Towards Complete Emulation of Quantum Algorithms using High-Performance Reconfigurable Computing

By
Naveed Mahmud

Submitted to the graduate degree program in EECS and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____

Chair: Dr. Esam El-Araby

_____

Dr. Prasad Kulkarni

_____

Dr. Perry Alexander

_____

Dr. Heechul Yun

_____

Dr. Tyrone Duncan

Date Defended: July 1st, 2022

The dissertation committee for Naveed Mahmud certifies that this is the approved version of the following dissertation:

Towards Complete Emulation of Quantum Algorithms using High-Performance Reconfigurable Computing

Chair: Dr. Esam El-Araby

Date Accepted: July 1st, 2022

Abstract

Quantum computing is a promising technology that can potentially demonstrate supremacy over classical computing in solving specific classically-intractable problems. However, in its current nascent stage, quantum computing faces major challenges. Two of the main challenges are quantum state decoherence and low scalability of current quantum devices. Decoherence is a process in which the state of the quantum computer is destroyed by interaction with the environment. Decoherence places constraints on the realistic applicability of quantum algorithms as real-life applications usually require complex equivalent quantum circuits to be realized. For example, encoding classical data on quantum computers for solving I/O and data-intensive applications generally requires complex quantum circuits that violate decoherence constraints. In addition, current quantum devices are of intermediate scale, having low quantum bit (qubit) counts and often producing inaccurate or noisy measurements. Consequently, benchmarking of existing quantum algorithms and the investigation of new applications are heavily dependent on classical simulations that use costly, resource-intensive computing platforms. Hardware-based emulation has been alternatively proposed as a more cost-effective and power-efficient approach. Hardware-based emulation methods can take advantage of hardware parallelism and acceleration to produce results at a higher throughput and lower power requirements.

This work proposes a hardware-based emulation methodology for quantum algorithms, using cost-effective Field Programmable Gate Array (FPGA) technology. The proposed methodology consists of three components that are required for complete emulation of quantum algorithms; the first component models classical-to-quantum (C2Q) data encoding, the second emulates the behavior of quantum algorithms, and the third models the process of measuring the quantum state and extracting classical information, i.e., quantum-to-classical (Q2C) data decoding. The proposed

emulation methodology is used to investigate and optimize methods for C2Q/Q2C data encoding/decoding, as well as several important quantum algorithms such as Quantum Fourier Transform (QFT), Quantum Haar Transform (QHT), and Quantum Grover's Search (QGS). This work delivers contributions in terms of reducing complexities of quantum circuits, extending and optimizing quantum algorithms, and developing new quantum applications. For example, decoherence-optimized circuits for C2Q/Q2C data encoding/decoding are proposed and evaluated using the proposed emulation methodology. Multi-level decomposable forms of optimized QHT circuits are presented and used to demonstrate dimension reduction of high-resolution data. Additionally, a novel extension to the QGS algorithm is proposed to enable search for dynamically changing multi-patterns of unordered data. Finally, a novel quantum application is presented that combines QHT and dynamic multi-pattern QGS to perform pattern recognition using dimension reduction on high-resolution spatio-spectral data. For higher emulation performance and scalability of the framework, hardware design techniques and hardware architectural optimizations are investigated and proposed. The emulation architectures are designed and implemented on a high-performance reconfigurable computer (HPRC). For reference and comparison, implementations of the proposed quantum circuits are also performed on a state-of-the-art quantum computer. Experimental results show that the proposed hardware architectures enable emulation of quantum algorithms with higher scalability, higher accuracy, and higher throughput, compared to existing hardware-based emulators. As a case study, quantum image processing using multi-spectral images is considered for the experimental evaluations. The analysis and results of this work demonstrate that quantum computers and methodologies based on quantum algorithms will be highly useful in realistic data-intensive domains such as remote-sensing hyperspectral imagery and high-energy physics (HEP).

