Resources Estimation for Quantum Computing Algorithms in Multiple Physical Platforms

Cesar Zaragoza Cortes

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2021

Reading Committee:
Jeffrey Wilkes, Chair
Boris Blinov

Program Authorized to Offer Degree:
Physics

University of Washington

Abstract

Resources Estimation for Quantum Computing Algorithms in Multiple Physical Platforms

Cesar Zaragoza Cortes

Chair of the Supervisory Committee:
Professor Emeritus Jeffrey Wilkes
Department of Physics

An important task in the development of quantum computing technologies is to determine the resources needed to execute a particular algorithm in a device with certain characteristics. Resources estimation allows not only to determine whether it is possible to execute an algorithm in a current Noisy Intermediate-Scale Quantum (NISQ) device, but also to experiment with the parameters of quantum hardware to find out how much it would have to scale to achieve quantum advantage. We use the Q# quantum programming language to implement the Bernstein-Vazirani algorithm, leverage simulators to perform resources estimation for trapped-ion and superconducting quantum hardware platforms, and compare the results.

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

Algorithms designed for quantum computers have the potential to solve some problems that cannot be efficiently solved by algorithms designed for classical computers. However, estimating how much resources are needed to execute a quantum algorithm that outperforms a classical one is a difficult task. There are many quantum programming languages and tools built around them such as Q#[4], Qiskit[5] and Cirq[2] that allow execution of quantum algorithms on simulators but out-of-the-box options to estimate resources are limited to the logical level or not existent.

1.1 The Purpose of This Thesis

This thesis aims to perform resources estimation at the physical level for trapped-ion and superconducting quantum hardware platforms. To do this, we will extend the simulators infrastructure built around Q# to calculate the maximum number of physical qubits, the total number of physical gates, and the maximum runtime required to execute a particular quantum algorithm. Additionally, the accumulated error for the computation, based on the fidelity of the physical gates, will also be calculated and analyzed to provide more information about the feasibility of obtaining reliable results from specific hardware platforms.

*ToDo: Expand on the following structure of the thesis:

1. Implement an algorithm using fault-tolerant error-corrected gates.

2. For each hardware platform do the following:

(a) Use an open-system simulator to analyze and optimize the effectiveness of fault-tolerant error-corrected gates.

(b) Use a resource estimator to determine the maximum input size for the algorithm to run on a real NISQ device.

(c) Use a resource estimator to determine the characteristics that a device should have to run the algorithm for a specific input size, and determine what would be the runtime.

3. Compare the results for each hardware platform.

*ToDo: Consider using an algotithm with more circuit depth to allow error correction to play a more important role.

We chose the Bernstein-Vazirani algorithm to perform resources estimation on because it is simple and becasue the amount of resources it demands is proportional to the size of its input. This provides the opportunity to analyze how different hardware platforms scale.

Chapter 2

BASICS OF QUANTUM COMPUTING

2.1 Dirac Notation

Also known as bra-ket notation, Dirac notation provides a convenient way of expressing the vectors used in quantum mechanics.

Dirac notation defines two elements:

- The *ket* ($|\psi\rangle$), which denotes a column vector in a complex vector space that represents a quantum state.

$$|\psi\rangle = \begin{bmatrix} \psi_1 \\ \psi_2 \\ \cdots \\ \psi_n \end{bmatrix}$$

- The *bra* ($\langle\psi|$), which denotes a row vector that is the conjugate transpose, or adjoint, of a corresponding *ket* ($|v\rangle$).

$$\langle\psi| = \begin{bmatrix} \psi_1^* & \psi_2^* & \cdots & \psi_n^* \end{bmatrix} = \begin{bmatrix} \psi_1 \\ \psi_2 \\ \cdots \\ \psi_n \end{bmatrix}^\dagger$$

An *operator* $\hat{A}$, represented by a $n \times n$ matrix, acting on a *ket* $|\psi\rangle$ produces another *ket* $|\psi'\rangle$ such that the produced *ket* can be computed by matrix multiplication:

$$|\psi'\rangle = \hat{A}\,|\psi\rangle = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} = \begin{bmatrix} A_{11}\psi_1 + A_{12}\psi_2 \\ A_{21}\psi_1 + A_{22}\psi_2 \end{bmatrix}$$

The *inner product* is denoted as a bra-ket pair $\langle\phi|\psi\rangle$ and represents the probability amplitude that a quantum state ψ would be subsequently found in state ϕ:

$$\langle\phi|\psi\rangle = \begin{bmatrix} \phi_1^* & \phi_2^* \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} = \phi_1^*\psi_1 + \phi_2^*\psi_2$$

This notation also provides a way to describe the state vector of n uncorrelated quantum states, the *tensor product*:

$$|\phi\rangle \otimes |\psi\rangle = |\phi\rangle\,|\psi\rangle = |\phi\psi\rangle = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} \otimes \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} = \begin{bmatrix} \phi_1\psi_1 \\ \phi_1\psi_2 \\ \phi_2\psi_1 \\ \phi_2\psi_2 \end{bmatrix}$$

Note that $|\psi\rangle^{\otimes n}$ represents the tensor product of n $|\psi\rangle$ quantum states:

$$|\psi\rangle^{\otimes n} = |\psi\rangle \otimes \cdots \otimes |\psi\rangle = |\psi\rangle \cdots |\psi\rangle = |\psi \cdots \psi\rangle = \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} \otimes \cdots \otimes \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix}$$

2.2 Quantum Systems

*ToDo: Explain the Schrodinger equation, describe what a quantum system is and mention basic postulates of quantum mechanics (similar to what Nielsen and Chuang explain).

