Exploring Novel Methods of Quantum Error Correction (QEC) Through the Implementation and Application of Traditional Error Correction

Joy Kim, Yulhee Han, Aiden Oh
Supervised by Gangjoo Robin Nam and Taemin An

Abstract

Quantum computers are anticipated to have many applications if used properly among enterprises and governments. However, Quantum Errors remain a problem for quantum computing, as the computers are not yet fully immune to "noisy environments" including electromagnetic and particle disturbances. While Quantum Error Correction (QEC) stands to be the most feasible solution to quantum computer errors, the high level of knowledge required to comprehend such algorithms pushes many interested programmers and individuals away from interacting with quantum computers. The paper questions the applicability of traditional computational error correction codes such as Hamming Codes or Reed-Solomon Codes, as a concept and/or fully as in practice, as a novel method to make QEC possible without the use of redundancy qubits. The study compared the performance, accuracy, and efficiency of devised and conventional algorithms by executing integer multiplication and prime factorization. The results show that such hybridization of quantum hardware with Traditional Error Correction has fair merit, especially in execution time with operations with larger data, but may have compromises in accuracy unless highly frequent executions of error corrections are done with constant conversions from traditional to quantum superpositioned data and back, which may eliminate the purpose of quantum hardware. Nevertheless, the paper poses positive prospects for the development of such hybrid technology and suggests methods in which the research could be expanded for further knowledge and specific implementations of programs, apps, or codes to make such error corrections a reality.

I. Introduction

Following the trend surmised by Moore's Law from 1965, traditional computing has accelerated over the past few decades in terms of development: the density of transistors in integrated circuits has doubled every two to four years, posing an exponential speed in computing development. Unfortunately, this speed in developing computer circuits, which directly correlates to a computer's speed and efficiency, is flattening out. Due to physical limitations in gaps between transistors, the consensus is that the development of traditional computers will slow down in speed by the end of 2025. (Shalf) To this end, scientists have put hope in quantum computing, a new area of computing that vastly contrasts with traditional computing. The development of quantum computers has been consistently researched and discovered since the introduction of modern quantum theory. The exploration of wave-particle duality led to physicists applying quantum mechanical models in computation, developing the basic unit of quantum computation known as a "qubit" (quantum bit). The development of quantum computers made a breakthrough in the physically possible limit in computing

calculations: qubits used in quantum computers differ from traditional bits in that they are "superpositioned" rather than binarily assigned a value. Simply put, assigning and using a value from traditional computers is similar to flipping coins, as each bit is assigned a binary value of either 1 or 0 and remains the same. However, qubits are similar to spinning coins, as they could be both 0 and 1 before they are observed. Quantum superposition is possible due to the theory of particle duality, where particles could be at multiple states if not observed. Once observed, the particle assigns itself to a certain value or state deterministically according to the observer effect. (Shimony) In turn, this makes quantum computers suited for brute-force calculations that would take years using a traditional supercomputer.

Quantum computers are developing at exponential speeds, and the current progress of quantum computer development is astonishing. As of 2024, industries such as IBM and Google are successful in running 1,000 qubit QPUs in large operations. For example, in 2019, Google's quantum processor Sycamore successfully calculated a mathematical model that would take over 10,000 years in traditional computers in less than 200 seconds. (Frank Arute et al.)

Nevertheless, quantum computers are still barred from full performance and application outside large enterprises or governments. A significant reason behind this drawback is the incomprehensible level of sensitivity - vulnerability to errors - that they have to the environment. Because qubits in quantum computers need to maintain their quantum properties to function, the computer must be protected from the slightest external interferences - interferences as small as the movement of air particles and electromagnetic fields. (Sergey Bravyi et al.) Many attempts have been made to both make qubits immune to such sensitivity and error and find practical use of such qubits despite their vulnerability. Unfortunately, the currently developed "noisy intermediate-scale quantum" (NISQ) processors could only run with methods that can identify and correct innate quantum errors that occur during processes in quantum gates.

Another major limitation behind NISQ computing is the significantly low number of processors they could run in. Granted, NISQ computers have developed in size and accuracy. As of 2023, the most recent breakthrough in the scale of Quantum Computing Devices (QCDs) was the success of creating a 1000-qubit QCD, which was done by IBM. (Castelvecchi) However, this number is still stunted significantly from the ideal hardware for fault-tolerant quantum computing, as fault-tolerant QCDs require at least a million qubits to function properly. NISQ computing establishes significance in this aspect, as it maximizes the useability of quantum computers in the current stage of development. In this context comes the importance of Quantum Error Correction (QEC). QEC algorithms are dedicated to QCDs that detect and correct the error within a quantum bit string while not damaging it. Traditional Error Correction Codes (TECC) cannot be replicated completely in QCDs, as qubits are significantly fragile to damage in data (i.e. losing the qubit's quantum state) once directly observed. Fostering this novel path of algorithm development will make QCDs more immune to quantum noise and vulnerability, making results from faulty quantum computers useable even for advanced researchers. Not to mention, success in QEC will bring quantum computing closer from NISQ to fault-tolerant quantum computing - the ideal future of quantum computing, as depicted in Figure 1.

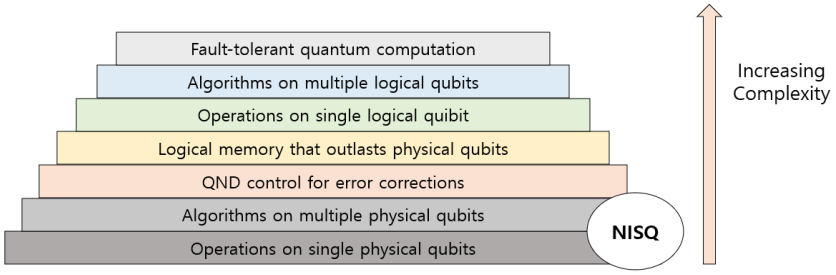


Figure 1- Step diagram of expected trajectory for quantum computer development *(Lau, Lim and Shrotriya)*

Indeed, this style of computing has limitations in expediting the process of quantum computing. The heavy focus on QEC in NISQ computing comes from the fact that it makes sure quantum algorithms and QCDs are tolerant to quantum errors and currently existing hardware limitations. Inevitably, this focus compromised the speed in developing and scaling up general algorithms that benefit from the structure of quantum computers. Alternatives have been suggested to redirect the current focus from developing optimal, accurate QEC methods to not being bothered by hardware limitations during quantum algorithm development. For instance, Bertels et al. introduced a new direction of coding known as Perfect Intermediate-Scale Quantum (PISQ) computing, which focuses on the development of quantum programs and algorithms without obstruction by hardware limitations or the need to develop QEC algorithms. (Bertels, Sarkar and Ashraf) This puts to the premise that quantum computing has enough budget and research at the current state to develop scalable quantum computers with error-free qubits. The method relies on the rationale that the quantum computing industry, which could only run on 1,000 noisy qubits, will mature itself in 10 years. Considering the current speed in QCD development, such an assumption is not far from idealistic. For context, quantum computing 20 years ago merely happened in labs with only 3 to 7 qubits in total, testing algorithms only as complex as bit-flip codes. (Zhang, Chen, and Diao)

Testing quantum algorithms, however, inevitably requires the use of current QCDs, which are imperfect and vulnerable to error, to say the least. To run simulations efficiently that would take years or decades in traditional computers, attaining an error-tolerant quantum computer is crucial, and before fault-tolerant quantum computers are guaranteed, the topic of developing QEC remains significant. Unfortunately, as the paper will discuss subsequently, QEC development both limits quantum computer development, to some extent, and is limited in useability from several perspectives. This paper will go over the fundamental logic and structure behind popular QEC algorithms to show the state-of-the-art QEC methods and, in turn, reveal potential flaws existing in the flow of development.