Acknowledgments

I would like to acknowledge the support and inspiration of my dissertation advisor Dr. Esam El-Araby for always encouraging me to produce the best possible work. I am grateful for his continuous motivation and guidance in this effort. I am honored to have Dr. Prasad Kulkarni, Dr. Perry Alexander, Dr. Heechul Yun, and Dr. Tyrone Duncan as my dissertation committee members, and I am grateful to them for their participation. I am also thankful to Dr. Prasad Kulkarni for being an inspiring teacher and mentor during my Ph.D.

I would also like to acknowledge the support of my colleagues and team members of the KU Advanced-Reconfigurable and Quantum computing (KUARQ) research group at the University of Kansas. I am also thankful to my friends and community members in Lawrence, Kansas, for all their good wishes. Finally, I am grateful to my parents for their belief in my abilities and to my loving wife for her care and support. I would like to dedicate this dissertation to my family, without whom my work and achievements would have no meaning.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Quantum computing is one of the promising technologies of today, but it is still in its nascent development stage. The initial idea of a quantum computer was put forward by Benioff in 1980 [1], who theorized a quantum mechanical model for computing represented by Turing machines. This was followed by significant contributions from Feynman who proposed simulating quantum physics using a universal computing machine [2], and Deutsch who extended theories on quantum computers and linked quantum physics with computing [3]. Later during the 90s, the introduction of quantum algorithms for integer factoring and discrete logarithms by Shor [4], and a quantum algorithm for database search by Grover [5] generated immense interest and triggered research and development efforts towards quantum computers.

1.1 Prospect of Quantum Computing

At present, quantum technology is developing rapidly and promises an exciting future for computing. The current state-of-the-art quantum computers are capable of processing hundreds of quantum bits (qubits) and are termed as Noisy Intermediate-Scale Quantum (NISQ) devices. Research is being conducted heavily to mitigate the noise in these systems, in order to achieve fully fault-tolerant computation. It is estimated [6] that a quantum computer should be able to process thousands of qubits, including error-correcting qubits, in order to achieve or exceed the level of performance of existing classical systems. The event that quantum computers can outperform classical machines has been termed as ‘quantum supremacy’ [7]. A research team led by Google [8] has claimed experimental demonstration of quantum supremacy using their 53-qubit Sycamore quantum processor [8]. They showed that Sycamore takes roughly 200 seconds to sample a quantum circuit a million times, while the state-of-the-art classical supercomputer would take thousands of years to complete an equivalent task. This demonstration greatly improves the

prospects for quantum computing. In particular, the ability of a quantum computer to solve NP-hard problems [4] [5] [3] [9] which are classically intractable, is of great significance and interest. For example, a quantum computer using Shor's algorithm [4] could potentially solve large integer factorization in polynomial time. Thus, existing security schemes such as Rivest Shamir-Adleman (RSA), which are widely used in state-of-the-art cryptosystems, would be severely compromised since these security schemes assume that factoring of large integers is intractable in polynomial time. Another quantum algorithm of great interest is Grover's search algorithm [5]. Grover's search can be used to find a specific item in an unordered list of N items in $O(\sqrt{N})$ time, achieving quadratic speedup over the best classical search algorithms. There are also potential applications of quantum computers in simulation of quantum systems in chemistry [10] and quantum mechanics [2]. Another feasible application of quantum computers is in the field of image processing. Images encoded in the quantum domain can be processed using quantum algorithms such as Quantum Wavelet Transform (QWT) and Quantum Fourier Transform (QFT) [11] with greater time and/or space efficiency compared to classical methods. The prospect of quantum computing is well recognized by big technology companies such as IBM, Google, Intel, and Microsoft [12], as well as new startups such as IonQ and Rigetti [12], and each is investing heavily in research and development of quantum computing hardware and software.