2.3 Qubits

In classical computation and classical information, a bit is the fundamental building block. Analogously, in quantum computation and quantum information, a quantum bit or qubit is the fundamental building block.

Physically, a quibit is a two-level quantum-mechanical system. The polarization of a single photon and the spin of the electron are examples of such systems.

Mathematically, a qubit is a linear combination of states $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ where α and β are complex numbers known as amplitudes and $|0\rangle$ and $|1\rangle$ are the computational basis states.

When a qubit is measured, the result is either $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $|\beta|^2$. Since the probabilities must sum to one, the qubit's state is normalized:

$$|\alpha|^2 + |\beta|^2 = 1$$

The computational basis states, which are analogous to the two values (0 and 1) that a classical bit may take, form an orthonormal basis represented by the following vectors:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Using the previous definitions, we can see a qubit as a unit vector in two-dimensional complex vector space:

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

2.4 Superposition

The principle of quantum superposition states that the most general state of a quantum-mechanical system is a linear combination of all distinct valid quantum states. A qubit is an example of a quantum superposition of the basis states $|0\rangle$ and $|1\rangle$.

A concrete example of a qubit in superposition that has the same probability of being measured as $|0\rangle$ or $|1\rangle$ is the following:

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$$

2.5 Single-Qubit Operations

*ToDo: Explain how single-qubit gates work in detail and mention some of the most common ones.

2.6 Multi-Qubit Systems

2.7 Entanglement

2.8 Multi-Qubit Operations

2.9 Interference

2.10 Measurement

2.11 Quantum Advantage

*ToDo: Explain how quantum computers can solve a problem that a classical computer can't efficiently by exploiting superposition, entanglement, and interference.

Chapter 3

BASICS OF QUANTUM ERROR CORRECTION

This chapter explains the basic concepts to understand how quantum computations can be performed reliably in the presence of noise.

3.1 Noise and Error Correction

3.2 Bit Flip Code

3.3 Phase Flip Code

3.4 Shor Code

3.5 Fault-Tolerant Quantum Computation

3.6 Threshold Theorem

Chapter 4
QUANTUM ALGORITHM (BERNESTEIN-VAZIRANI)

The Bernstein-Vazirani algorithm[1] is a quantum algorithm that finds a secret string $s \in \{0,1\}^n$ given an oracle that implements a function $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = (x \cdot s)\ modulo\ 2$. The most efficient classical algorithm evaluates the function n times to find s. In contrast, this quantum algorithm only needs to evaluate it once.

4.1 Algorithm

*ToDo: Mathematically describe the algorithm.

4.2 Circuit Representation

*ToDo: Show the circuit representation of the algorithm.

4.3 Q# Implementation

The following code presents a simple implementation of the Bernstein-Vazirani algorithm in the Q# programming language:

*ToDo: Make the implementation more generic by receiving the oracle as on argument to the BernsteinVazirani operation.

*ToDo: Consider creating a Q# language option for lstlisting.

```
@EntryPoint()
operation BernsteinVazirani () : Unit {
    let secret = [One, Zero, One, One, Zero];
    use (qubits, aux) = (Qubit[Length(secret)], Qubit()) {
        X(aux);
```

```
        H(aux);
        ApplyToEach(H, qubits);

        // Oracle.
        for index in 0 .. Length(qubits) - 1 {
            if (secret[index] == One){
                CNOT(qubits[index], aux);
            }
        }

        ApplyToEach(H, qubits);
        let results = ForEach(M, qubits);
        ResetAll(qubits);
        Reset(aux);
    }
}
```

Chapter 5
ALGORITHM EXECUTION ANALYSIS FRAMEWORK

*ToDo: Briefly describe each step in the process

5.1 Open-System Simulator

*ToDo: Describe what is the purpose of using an open-system simulator and how it works.

5.2 Resources Estimation Strategy

The strategy we use for resources estimation is very similar to the one proposed by Soeken et al.[3]. The process is the following:

1. Implement a quantum algorithm using a high-level programming language (Q# in this case).

2. Verify the correctness of the implementation by executing the algorithm in a full state simulator.

3. Setup the simulator to estimate physical resources with the parameters that are specific to a hardware platform.

4. Analyze the results obtained from the resources estimator.

*ToDo: Explain the following

- Resources Metrics: Describe the values obtained from the resources estimator (gate count, runtime, accumulated error), and how they are calculated.

- Gate Decomposition: Describe why logical-level gates have to be decomposed into physical-level gates.

- Limitations: Describe the limitations that this resources estimation has in regards to runtime (sum of the runtimes of individual gates rather than the critical path), and types of computations (trouble with mixed states).