A qubit is in superposition if it has not been observed directly yet. This means the qubit is in a state with probabilities of being both 0 and 1. This could be represented with a column vector as $|\psi\rangle$, where the superposition could be represented as $\alpha|0\rangle + \beta|1\rangle$. Before collapsing to a

single state, this aspect makes qubits able to flexibly behave as a bit with a state of both 0 and 1, depending on the program's needs.

Another aspect of quantum computers is the fact that it has a phase of |+⟩ or |-⟩. The extra degree of phase enables quantum computers to operate in a complex vector state, which translates to the benefit of using the amplitude of qubit vectors with directions as well. This lets quantum computers encode information into a qubit in infinite ways, and it also lets qubits' amplitude be manipulated with interference, which is the reason behind the incomparable speed of quantum computers in mathematical operations.

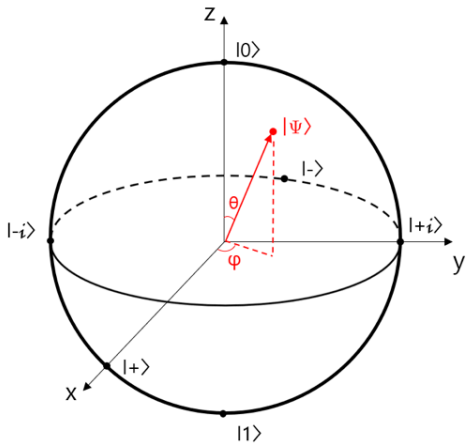


Figure 2 - Bloch Sphere representation of a qubit *(Shettell)*

Therefore, a quantum computer's logical gates are tailored to manipulate both states and phases of qubits. Generally speaking, the Pauli-X gate helps change the bit state probability of a qubit; the Pauli-Z gate helps change the phase vector of a qubit; and the Pauli-Y vector helps flip the quantum bit state but also with a phase change. The way qubit logical gates change a qubit vector could be more intuitively understood with a Bloch Sphere representation of a qubit vector, as seen in Figure 2. Each gate controls each 'axis' represented in the Bloch Sphere space. Another quantum logic gate to remember is the Hadamard gate, which gives a 'spin' to a qubit in a definite state. This makes the probability of a qubit's state of |0⟩ and |1⟩ balanced.

Pauli-X	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\|0\rangle \to \|1\rangle$ $\|1\rangle \to \|0\rangle$
Pauli-Y	$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$\|0\rangle \to i\|1\rangle$ $\|1\rangle \to -i\|0\rangle$
Pauli-Z	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\|0\rangle \to \|0\rangle$ $\|1\rangle \to -\|1\rangle$
Hadamard	$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$\|0\rangle \to \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$ $\|1\rangle \to \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$

Figure 3 - Matrices of the basic quantum logic gate and their manipulation of a qubit vector *(Deng, Zhang, and Zhang)*

Hence, a successful QEC algorithm must accurately and efficiently correct both bit-flip (X errors) and phase-flip (Z errors) in a QCD of qubits. This makes QEC function and has different objectives from traditional error correction algorithms like Hamming Codes, which simply need to check the number of faulty error bits and correct them from their state of 0 or 1 to the opposite; QEC has significantly more layers and complexities to the error correction process.

There is another problem lying with QEC development, and quantum algorithm development in general: interest and the number of interested individuals/students in quantum computers. Quantum computing in itself is gaining increasing interest in both enterprises and governments around the globe. The incomparable speed, energy efficiency, and possibility to decode advanced encryptions (e.g. RSA) make such institutions highly invested in the topic. Studies show that following the current trajectory, by 2030, the global quantum computing market is expected to surpass $65 billion with a compound annual growth rate (CAGR) of 32%. (D) (QuantumNews) Despite the popular demand for quantum computers, there is a significant talent shortage from a lack of individuals and students invested and fully educated, or at least committed, in quantum computing. (D) (Kaur and Venegas-Gomez) A McKinsey & Company report in 2022 confirmed that as of June of that year, only one candidate for every three jobs in quantum computing occupations was qualified. This correlates with the relatively significant lack of both demand and supply of education for quantum computers as of the current state, according to a Conference Proceeding in 2024. (Ebai, Schneider and Bächli) The genesis of the problem remains in the fact that High School education regarding computing, and physics in general, are mostly limited to non-existent in the realm of quantum computation or mechanics. QEC algorithms popularly used by scientists and researchers require advanced knowledge regarding quantum mechanics and physics, and the fact that students need to follow and learn the logic from a clean slate is not only challenging but intimidates many out of taking college courses related to quantum computing. (Sun, Q., Zhou, S., Chen, R. et al.)

Such existing problems lead up to a question: is there any method to incorporate traditional knowledge and techniques used for computer error correction when correcting errors from QCDs from quantum errors that can efficiently and accurately function in comparison to other QEC algorithms and traditional computers? When it comes to building an error correction algorithm different from currently existing QECs that function better than the current state-of-the-art algorithms, the paper acknowledges there is little to no chance of success. Throughout the past three decades, these QEC algorithms, including CSS Codes, Surface Codes, and Toric Codes, have significantly been optimized using complex mathematics and advanced knowledge of quantum mechanics and physics. Outperforming such algorithms using a novel method that ‘roundabouts’ to traditional error correction techniques is improbable – if not, impossible. Given the researchers’ occupation as Secondary and High School researchers with severely limited access to documents, knowledge, and technology pertinent to quantum computing, making observations more accurate than those from college-level researchers, scientists, and professors, is an idealistic sentiment.

Then what makes this question still valid for discussion of quantum computer development? The question of developing traditional ECC-inspired QEC algorithms, or incorporated algorithms, is present to ask useability to a basic extent. The paper's objective is not to make quantum computers run and correct errors at unprecedented speeds but to make the concept of error correction less verbose and more intuitive to average computer scientists less knowledgeable about quantum computing, thereby reducing the gap between their state of knowledge and access to quantum algorithms and replications of quantum programs that could offer results with a reasonable level of accuracy.