1.2 Challenges and Motivation

Despite many companies having operational quantum hardware, the implementation of realistic quantum algorithms and their equivalent circuits on quantum computing architectures is extremely challenging. Quantum computers are highly sensitive to external environmental noise and quantum hardware must be isolated and maintained in cryogenic temperatures. The process by which the environment affects the state of a quantum computer is called quantum state

decoherence [11][13]. Interactions with the environment cause information to be lost and the quantum state to collapse, i.e., lose its quantum mechanical properties. Decoherence is a fundamental constraint for practical implementation of quantum circuits on quantum computers [13]. A quantum circuit is generally modeled two-dimensionally, with the y-dimension (width) representing qubits, and the x-dimension (depth) representing the levels of quantum transformations or the circuit time-steps. The depth determines the execution time of the quantum circuit, i.e., higher the depth, higher the circuit execution time. A quantum circuit must complete execution within a constrained time frame before decoherence causes the state to collapse. Therefore the depth of the circuit that can be implemented on a quantum computer is also limited. For any quantum computing system, the decoherence time constraints are termed as T1 and T2 times [14] [15] and are determined by the quality of the underlying quantum technology. T1 is the time taken for natural relaxation of the qubit to its ground state, while T2 is the time taken for the qubit to get affected by environmental noise [15]. To mitigate the decoherence problem, methods need to be investigated at the quantum device level to achieve higher T1 and T2 times. It is also necessary to optimize quantum circuits to reduce depth, such that circuit execution times are less than the system decoherence times.

Another critical challenge that arises because of decoherence time constraints is encoding classical data onto the quantum computer, or classical-to-quantum (C2Q) data encoding [16]. A quantum algorithm is a sequence of transformations on an initial quantum state, resulting in an output quantum state. C2Q is the process of encoding classical data required by the algorithm onto the initial quantum state. A state-preparation circuit is required to perform C2Q data encoding, in addition to the circuit performing the quantum algorithm operations. For I/O intensive applications, C2Q data encoding is problematic as the state-preparation circuit execution often

exceeds the decoherence time constraints of the system. The data encoding time becomes large compared to the algorithm compute time, thus nullifying any computational benefits of the algorithm alone, and having an adverse effect on overall system performance. Measuring/observing the output of a quantum circuit and extracting useful classical data from the output quantum state, or quantum-to-classical (Q2C) data decoding, is also another challenge [16]. Measurement/observation of any quantum state destroys the properties of that state, thereby data encoded in that state is lost. To recover useful data about the output quantum state, the quantum circuit is 'sampled' repeatedly, i.e., the circuit is executed multiple times and the output is measured each time. Performing multiple circuit executions deteriorates the overall system time complexity. The C2Q and Q2C processes are integral when benchmarking the performance of any quantum system and/or algorithm. Improving only the computation component of any quantum algorithm will not be sufficient if C2Q and Q2C components remain performance bottlenecks. Therefore, it is vital to investigate time-efficient and decoherence-optimized quantum circuits for C2Q and Q2C processes, along with improving and optimizing circuits for quantum algorithms.

The current state-of-the-art quantum computers are of intermediate scale, i.e., they have low number of qubits relative to the actual number of qubits required to encode realistic problems. Scaling up quantum systems is extremely challenging as it is difficult to maintain full physical connectivity between the qubits. Moreover, the required addition of error-correcting qubits for reducing errors make quantum systems less scalable. Building a quantum hardware and software system is expensive too [17]. Consequently, it is very costly for users to gain access to these systems. Researchers and students can have either limited access to these systems, or are limited to only the small-scale systems with low qubit counts. The low scalability and noisy nature of current NISQ-era quantum devices, as well as cost of access hinders research and slows the growth

of quantum computing knowledge. Efforts into simulation and emulation of quantum computers have emerged consequently, to help researchers validate existing algorithms as well as evaluate newer quantum algorithms. There exists a plethora of quantum simulators which require costly, resource-intensive, and power-hungry supercomputing platforms to run quantum algorithms. Therefore, there is a need for more cost-effective, resource-efficient, and power-efficient simulators. Another class of simulators for quantum algorithms being developed are hardware-based emulators. Hardware-based emulation methods can take advantage of hardware parallelism and acceleration to produce results at a higher throughput. Most hardware-based emulators are based on reconfigurable hardware such as Field-Programmable Gate Arrays (FPGAs) and therefore are more cost-effective and power-efficient. However, current FPGA-based emulators have low scalability, low accuracy, and low throughput. They can emulate only a small number of qubits, use low-precision, and have low operating frequencies. Further investigation is needed for performing scalable, high-precision, and high-throughput hardware-based emulation of quantum algorithms.