5.3 Extending Q# Simulation Infrastructure for Estimation of Physical Resources

Microsoft's Quantum Development Kit (QDK) supports the implementation of custom simulators that can be used to run Q# programs. We leverage this capability and implement a simulator that calculates the resources a quantum algorithm would require to be executed in a hardware platform with specific characteristics.

*ToDo: Explain the following

- QDK Custom Simulators: Describe how custom simulators are implemented using diagrams and code snippets.

- Software Architecture of Physical Resources Estimator Simulator: Describe the software architecture using diagrams and code snippets.

Source code of a working version can be found in in GitHub.

Chapter 6

TRAPPED-IONS HARDWARE PLATFORM

*ToDo: Briefly describe this quantum computing platform.

6.1 Native Gates and Platform Characteristics

*ToDo: Enumerate the native gates that this platform implements, its characteristics (fidelity, gate time), and how logical gates are implemented (using circuits to illustrate them).

6.2 Fault-tolerant analysis and optimization

*ToDo: Show the use of an open-system simulator to analyze and optimize the effectiveness of fault-tolerant error-corrected gates.

6.3 Resources Estimation Analysis

*ToDo: Show (using tables and/or plots) how resources escalate as the size and pattern of the input changes.

Example of ouput of resources used by the Bernstein-Vazirani algorithm using a secret string of size 5:

```
Ion Platform Resource Estimation
Bernstein-Vazirani
PHYSICAL LAYER
Total Statistics
____________________
Qubits: 6
Gate Count: 38
```

```
Time: 1175
Error: 0.3500000000000003

Gate Statistics
-----------------
R:
 - Count: 35
 - Time: 470
 - Error: 0.2300000000000002

XX:
 - Count: 3
 - Time: 705
 - Error: 0.1200000000000001
```

6.4 Analysis of Execution in NISQ Devices

*ToDo: Analyze what would be the maximum size (and difficulty of pattern) of the secret string that can be used in a current NISQ device based on this platform.

6.5 Analysis of Execution of Algorithm for Input of Specific Size

*ToDo: Use a resource estimator to determine the characteristics that a device should have to run the algorithm for a specific input size, and determine what would be the runtime.

Chapter 7
SUPERCONDUCTING HARDWARE PLATFORM

*ToDo: Briefly describe this quantum computing platform.

7.1 Native Gates and Platform Characteristics

*ToDo: Enumerate the native gates that this platform implements, its characteristics (fidelity, gate time), and how logical gates are implemented (using circuits to illustrate them).

7.2 Fault-tolerant analysis and optimization

*ToDo: Show the use of an open-system simulator to analyze and optimize the effectiveness of fault-tolerant error-corrected gates.

7.3 Resources Estimation Analysis

*ToDo: Show (using tables and/or plots) how resources escalate as the size and pattern of the input changes.

7.4 Analysis of Execution in NISQ Devices

*ToDo: Analyze what would be the maximum size (and difficulty of pattern) of the secret string that can be used in a current NISQ device based on this platform.

7.5 Analysis of Execution of Algorithm for Input of Specific Size

*ToDo: Use a resource estimator to determine the characteristics that a device should have to run the algorithm for a specific input size, and determine what would be the runtime.

Chapter 8

COMPARISON BETWEEN TRAPPED-IONS AND SUPERCONDUCTING HARDWARE PLATFORMS

*ToDo: Compare the results obtained from both hardware platforms and comment on the insights obtained.

Chapter 9

FUTURE WORK

*ToDo: Mention how this framework can be used to analyze and compare other hardware platforms.

BIBLIOGRAPHY

[1] E BERNSTEIN and U VAZIRANI. Quantum complexity theory. *SIAM journal on computing*, 26(5):1411–1473, 1997.

[2] Cirq Developers. Cirq, May 2021. See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors.

[3] Mathias Soeken, Mariia Mykhailova, Vadym Kliuchnikov, Christopher Granade, and Alexander Vaschillo. A resource estimation andverification workflow in q#. *Design, Automation and Test in Europe Conference*, 2021.

[4] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, RWDSL2018, New York, NY, USA, 2018. Association for Computing Machinery.

[5] Matthew Treinish, Jay Gambetta, Paul Nation, Paul Kassebaum, qiskit bot, Diego M. Rodríguez, Salvador de la Puente González, Shaohan Hu, Kevin Krsulich, Laura Zdanski, Jessie Yu, David McKay, Juan Gomez, Lauren Capelluto, Travis-S-IBM, Julien Gacon, Ashish Panigrahi, lerongil, Rafey Iqbal Rahman, Steve Wood, Luciano Bello, Divyanshu Singh, Drew, Joachim Schwarm, MELVIN GEORGE, Manoel Marques, Omar Costa Hamido, RohitMidha23, Sean Dague, and Shelly Garion. Qiskit/qiskit: Qiskit 0.26.2, May 2021.

Appendix A

IMPLEMENTATION OF GENERIC PHYSICAL RESOURCES ESTIMATION FRAMEWORK

*ToDo: Add source code that implements the physical resources estimation framework.

Source code can also be found in GitHub.