There is also a question in verifying the value of research even in the case of failure of structuring the algorithm/platform of a dream. Failure to create a TECC-inspired or TECC-incorporating QEC algorithm may lead to one, if not many, out of three responses. First, it would open a new perspective in quantum computing and error correction, and ideally speaking, future researchers by like-minded enthusiasts hopefully with more knowledge and professional access to quantum computers could develop on this paper's suggestion and attempt. Second, the failure will confirm the current direction of QEC development, substantiating the value of needing to involve advanced mathematics and mechanics to craft a functional, reasonable, and accurate QEC algorithm. Finally, for enthusiasts of quantum computing or quantum mechanics who similarly do not have much access or prior knowledge in QCD and quantum programming, this paper will serve as one of the platforms to learn the process of encoding and decoding quantum circuits and programs. The program surely will be an amateur attempt but will be a significantly more intuitive walkthrough of using quantum computers and running quantum programs for less knowledgeable individuals. Ultimately, both the success and failure of the experiment will lead to a positive outcome in which the paper serves as an approachable platform between amateur computer programmers and the field of quantum computers.

II. Method

As a disclaimer, the methods section for the paper will include Python codes and demonstrations and graph figures showing data inside the section while excluding C++ code snippets and programs for the Appendices. This is to improve the visibility of the section from lengthy program snippets in C++. Also, because C++ code snippets mostly cover replications of already-existing algorithms and the paper both explains and assumes the readers' preliminary amount of understanding of such algorithms, the paper sees the priority in showing the method to graphically represent such data instead of guiding steps to implementing traditional, well-known computer algorithms.

The first step to comparing the efficiency and performance of quantum to traditional computers is to choose the mathematical model (i.e. computer operation) that will be the base of measuring execution time. The amount of compromise that error correction offers in the two types of computers must be measured in percent differences in execution times rather than objective terms (e.g. milliseconds), as such measurement would mean significantly different

interpretations depending on the size and amount of operation done. A problem mentioned in the Introduction regarding QEC and traditional computer comparison in benchmarking is the lack of intuitive mathematical models that are understandable in applicability to non-enthusiasts of quantum computing (or computing in general). Hence, the paper chose math models accessible to the general public but with quantum algorithms developed that could fully use the advantage of quantum computer architecture: integer multiplication and prime factorization of large integers. The following paragraphs introduce specific parameters and environments in which this calculation was performed.

Traditional computer operations were done using Intel(R) Xeon(R) CPU E5-2686 v4. The specification of the CPU includes the following: clock speed of 2.3 GHz, 18 cores, 36 threads, average single thread rating of 1812, and multithread rating of 20415. (Intel Xeon E5-2686 v4 @ 2.30GHz) Background processes in the computer were isolated by using Ubuntu 24.04 ('Noble Numbat') LTS Server as the Operating System.

For quantum computer operations, the method uses IBM's Quantum Platform (formerly called IQX) and executes codes using the Python Language and its Qiskit Library. Because of problems in incompatibility of published Qiskit codes in open platforms such as GitHub or reputable research papers, the method uses Jupyter Notebook with older versions of kernel and Libraries, manually installing Qiskit 0.32.1 and Python 3.8.18 as the main system in the virtual environment. This environment is significantly older than the modern 1.x versions released by IBM, which leads to some limitations in the study. One of these limitations is the fact that actual quantum hardware cannot be used with API due to the incompatibility of the older Qiskit library and software. For this reason, the study natively uses a simulated quantum circuit environment (qasm_simulator) run by a Windows PC with Ryzen 3 5950X as the CPU and Nvidia GeForce 3090 as the GPU. However, the significant challenge in running a qasm_simulator program using a virtual environment in a server computer environment led to the research needing to use Windows as the environment for quantum computer simulations. Implications regarding this limitation will be discussed later, alongside the justification behind using such deprecated libraries.

This section of Methods discusses specific algorithms and methods used to replicate integer multiplication and measure the execution time on both quantum and traditional computers. Traditional computer algorithms for integer multiplication comprised of three well-known algorithms optimized multiplication of integers: Grade school, Recursive Karatsuba, and Toom-Cook (also known as Toom3) multiplication algorithms.[1] The three used algorithms have time complexity of $O\left(n^2\right)$, $O(n^3)$, and $O(n^{\frac{\log log(5)}{\log log(3)}})$, respectively. From Grade school multiplication, Karatsuba and Toom3 multiplication algorithms can earn more efficiency through divide-and-conquer approaches, where large integers are recursively divided into sub-strips of integers to add to each other. These algorithms, developed in the early 1960s, are well-known

[1] The C++ replications of these algorithms could be found at the Github Page at https://github.com/dev-grn/QandAI.git

methods of integer multiplication that take advantage of the fact that addition takes significantly less time in computers compared to multiplication operators.

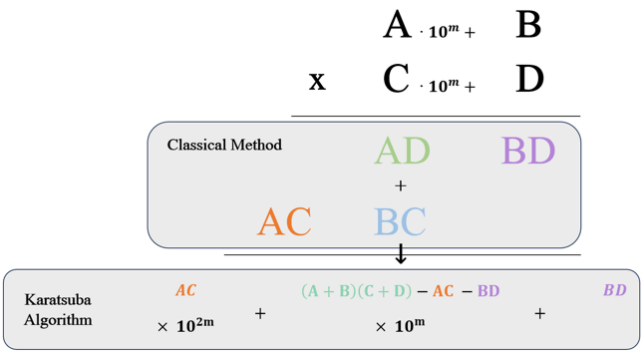


Figure 4 - Diagram representation of recursive Karatsuba multiplication algorithm

The program ran integer multiplication for two n-digit integers, where each integer would only be comprised of numbers from 1-9 using a Mersenne Twister (MT19937) pseudorandom number generator. 0's was excluded from the randomized number set to maximize the entropy of the operation. The program, for both quantum and traditional computers, was run until the most efficient algorithm within the n-digit multiplication would take over 1 minute from the used traditional computer. In the case of traditional computers, the performance in execution time, measured in milliseconds, was translated to a plot graph using Python Pandas and Matplotlib to intuitively represent a comparison. Figure 4 shows a combination of all three algorithms in execution time (the axis showing digit n and the y-axis showing time in milliseconds), and the subsequent figure shows a single graph with a collection of the minimum execution time out of the three programs.