1.3 Problem Statement

We identify that implementation of deep quantum circuits is a critical problem for current class of quantum computers. Quantum circuit execution constraints put in place due to decoherence make it difficult to efficiently perform classical-to-quantum (C2Q) data encoding, and quantum-to-classical (Q2C) decoding in quantum systems. For I/O-intensive applications such as image processing, it becomes impossible to transfer large amounts of data to/from the quantum computer. As a result, investigation of I/O-intensive real-life applications on quantum computers is hindered. In addition, current quantum computing systems have problems such as low qubit counts, noisy low-fidelity outputs, and high cost of access. In this regard, there is a critical need for cost-

effective, power-efficient simulation/emulation platforms for verification and benchmarking quantum algorithms. Existing simulators are generally based on costly supercomputing platforms that consume a lot of resources and power. Alternatively, hardware-accelerated emulation is more cost-effective, but existing hardware-based emulators face challenges such as low scalability, low accuracy, and low throughput.

1.4 Research Goals and Approaches

1.4.1 Overview

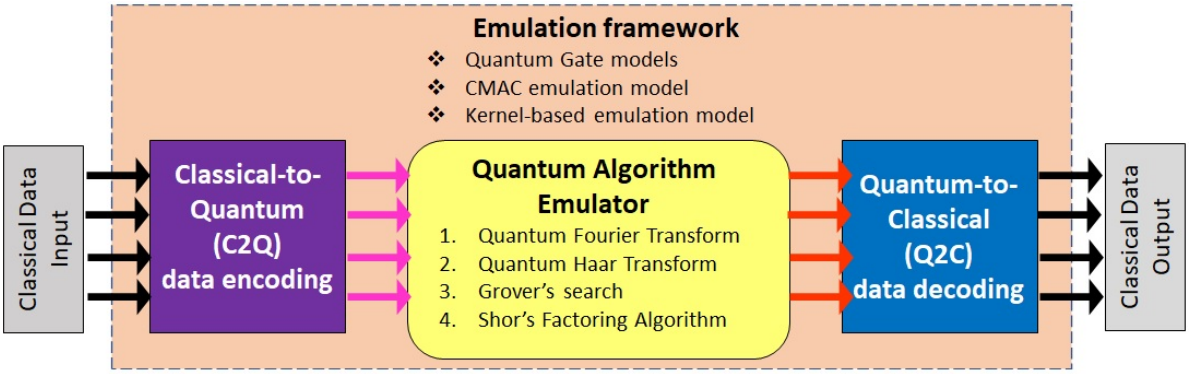


Fig. 1: Overview of the proposed emulation system.

According to DiVincenzo's criteria [18], a quantum computing system has the following features: (a) a well characterized unit of storing information, i.e., a quantum 'bit' or qubit, (b) ability to initialize/prepare the state of the qubits, (c) long decoherence times, (d) a universal set of quantum gates, and (e) the ability to measure the final state of the quantum bits. The operation of a quantum computing system has the following cycle: *prepare input state*, *apply transformations*, and *measure output state* [16]. In this work, the primary research goals are to perform realistic and complete emulation of such a quantum computing system [18], and to evaluate quantum algorithms and quantum applications by emulation. The proposed approach for emulation consists of three components, a C2Q data encoding model for state preparation and initialization, a quantum algorithm emulator model for emulating quantum gates and algorithmic

