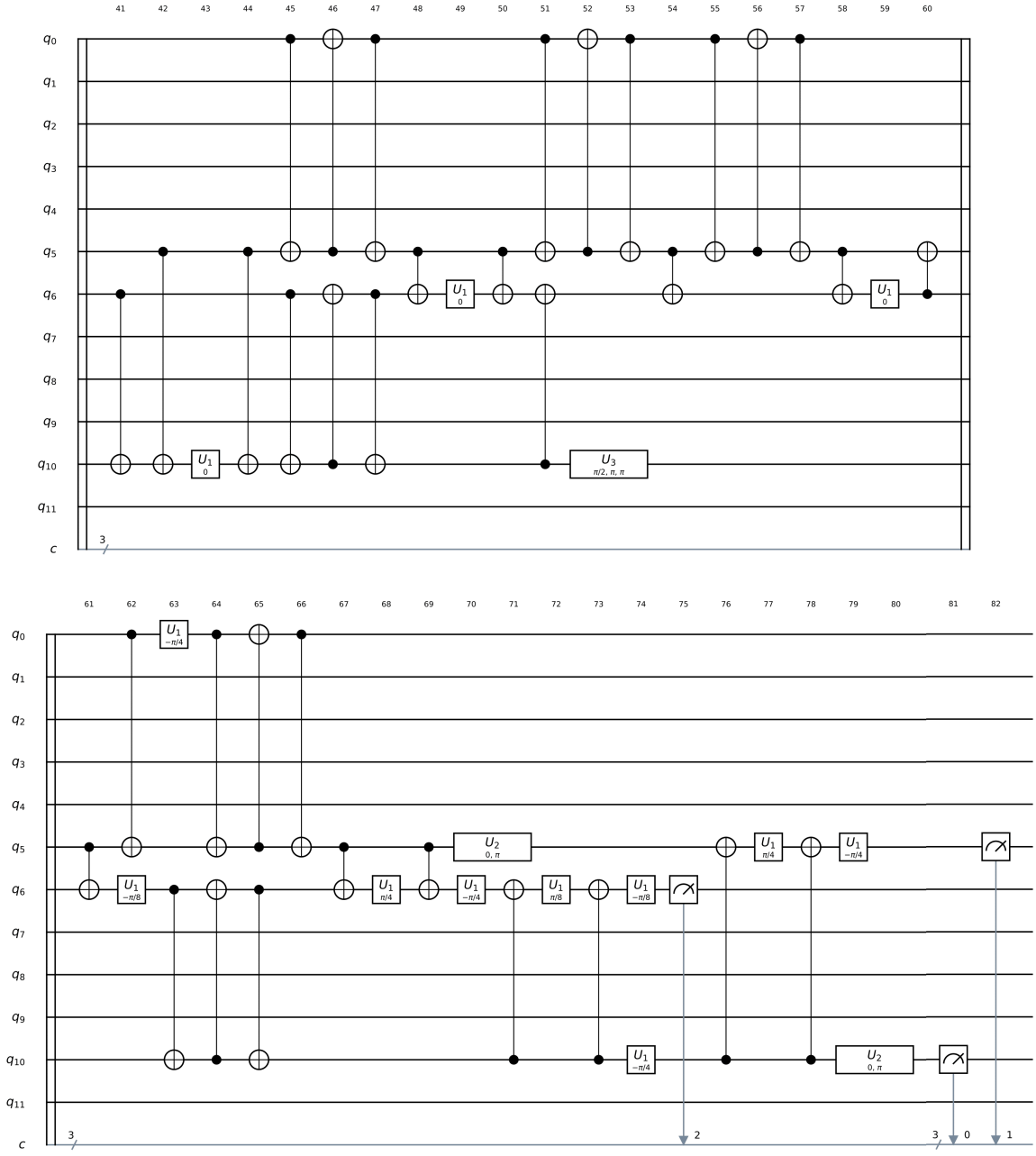


Figure 50.: Circuit representation of the 3-qubit simulation algorithm implemented using QISKit's compiler in *ibmq20*, for $\phi = 0$ (section 2).