With a lack of resources in previous attempts to replicate integer multiplication using the Qiskit library in quantum computers, the method heavily designs the quantum code based on a previous successful attempt of implementing a desktop calculator with quantum computers using Quantum Fourier Transform (QFT).[2]

For traditional computers as well, optimized algorithms of integer multiplication using Fast Fourier Transforms (FFT) exist. For C++, libraries such as the GNU Multiple Precision Arithmetic Library exist that perform arithmetic operations such as integer multiplication in optimal time with a combination of FFT. There are a couple of reasons why the method does not use this state-of-the-art algorithm for integer multiplication in traditional computers. Firstly, using divide-and-conquer methods such as Toom3 or Karatsuba algorithms that apply recursive functions becomes significantly more intuitive and approachable to traditional computer programmers in contrast to using FFT, which requires some level of preliminary mathematical knowledge. Secondly, the execution time of integer multiplication using the GMP library leads to

[2] Access the referred project via Github: https://github.com/lmarza/QuantumDeskCalculator.git

significantly low, negligible execution times that are significantly hard to differentiate for the digits of n the devised algorithms in this method ran in.

Even if FFT algorithms could be implemented to significantly large integers to the extent of which noticeable differences happen, such numbers are not calculable for currently accessible quantum computers made by IBM. The goal of this research includes creating replicable results that are free to access by walking through and executing the mentioned codes in the paper. The default given limit of time in executing a quantum computer with IBM Quantum Platform is 10 minutes, with each additional minute taking an extra over $90. Given the resources and compromises, the researchers concluded that the best option to show a difference in the performance of traditional and quantum computers is to use the most well-known, accessible algorithms, which are recursive divide-and-conquer functions for traditional computers and QFT for quantum computers.

QFT is the optimized version of the algorithm for quantum computers performing Discrete Fourier Transforms (DFT). For DFT in traditional computers, takes the time complexity of $O(n^2)$. However, using quantum superposition and characteristics of QCD architecture, QFT could achieve this at exponentially faster speeds. QFT is a reversible algorithm, which means it can not only convert a quantum state into a frequency domain but also the opposite. The method makes a superposition of quantum states in the frequency domain through a continued series of Hadamard and Controlled phase-shift gates. QFT helps detect periodicities and symmetrical patterns within data sets, which is tedious and complex in practice for traditional computers.

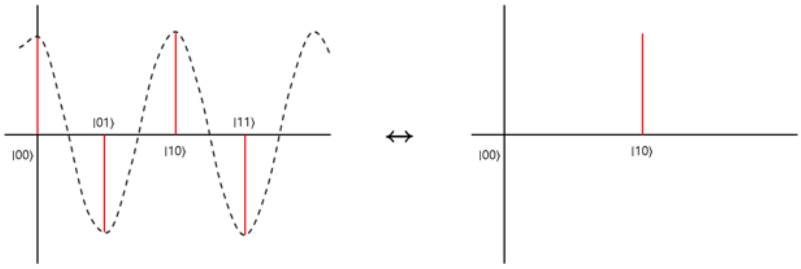


Figure 5 - Depiction of a Quantum Fourier Transform

Equation 1 - Mathematical definition of QFT

$$\sum_j \alpha_j |j\rangle \rightarrow \sum_k \tilde{\alpha}_k |k\rangle, \quad \text{where} \quad \tilde{\alpha}_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k / N} \alpha_j$$

As a side note, this is the mathematical definition of a Quantum Fourier Transform. However, in practice for the paper's purposes, only a tangential understanding of a QFT and its

applications are required, as Python's Qiskit library offers the ability to abstract QFT operations to functions less convoluted and easier to understand.

Not only does QFT, according to the nature of Fourier Transforms, make integer multiplication more efficient and faster, but the ability of the method to find periods also makes it have practical implications in algorithms requiring the detection of periods such as Shor's algorithm, an algorithm dedicated for QCDs when factoring large integers. Using QFT in Shor's algorithm will further be explained in the later part of the Methods section.

For the first 5,000 digits of n for integer multiplication, 100 trials were run for each algorithm in each value of n, since the low execution time for the algorithm meant a higher level of change in numbers with potential delays in operations from the computer. Using the redundant data, outliers were automatically deleted in case of significant deviation from the calculated IQR. After 5,000-digit multiplication, 20 trials were run for each digit. Note that these numbers are arbitrary, subjective standards set by the researchers. However, in the case of attempting to replicate the data, conducting more or fewer trials in isolated conditions will likely offer the same trend in numbers (despite a potential difference in execution time if run by a different OS or CPU).

Prime factorization of large integers serves as an accurate assurance factor of the found results. Unlike large integer multiplication, prime factorization is well-discovered in optimization and algorithmic development for both traditional and quantum computing. While integer multiplication for quantum computers requires devising a new, less-optimized algorithm relying only on QFT, and traditional computers use recursion, landmark algorithms exist for both types of computers in prime factorization.

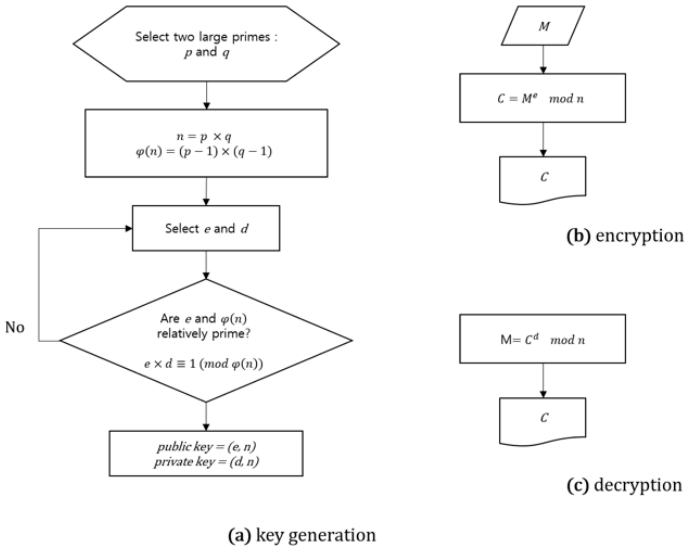


Figure 6 - Flow chart representation of RSA encryption and decryption using prime factorization *(Aljuaid and Gutub)*

Prime factorization also has significant applications in both computers in large enterprises and, more importantly, governments. In cryptography, when encrypting and decrypting keys in RSA encryption, the method heavily relies on the product of two large prime numbers. To crack RSA, the user must be able to factorize the large integers, and using quantum computers for decryption will be revolutionary in practical and correct application, as it has the potential to complete the process at speeds incomparable to traditional computer algorithms. (Omollo and Okoth) (Hoffstein) (Rutkowski and Houghten) Using integer factorization to compare the performances of quantum and traditional computers will be more useful in direct terms regarding the useability of quantum computers in such practical fields of cryptography.

To explain the algorithms involved in both computers, the method will first introduce Pollard's Rho algorithm, which was the method used to prime factorize large integers in traditional computer codes.