computations, and a Q2C data decoding model for measuring quantum output state, as shown in Fig. 1. The C2Q model emulates the process of state synthesis, i.e. encoding classical data onto a state in the quantum domain. Our goal is to improve the C2Q process by developing faster data encoding methods. We will investigate and propose space and time-efficient methods and efficient quantum circuits for state-preparation and C2Q data encoding. Corresponding emulation architectures for evaluating C2Q will also be presented. For emulating algorithms, a hardware-based quantum algorithm emulator will be developed. We will propose and investigate different emulation models for the quantum algorithm emulator, for investigating a variety of quantum algorithms. We will discuss advantages and disadvantages of each model and the type of quantum algorithm they are most suitable for. Different hardware design techniques and trade-offs will be investigated for improving the performance and efficiency of the models and the emulator. A variety of quantum algorithms will be investigated, including Quantum Fourier Transform (QFT), Quantum Wavelet (Haar) Transform (QHT), and Quantum Grover's Search (QGS). The Q2C model, see Fig. 1, measures the output state of the algorithm, and extracts useful classical information about the output quantum state. For the Q2C model two methods will be investigated. In these methods, QFT and optimized QHT circuits will be utilized respectively to improve the time complexity of the Q2C process.

1.4.2 Classical-to-Quantum Data Encoding

Initially, a survey and analysis of the existing methods for C2Q data encoding will be performed. Our goal is to develop decoherence-optimized methods and circuits for C2Q and verify their functionality and feasibility by emulation. A quantitative comparison of our proposed methods with existing methods will also be provided. The process of synthesizing a quantum state is called arbitrary state synthesis in the literature [19], and the circuit required is called state-

preparation and/or state-initialization circuit. Our aim is to investigate and develop state-preparation circuits that have low spatial and temporal complexities, i.e., low gate count and low circuit depth. For circuit synthesis generally recursive circuit methods have been proposed that assume unity global scale and phase of qubits, and which result in large gate counts and circuit depth. In our approach we use, instead, quantum multiplexor circuits that result in lower spatial and temporal complexities. We then apply efficient decompositions to the multiplexor operations and analyze the final circuit depth and gate count. We propose two methods for C2Q: Method 1, which includes the global scale and phase of qubits, and Method 2, in which the global scale is unity. Analysis of the corresponding circuit depths for both proposed methods will be performed and compared to existing methods.

1.4.3 Emulation of Quantum Algorithms

At present there is a lot of work being done on large-scale simulation of quantum computers [6] [20] [21] [22] [23] [24]. Quantum computing simulators are generally run on costly and resource-intensive hardware platforms. On the other hand, FPGA-based hardware emulators have shown that quantum circuits can be emulated at lower costs [25] [26] [27] [28] [29] [30], but lack scalability, and have low accuracy and throughput. In this work, we aim to develop FPGA-based emulation methods for quantum computing that are highly scalable, maintain the inherent parallelism of quantum algorithms, maintain a high-level of accuracy and high-level of throughput. The goal of this effort is to develop a methodology/framework that is flexible for investigating a variety of quantum algorithms such as QFT, QHT, and QGS. The emulation framework will be utilized to extend algorithms with newer capabilities, optimize algorithms, and combine algorithms to develop new applications. For example, we will present the hardware architecture to dynamically generate the transformation matrix of QFT during emulation. Quantum circuit

generalization and optimizations will be investigated for QHT for multi-dimensional data processing. The circuits will also be extended for multi-level decomposable, multi-dimensional QHT operations and optimizations will be applied to reduce circuit depth. Using the emulation framework, we will also extend the conventional QGS circuit from static, single-pattern searching to dynamic, multiple-pattern data search. Finally, a methodology for dimension reduction of spatio-spectral data using multi-level decomposable, multi-dimensional QHT and pattern matching using dynamic, multi-pattern QGS will be investigated and evaluated using the emulation framework. The feasibility and usability of this methodology will be demonstrated experimentally by a quantum image processing application. The proposed emulation framework will use 32-bit floating point precision for higher accuracy, and a fully pipelined hardware architecture for highest throughput. Emulation techniques based on *complex multiply-and-accumulation* and *kernel* operations will be analyzed, and different methods of computation such as *lookup*, *dynamic generation*, and *streaming* will also be investigated. Architectural optimizations and area / speed trade-offs for these methods will also be explored for improving the space and/or space-time complexities of the emulation.