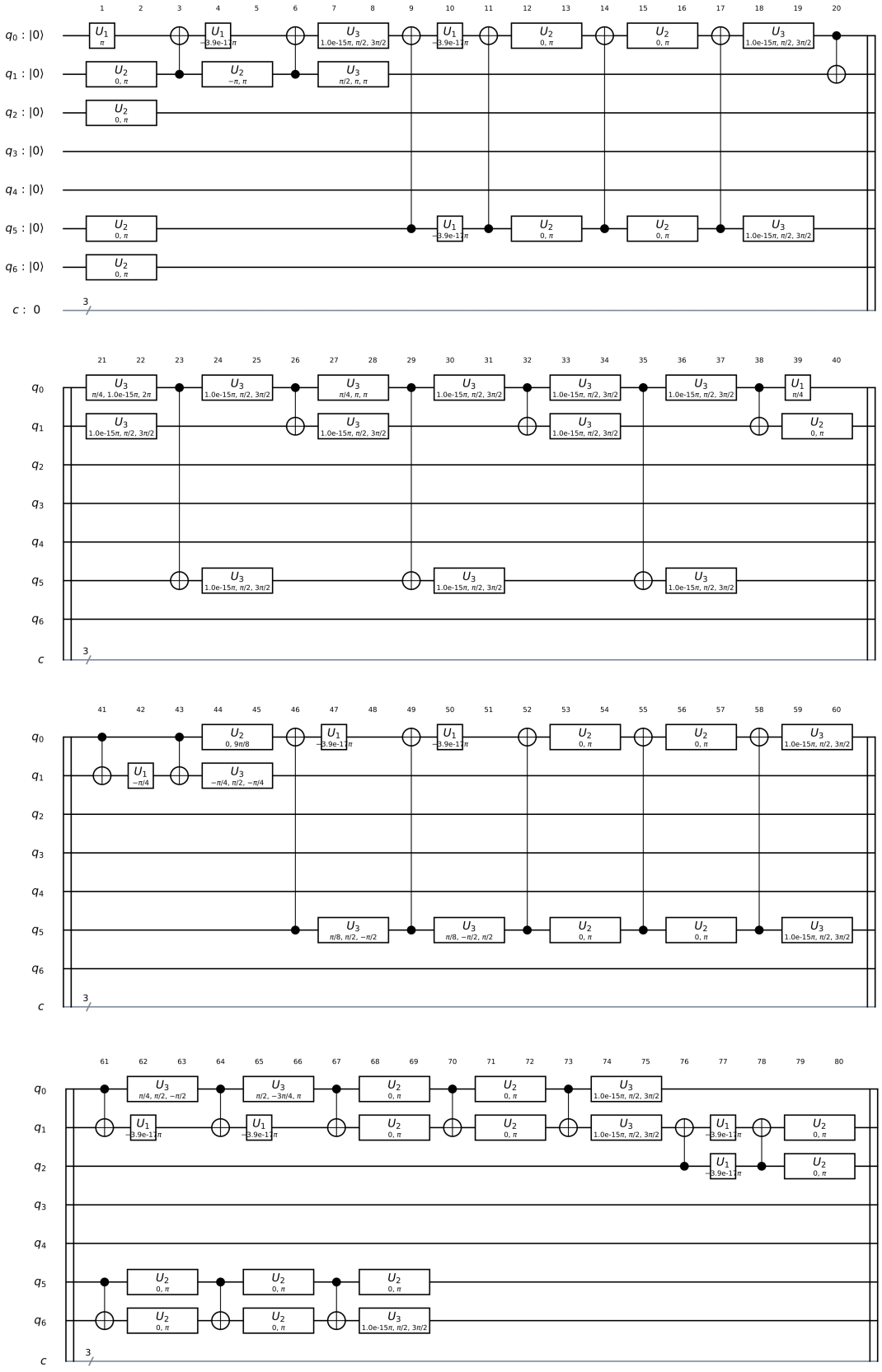


Figure 51.: Circuit representation of the 3-qubit simulation algorithm implemented using the alternative compiler in *ibmq20*, for $\phi = 0$ (section 1).