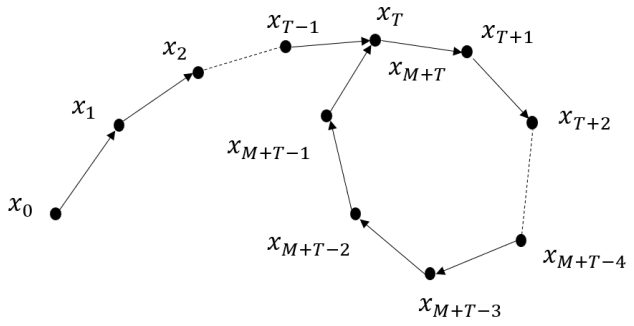


Figure 7 - Diagram representation of Pollard's Rho algorithm *(Deepthi and Hegde)*

Developed by John Pollard in 1975, the algorithm is one of the most well-known prime factorization algorithms in traditional computers that completes its operation in the time complexity of $O(\sqrt{n})$. Among traditional computers, the algorithm is used to break complex algorithms such as RSA that rely on the product of two large prime numbers. Pollard's Rho algorithm achieves this efficient time complexity by searching a cycle among an iteration sequence orbiting a function f. After a cycle is found, the program finds the Greatest Common Divisor (GCD) to see the relation between the suggested cycle and a factor of the number n.

The C++ code replicating Pollard's Rho algorithm considered that current Qiskit Codes and, more importantly, IBM's publicly accessible quantum computers have little to no support or instructions on using quantum computer memories. To prevent the digression of the research into verbose quantum memory allocations of data, the program takes into the premise that the largest integer that a quantum computer could calculate directly relates to the number of qubits inside the QPU. To maximize the size of the number that the program could factorize, the code uses __int128 datatype instead of conventional unsigned long long integers. Through Python3, the program draws an exponential regression line of the plots on the data's average points to find a

trend in the increase of execution time with an increase in the size of the dataset by adding a single digit for the number n.

```
import numpy
import matplotlib.pyplot as plt

#x, y elements could freely be changed.
x =
[3,6,10,13,16,20,23,26,30,33,36,40,43,46,49,53,56,59,63]
y = [16.2, 53.2, 91.8, 131.7, 171.9, 216.6, 262.1, 312.9,
375.3, 457.3, 605.0, 799.9, 1052.2, 1382.3, 1951.4, 2815.2,
4084.2, 6356.8, 9424.3]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(1, 70, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Note that the Python code above imported data from Pollard's Rho algorithm in the array y_data. Using the program for experimentation will require readjustment of the arrays x_data and y_data depending on the size and scale of the data set.

For every integer *N* that was factorized, the digits were randomized between 1-9. The time, represented in microseconds, shows the amount of time taken to execute Pollard's Rho algorithm for the number *N* 10,000 times, excluding the time taken to generate the number. To connect the plots of executed time for every value of *d*, the number o digits, the trend shows an exponential trajectory in comparison to the expected sub-exponential trajectory according to the Big O notation of Pollard's Rho algorithm. While this may seem deceiving initially, the representation is plausible in growth rate because the sub-exponent, ½, combined with the rate of change of *d*, turns the net exponent of the equation to a growth rate of around 1.2 to 1.8.

To implement prime factorization in quantum computers, the method uses the well-known Shor's algorithm. Developed by Peter Shor, this algorithm finds prime factors of large integers, searching non-trivial factors of a number at speeds incomparable to that of traditional computers. Let the large integer being factorized be named *N*. The algorithm first starts by using a random number *x* to define the function $f(x) = x^a \, mod \, N$. After an integer *r*, the function is bound to repeat and have a period. In this situation, a quantum computer can efficiently use QFT and Quantum Period Finding (QPF) techniques to collapse the result into data related to the number *r*. When *r* is found, the program uses classical post-processing algorithms by checking if $\left(x^{\frac{r}{2}} - 1\right)$ or $(x^{\frac{r}{2}} + 1)$ offering non-trivial factors of the number *N*, assuming that *r* is a positive integer.

The benefit of Shor's Algorithm using quantum computers comes from the fact that QFT and QPF algorithms are incomparable in speed and efficiency in comparison to popular traditional prime factorization algorithms including Pollard's Rho algorithm. Even algorithms claimed more efficient than Pollard's Rho algorithm including quadratic sieve or Dixon's

algorithm is, at most, with sub-exponential time complexity. In stark contrast, Shor's algorithm has the advantage of completing the procedure in polynomial time, which means as the integer to factorize becomes larger, the benefit of the quantum algorithm over the traditional algorithm shows more explicitly.

The goal of using Shor's algorithm for this comparison is to check whether using the new QEC method still offers a significant advantage in prime factorization through quantum computers while still offering tolerable data accuracy.

The use and implementation of Shor's algorithm reveal the main reason why the method uses older versions of Qiskit and Python versions in the virtual environment. During the initial stages of IBM Qiskit development (0.x versions), IBM used to support the AQUA Library within Qiskit, which includes various easy-to-use functions dedicated to complex yet famous and well-used quantum algorithms such as the Grover Search Algorithm and QPF. One of those algorithms was Shor's algorithm. However, this feature was deprecated in the process of developing Qiskit from 0.x to 1.x versions, which meant that using AQUA was not possible using the newest Qiskit libraries. Due to limitations in time as well as scholarly materials for the student researchers to acquire knowledge to fully comprehend and create a Qiskit implementation of Shor's algorithm, the feasible solution was to use deprecated libraries to use the Shor function within AQUA. By using AQUA's Shor library, the researchers were able to devise the following Python Code that could easily execute Shor's algorithm given a semiprime number *N*.

```
from qiskit import *
from qiskit.aqua.algorithms import Shor
from qiskit.aqua import QuantumInstance
from qiskit import IBMQ, assemble, transpile
from qiskit import Aer
import datetime

#key, base values could be changed.
key = 77
base = 2
backend = Aer.get_backend('qasm_simulator')
qi = QuantumInstance(backend=backend, shots=1024)
shors = Shor (N=key, a=base, quantum_instance = qi)

a = datetime.datetime.now()
results = shors.run()
b = datetime.datetime.now()
#b-a is measured in seconds. Additional conversion to microseconds is required.

print(b-a)
```

The next step for the method is to measure the time it takes to detect and correct errors in both quantum and traditional computers. For a couple of reasons, measuring time for ECC was done separately for both quantum and traditional computers instead of incorporating them into each program for integer multiplication and prime factorization. Explaining in terms of traditional computers, which mostly reflect the case of quantum computers as well, using ECC is

the same method for any math model or procedure, as the algorithm strictly observes the result data rather than being within intermediate steps of the program. For such a scenario, using the same data and estimating time to add to the two algorithms is more accurate, as every observation of execution time for a computer code offers more room for error in small units of time, which is crucial for measuring time to correct errors within bit strings. Furthermore, for transparency and the ability for the code to intuitively be replicated even outside of the two used mathematical models, the method separates the process of error correction so future researchers could use the paper's measured data to add to the execution time of their programs that may be both in forms of replicating the paper's codes or different programs. This expands the applicability and value of the research.