1.4.4 Quantum-to-Classical Data Decoding

Measuring or observing the output state of a quantum circuit results in a non-deterministic outcome [16]. To decode meaningful classical data from the output of a quantum circuit, the general approach involves sampling the quantum circuit multiple times, and counting the frequencies of the different outcomes. The outcome frequencies are then used to construct a probability distribution with probability set $\{P_i\}$, in which the set $\{\sqrt[2]{P_i}\}$ represent the output of the quantum circuit. In this approach, there is significant overhead due to the repeated sampling of the circuit. Another approach for Q2C data decoding, based on using the QFT, was proposed in [16].

However, the QFT-based approach is specific to image processing applications, and no experimental evaluation was provided. In this work, we propose and evaluate a novel Q2C data decoding method, based on using the QHT algorithm. The QHT algorithm can be effectively used to reduce dimensionality of data while retaining both spatial and temporal locality, and thus reducing the number of qubits required to represent the data. Therefore, the proposed QHT-based approach will be effective in reducing the sampling overhead of the Q2C data decoding process. In this work, we will demonstrate multi-level decomposable, multi-dimensional QHT circuits for Q2C data decoding. Specifically, we will investigate the packet and pyramidal forms of decomposition for two-dimensional (2D) and three-dimensional (3D) QHT. We will use simulation on a quantum device to experimentally evaluate both the QFT and QHT based approaches.

Chapter 2: Background and Related Work

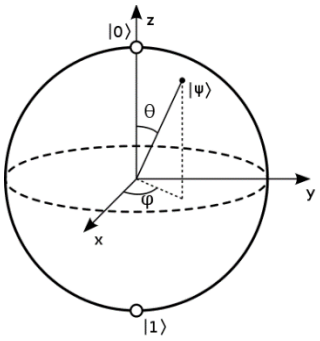
2.1 Quantum Computing

The general and commonly adapted model of quantum computation is the gate-model [11]. In this model, computation begins with the system set in an initial quantum state denoted as $|\psi_{in}\rangle$ in the bra-ket or Dirac notation [31]. Depending on the quantum algorithm, a sequence of unitary transformations comprised of gates, i.e., $(U_0, U_1, ..., U_M)$, are applied to the input quantum state to reach a final output quantum state $|\psi_{out}\rangle$, see (1).

$$\left|\psi_{out}\right\rangle = U_m \cdot U_{m-1} \cdot \ ... \ \cdot U_2 \cdot U_1 \cdot \left|\psi_{in}\right\rangle \tag{1}$$

2.1.1 Qubits, Superposition, and Entanglement

The quantum bit or *qubit* is the smallest unit of quantum information [11]. A single qubit can exist in *superposition* of two basis states, $|0\rangle$ and $|1\rangle$, which can be represented by a Bloch sphere [11] as shown in Fig. 2. The north pole of the Bloch sphere represents the basis state $|0\rangle$ while the south pole represents the basis state $|1\rangle$. Any other point on the surface of the sphere is a valid pure state or *superimposed* state of the two basis states, denoted as $|\psi\rangle$. The overall state of the qubit is satisfied by the linear superposition equation, see (2), where α and β are complex coefficients, also termed as *amplitudes*, whose values depend on the azimuth and elevation angles φ and θ, respectively, as shown in Fig. 2. Algebraically, the qubit can be represented by a column vector of the complex coefficients, see (2). When a qubit is measured, the superposition is lost, and the qubit will collapse to a basis state. According to the Born rule [11], the magnitudes of the complex coefficients/amplitudes, i.e., $|\alpha|^2$ and $|\beta|^2$ represent the probabilities of measuring the qubit in corresponding $|0\rangle$ and $|1\rangle$ basis states, respectively.



$$|\psi\rangle = cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$
$$= \cos\left(\frac{\theta}{2}\right)|0\rangle + (cos\varphi + isin\varphi)\sin\left(\frac{\theta}{2}\right)|1\rangle$$
$$= \alpha|0\rangle + \beta|1\rangle$$
$$where, 0 \le \theta \le \pi \; and \; 0 \le \varphi \le 2\pi$$

Fig. 2: Bloch sphere representation of a qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \equiv \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \tag{2}$$