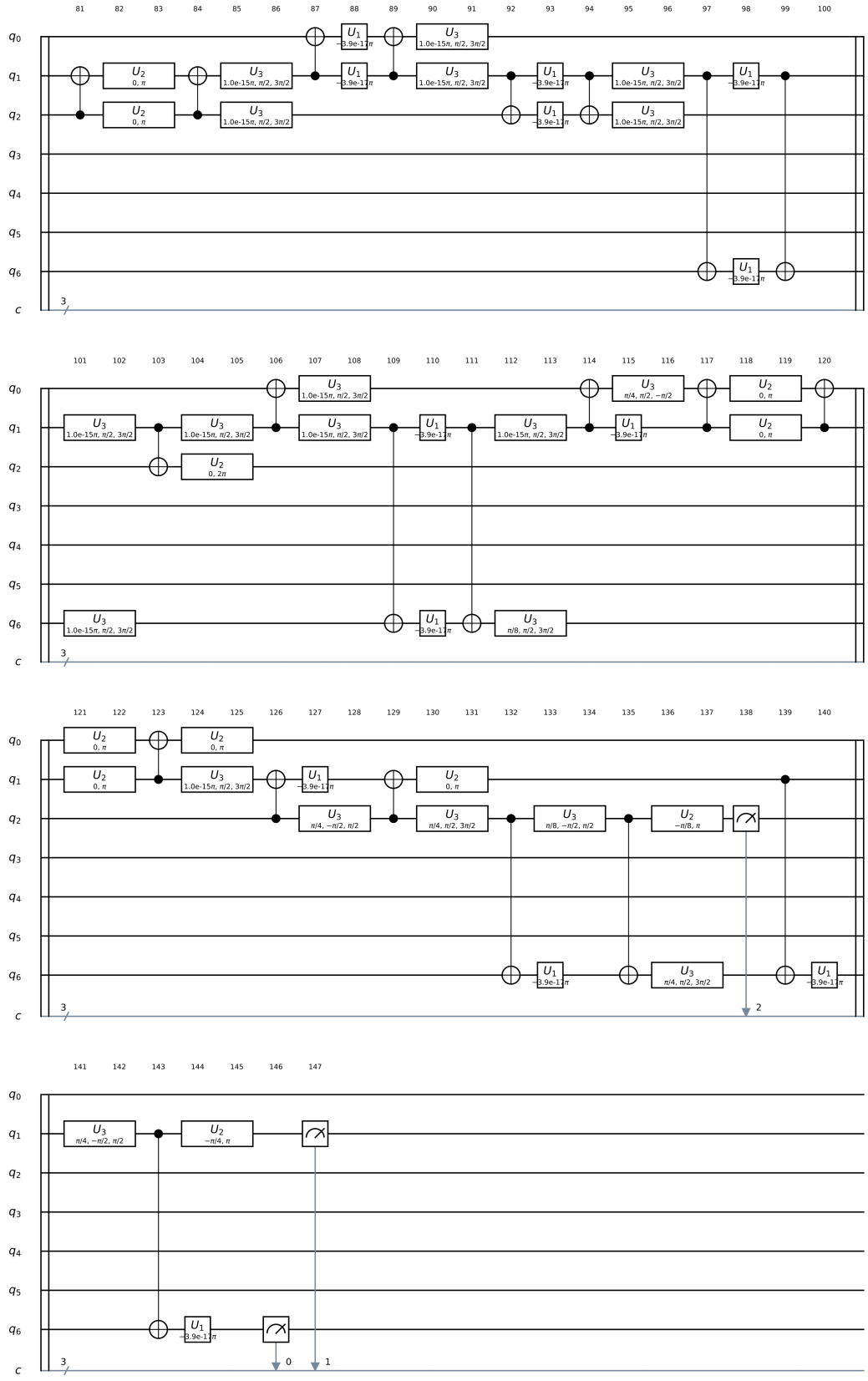


Figure 52.: Circuit representation of the 3-qubit simulation algorithm implemented using the alternative compiler in *ibmq20*, for $\phi = 0$ (section 2).

NB: place here information about funding, FCT project, etc in which the work is framed. Leave empty otherwise.