```
int main(){
ull limlen; int err = 5; //Set error rate to 5(%)
cout << "Input the length limit of the data : "; cin >> limlen;

string fForm = fileName + ".csv";
ofstream file(fForm, ios::app);

Data.push_back(-1);
Hamming(); Data.clear(); //Run Hamming Code once in the prepration phase
//(To increase the accuracy of time measurements)

for(ull len = 1; len<=limlen; len++)
{
string x = generate(len), y = generate(len);
const char *a = x.c_str(), *b = y.c_str();
mpz_t P, Q, R;
mpz_init(P); mpz_init(Q); mpz_init(R);
mpz_set_str(P, a, 10); mpz_set_str(Q, b, 10);
mpz_mul(R, P, Q); //Calculate the multiplication of two generated numbers

char *c = mpz_get_str(NULL, 10, R);
string alpha = c;
mpz_class L(alpha);
string result = L.get_str(2); //Convert calculated values to string form

Data.push_back(-1);
ull length = result.size();
for(int i = 0; i<length; i++) Data.push_back(result[i] - '0'); //Move to vector

system_clock::time_point Hs = system_clock::now(); Hamming();
system_clock::time_point He = system_clock::now();
//Measure Hamming Code execution time

system_clock::time_point Es = system_clock::now(); Cause_Error(err);
system_clock::time_point Ee = system_clock::now();
//Measure error generation time

system_clock::time_point Rs = system_clock::now(); Reception();
system_clock::time_point Re = system_clock::now();
//Measure the time to analyze the result

nanoseconds Hdur = He - Hs, Edur = Ee - Es, Rdur = Re - Rs;
file << Hdur << "," << Edur << "," << Rdur << endl; //Enter results in file