Multiple qubits can form a quantum state. The state space represented by n qubits is determined by the Kronecker product, denoted by ⊗, of the individual qubit vector spaces, see (3). The n-qubit quantum state can also be described as a superposition of $2^n = N$ basis states, generally known as the computational basis [16], see (3), where $C_0, C_1, \dots, C_{N-1}$ are the complex coefficients/amplitudes of the basis states. Algebraically, the n-qubit state can be represented by a column vector, termed as a *state vector*, comprising of the N coefficients, see (3). An example of a 3-qubit quantum state is shown in (4), where $C_0 ... C_7$ are complex coefficients of the computational basis states ranging from $|000\rangle$ to $|111\rangle$.

$$|\psi\rangle^{\otimes n} = |q_{n-1}\rangle\otimes|q_{n-1}\rangle\otimes..\otimes|q_1\rangle\otimes|q_0\rangle = \sum_{i=0}^{N-1} C_i\,|i\rangle = \begin{bmatrix} C_0 \\ C_1 \\ \vdots \\ C_{N-1}\end{bmatrix} \tag{3}$$

$$|\psi\rangle^{\otimes 3} = C_0|000\rangle + C_1|001\rangle + \cdots + C_7|111\rangle \tag{4}$$

Entanglement is another distinguishing property of qubits [11]. Two or more qubits may become entangled meaning that each entangled qubit becomes strongly correlated to the other and the quantum state cannot be factored into a Kronecker product of the individual qubits, i.e.,

$|\psi\rangle = |q_{n-1}q_{n-2}...q_1q_0\rangle \neq |q_{n-1}\rangle \otimes |q_{n-2}\rangle \otimes ... |q_1\rangle \otimes |q_0\rangle$. The benefit of quantum entanglement is that operations on one entangled qubit can affect other entangled qubits. Likewise, measuring an entangled qubit can give information about the state of other entangled qubits [16].

2.1.2 Quantum Gates

In gate-model quantum computing, quantum gates are the set of unitary transformations on qubits and are analogous to classical logic gates [16]. Quantum gates are used to manipulate the states of qubits and are represented by $N \times N$ unitary matrices where $N = 2^n$ and *n* is the number of qubits. In other words, a one-qubit gate is represented by a 2×2 unitary matrix, a two-qubit gate is represented by a unitary 4×4 unitary matrix, and so forth. Commonly used quantum gates like the Hadamard (*H*), SWAP, controlled NOT (CNOT), controlled phase shift gate (R_k), controlled Pauli gates (*cX, cY, cZ*), rotation gates (R_x, R_y, R_z) and multi-controlled gates are discussed in the next sections. The quantum gate symbols, and corresponding matrix representations are shown in Fig. 3.

The Hadamard or *H* gate is an important single-qubit gate that creates a superposition of the basis states with equal coefficients [11]. An *H* gate applied on the ground basis state $|0\rangle$ puts the qubit to a resulting state with equal probability superposition between the $|0\rangle$ and $|1\rangle$ states, i.e., $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. A set of Pauli X, Y, and Z gates [16] exist that equate to rotations around the *x*, *y*, and *z* axes of the Bloch sphere respectively. The Pauli X gate symbol and matrix are shown in Fig. 3. The SWAP gate is a two-qubit gate that simply exchanges the bit values provided as input [11].