Data.clear(); Transmit.clear();
}
```

```
return 0;
}
```

For the sake of this paper's research, the Hamming Time added will regard the first graph in Fig. 12, as comparing the error correction time with quantum computers means that the comparison assumes errors naturally happen for both quantum and traditional computers.

The traditional ECC-inspired QEC will take the following steps to attempt to correct quantum errors. Firstly, instead of using super-positioned qubits to indirectly correct qubits, the method will observe the qubits to determine the state and then use the Hamming Code to correct the bit string. Because this goes against the traditional premise that quantum error correction is only possible through indirect manipulation of errors, the error correction method will be measured not only in execution time but also in accuracy. This is because unlike Hamming Codes, which provide near 100% accuracy unless significant damage is made to the bit string in the same pairs of bits, the algorithm has never been discovered in accuracy and useability. Both the compromise in execution time and accuracy of the error correction method will be measured in percent compromise to maximize the applicability of the results regardless of the size of the data size used. For compromise in execution time, the two computers will both be measured and compared intuitively with a graphical representation. For quantum error correction, the method will compare the accuracy and time required for Steane Code, a parity-checking QEC algorithm. Below is a quick explanation of the Steane Code algorithm along with an explanation of the method's choice of Steane Code as the primary algorithm of comparison against the two other types of error correction algorithms.

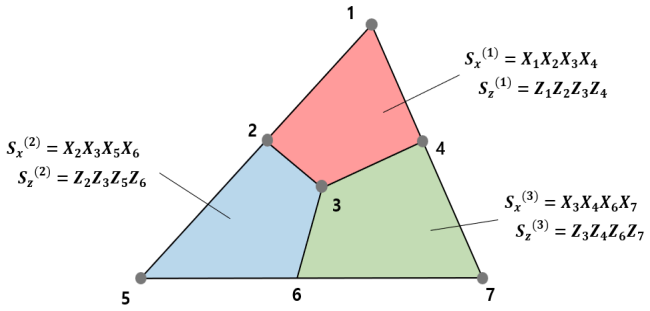


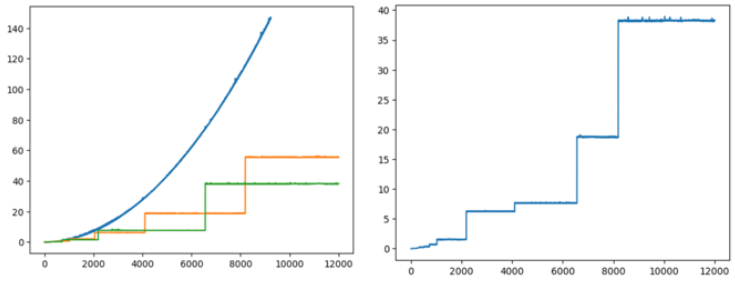
Figure 8 - Diagram representation of encoded logical qubits for Steane Code QEC *(Martínez-García, Vodola, and Müller)*

Developed by Andrew Steane in 1996, the algorithm uses parity checks with ancilla qubits that take the role of indirectly observing the state of the encoded qubit. The algorithm was inspired by the Hamming Code for creation, as both algorithms use parity checks to monitor the damage of data. (Steane) The algorithm corrects both bit-flip and phase-flip errors instead of collapsing the qubits into a state and correcting the finalized data string. The algorithm works by

first encoding the information of a qubit into seven physical qubits, known as logical qubits. This redundancy of qubits will serve the purpose of adding accuracy and assurance to the information the logical qubits hold, which are easily vulnerable to quantum errors just like the encoded physical qubit does. Using specific properties, called stabilizers, of the encoded qubits, the logical qubits measure and detect any location or occurrence of bit-flip and phase-flip errors. To recover the information of the corrupt qubit, the algorithm restores it to the original state.

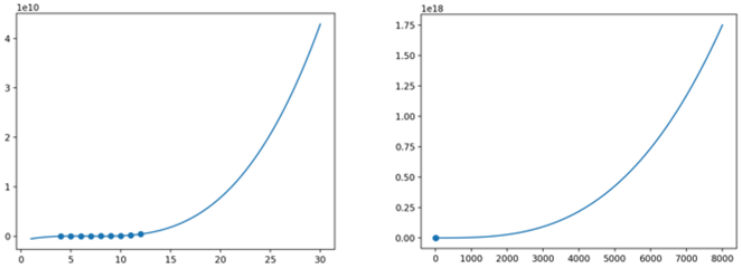
Relevance and simplicity were the two main reasons for choosing the Steane Code algorithm as the primary QEC algorithm to compare the two other ECCs against. As mentioned above, Steane Codes were heavily inspired by traditional ECC unlike other topological error correction codes, which are vastly different in logic compared to traditional computers. This would make the Steane Code a great target of comparison to the suggested fusion of traditional ECC with qubit data, as both use concepts of parity checks and root from logic posed by the Hamming Code. Furthermore, the algorithm is more 'well-supported' in replications and uses general quantum computer scientists, which reduces the obscurity of the algorithm. By this, the researchers refer to the subjective accessibility of attempts by regular programmers to use and benchmark a QEC algorithm in open platforms including GitHub. Surface codes, which are significantly more efficient and use the advantages of quantum computer architecture to their fullest, are unfortunately complex in both used logics and are also obscure in the Web when it comes to attempted uses or implementations of the algorithm in publicly available QCD programming environments (such as Azure Quantum, IBM Quantum Platform, which use languages and libraries such as Python's Qiskit Library or QASM).

III. Results



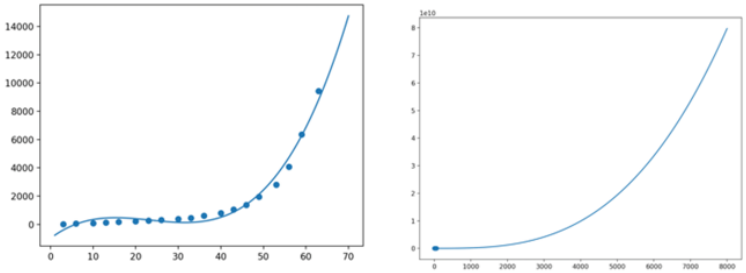
(a) (b)

Figure 9 - Execution time (μs) of big integer multiplication
(a) Blue = Gradeschool, Green = Karatsuba, Orange = Toom3 (b) Minimum t values



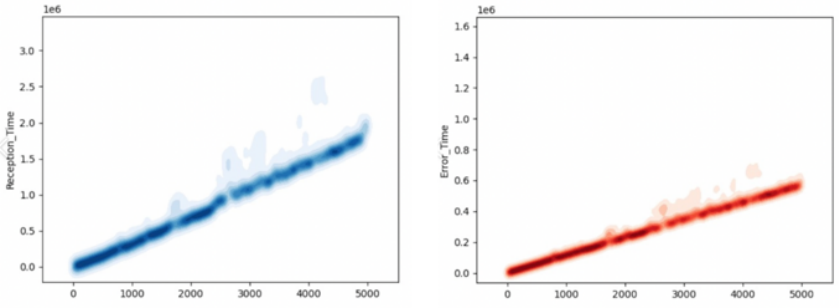
(a) n=30 (b) n=8,000

Figure 10 - Execution time (µs) scatter plot and regression line for Shor's algorithm



(a) n=70 (b) n=8,000

Figure 11 - Execution time (µs) scatter plot and regression line for Pollard's Rho algorithm



(a) Reception & Correction (b) Encoding

Figure 12 - Execution time (µs) data distribution of Hamming Code

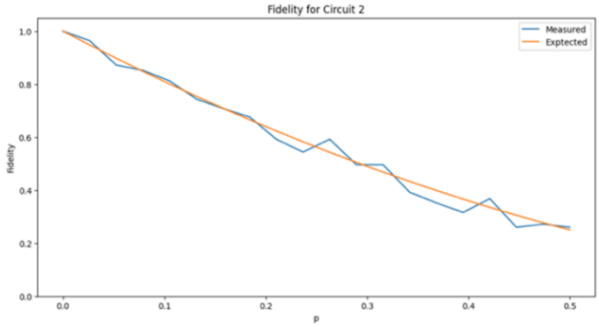
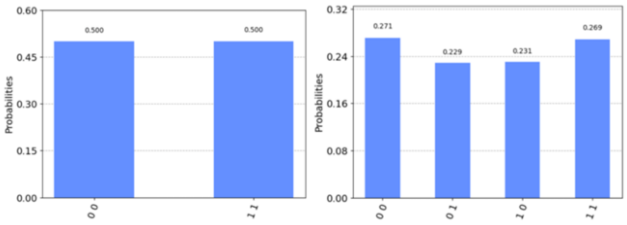


Figure 13 - Expected vs. Actual fidelity of the simulated quantum circuit using qasm_simulator



(a) (b)

Figure 14 - Probability distribution histogram of the quantum circuit from |00> to |11>
(a) Without error rates (b) With bit and phase error for each quantum gate, undergone QEC

IV. Discussion

A point worthy of discussion regarding the trends found in the Results section is how the trend in the calculation of prime factorization and integer multiplication all show a near-exponential trend (where the ratio between adjacent data points rises significantly. This differs from the trend expected from the Big O analysis of the time complexity of these algorithms, as they were expected to show a logarithmic trend. The difference in such data may be attributed to the limitations and considerations of program execution in actual computer hardware: while these analyses show the theoretical efficiency of the program, it does not take into consideration the fact that larger and longer calculations stress the computer performance and memory increasingly. Further studies will need to be done to verify the results, however, as the near-inverse trend found in the execution time of such calculations seems quite anomalous from the initial expectations.

Compared to the discussed trend of computational algorithms, the three graphs showing the trend of creating, encoding, and correcting in the Hamming Code algorithm, as shown in Figure 12, all show a linear trend. This implies that the relative impact of error correction execution time on program execution will decrease as the size of the data used increases.

The probability distribution of the quantum circuit before and after the error correction suggests a different story regarding the feasibility of error correction. Compared to the expectations of the authors, the research data showed a significant distribution of probability in data even after QEC. After the initial data distribution in which the states of |00> and |11> have 0.500 probability each, the operation decreased their probabilities down to nearly 0.275. The significant decrease in this probability proves that error correction must be frequently implemented during operations if worked with traditional, physical qubits, to extract and operate with useable, correct data. This poses a possible limitation in which the highly frequent need for conversion from traditional to quantum data and reverse if correcting quantum errors with traditional ECC will not only lead to a significant increase in execution time but also a higher chance of error rate during data conversion and transmission.

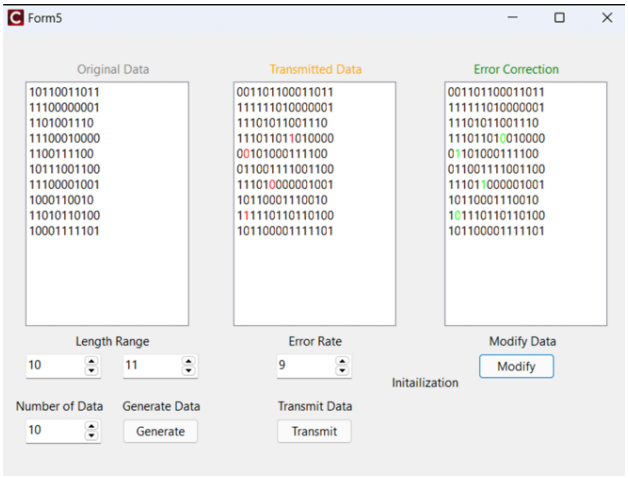


Figure 15 - Graphical User Interface (GUI) of implemented C Builder Program to simulate Hamming Code-incorporated QEC

Another way in which this discussion could be extended is to foster the development of an app (or a GUI-incorporated program) that would show the process of error correction. The research used C Builder, as shown in Figure 15, to attempt to build a brief application simulating error rates and showing the accuracy in error correction. The method successfully executed error codes and corrected them for Hamming Codes. The implication of this semi-study within the research is that the same application for QEC could be implemented if the proper technology, programs, and methods for hybrid QCDs are researched with enough knowledge. Future studies could expand this attempt and discussion by creating programs that convert quantum superpositioned data appropriately to binary data, which could be used for traditional ECC execution. These applications could also be used to intuitively represent the process of error correction and show the user the accuracy of error correction of their data.

V. Conclusion

The study was able to unravel different points of merit and limitations regarding the technology of 'hybrid' quantum computers incorporating traditional ECC. Compared to the Big O analyses showing logarithmic trends of execution time with an increase in data size for optimized algorithms conducting integer multiplication and prime factorization, both showed a trend of the opposite, where the rate at which the execution time increased continued to increase with the increase in data size. As discussed partly in the Discussion section, the odd trend in data may be attributed to limitations in the researchers' ability to confine the experimental conditions and environments, causing throttling and memory allocation issues with large data sets that inadvertently led to longer execution times. Though further studies need to be conducted to find the cause of such a trend, if these results are verified true for real-life applications, this suggests an interesting result. It means that at a certain point of data size, the relative portion of execution time that Hamming Code (or most forms of traditional ECC programs) will be trivial compared to the program execution time. This is because as seen in Figure 12 showing the execution time for creating, encoding, and correcting errors, the trends are all linear as opposed to exponential.

Therefore, it means that the program execution time increases with a more rapid rate of change eventually compared to that of Hamming Codes. This may mean that these hybrid machines will show more advantage (i.e. less compromise in time while same merit in accuracy) as the size in operation they handle becomes large enough to a certain point. Examinations regarding the existence and specific aspects of this point must be further discovered and researched.

The research discovered that the most important point future attempts to make such hybrid technology must consider is the frequency of converting quantum data to traditional data (and vice versa) to execute error correction. As discovered by examining a simulated quantum circuit environment, the accuracy of data after the circuit undergoes quantum gates faulty bit and phase errors significantly decreases the accuracy, even with Steane Code error correction. Though the research was conducted using a simulated circuit rather than an actual IBM computer, the fidelity of the circuit (shown in Figure 13) signifies that the circuit mostly resembles the actual QPU hardware, shown by the near alignment of the fidelity trend of the simulated circuit and the actual IBM QPU. This implies that the compromise in time will likely come most from the high frequency of error correction instead of the sheer execution time of individual error correction cycles.

(a) (b)

Figure 16 - (a) Image of IBM's QCD (b) Image of IonQ's QCD

A potential solution may relate to the rise in quantum hardware using 'logical qubits' instead of faulty physical qubits. Industries such as IonQ that use Ion Trapping technology use mechanisms different from those used in larger corporations such as Google and IBM. While these large industries focus on increasing the sheer number of qubits the hardware functions on to maximize the space for redundancy and ancilla qubits for error correction, QPUs developed by industries like IonQ have individual qubits with higher accuracy. With this quantum hardware, error corrections will likely be less required due to their fault-tolerant architecture and resilience to external interferences such as temperature. Therefore, the implementation of hybridizing quantum and traditional hardware may show more feasibility in this Ion-Trapped quantum hardware, which should be explored further in the area and examined in performance and accuracy both objectively and relatively compared to the IBM hardware experimented in the current study.

Works Cited

Aljuaid, Nouf and Adnan Gutub. "Combining RSA and audio steganography on personal computers for enhancing security." SN Applied Sciences 1.8 (2019).

Bertels, Koen, Aritra Sarkar and Imran Ashraf. "Quantum Computing—From NISQ to PISQ." IEEE Micro 41.5 (2021): 24-32.

Castelvecchi, Davide. IBM releases first-ever 1,000-qubit quantum chip. 4 December 2023. Nature. November 2024. <https://www.nature.com/articles/d41586-023-03854-1>.

D, Kyrlynn. Quantum Computing for 2024: What you need to know to get a Job. 31 May 2024. Quantum Zeitgeist. November 2024. <https://quantumzeitgeist.com/quantum-computing-for-2024-what-you-need-to-know-to-get-a-job/>.

Deepthi, P. and Nagaratna P. Hegde. "Comparison of Pollard's rho Algorithm Based on Cycle Finding Methods." Springer. New Delhi, 2022.

Deng, Zhiliang, et al. "Privacy-preserving quantum multi-party computation based on circular structure." Journal of Information Security and Applications 47.2 (2019): 120-124.

Ebai, Franka Ebob Enow, et al. "Quantum Computing as Uprising Topic for Business Students in Higher Educational Institutions." The Future of Education 14th Edition. Florence, 2024.

Frank Arute et al. "Quantum supremacy using a programmable superconducting processor." Quantum supremacy 574.7779 (2019): 505-510.

Hoffstein, Jeffrey. Integer Factorization and RSA. New York: Springer, 2008.

Intel Xeon E5-2686 v4 @ 2.30GHz. 31 October 2024. November 2024. <https://www.cpubenchmark.net/cpu.php?cpu=Intel+Xeon+E5-2686+v4+%40+2.30GHz&id=2870>.

Kaur, Maninder and Araceli Venegas-Gomez. "Defining the quantum workforce landscape: a review of global quantum education initiatives." Optical Engineering 61.8 (2022).

Lau, Jonathan Wei Zhong, et al. "NISQ computing: where are we and where do we go?" AAPPS Bulletin 32.27 (2022).

Martínez-García, Fernando, Davide Vodola and Markus Müller. "Adaptive Bayesian phase estimation for quantum error correcting codes." New Journal of Physics 21.12 (2019).

Omollo, Richard and Arnold Okoth. "Large Semi Primes Factorization with Its Implications to RSA Cryptosystems." BOHR International Journal of Smart Computing and Information Technology 2.1 (2021): 1-8.

QuantumNews. Quantum Computing Jobs The Careers of the Future. 18 September 2024. Quantum Zeitgeist. November 2024. <https://quantumzeitgeist.com/quantum-computing-workforce-development-quantum-computing-job-market-trends/>.

Rutkowski, Emilia and Sheridan Houghten. "Cryptanalysis of RSA: Integer Prime Factorization Using Genetic Algorithms." IEEE. Glasgow, 2020.

Sergey Bravyi et al. "High-threshold and low-overhead fault-tolerant quantum memory." Nature 627.8005 (2024): 778-782.

Shalf, John. "The future of computing beyond Moore's Law." The Royal Society Publishing 378.2166 (2020).

Shettell, Nathan. "Quantum Information Techniques for Quantum Metrology." ArXiv. Paris, 2022.

Shimony, Abner. "Role of the Observer in Quantum Theory." American Journal of Physics 31.10 (1963): 755-773.

Steane, Andrew. "Error Correcting Codes in Quantum Theory." Physics Review Letter 77.5 (1996).

Sun, Q., Zhou, S., Chen, R. et al. "From computing to quantum mechanics: accessible and hands-on quantum computing education for high school students." EPJ Quantum Technology 11.58 (2024).

Zhang, Zhigang, et al. NMR Quantum Computing. Boston: Springer, 2009.