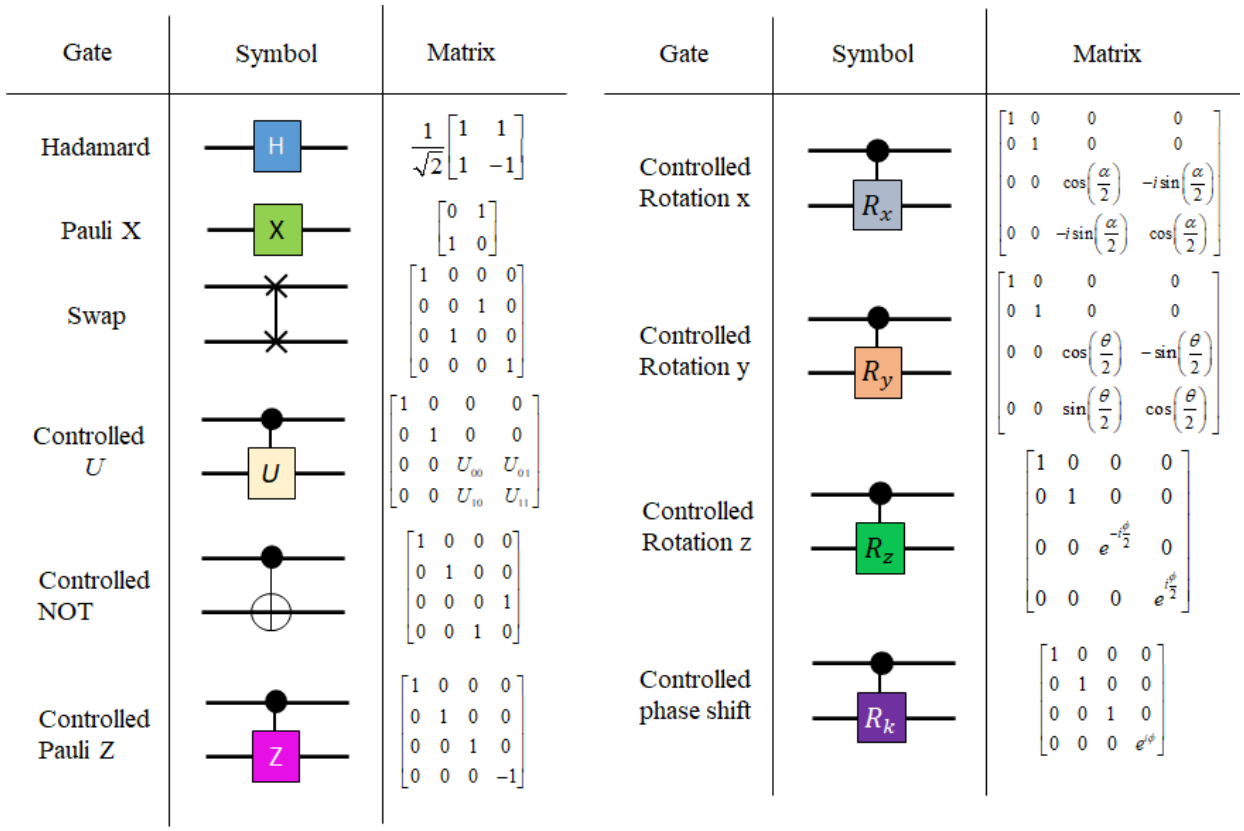


Fig. 3: Quantum gates, gate symbols, and matrix representations.

Any 1-qubit gate with matrix operation U can be extended to form a controlled U gate, see Fig. 3, where U_{00}, U_{01}, U_{10}, U_{11} are the elements of the 2x2 matrix representation of U. A multiple-controlled gate can be formed by adding multiple number of qubits controlling the operation of a target qubit. The NOT gate is a single qubit gate that inverts the state of the qubit. A control qubit can be added to a NOT gate to create a two-qubit gate referred to as controlled-NOT or CNOT. When the control qubit is $|1\rangle$ then a NOT inversion will be applied to the other qubit, otherwise the other qubit remains unchanged. The CNOT gate and its corresponding matrix is shown in Fig. 3. Control qubits can be added to the Pauli gates to create controlled Pauli gates cX, cY, and cZ, which are used in many quantum algorithms. The operation of cX represents a rotation of π around the x-axis and is synonymous with the CNOT gate. The cY gate represents a rotation of π about

the *y*-axis of the Bloch sphere. The *cZ* gate is a rotation of π about the *z*-axis, and its gate and matrix are shown in Fig. 3.

There exists a specific class of rotation gates for performing arbitrary rotations about the *x*, *y*, and *z* axes of the Bloch sphere. These gates are important as they can be used to rotate a qubit into any arbitrary state from the ground state and vice-versa. Thus, any arbitrary 1-qubit gate can be decomposed into a sequence of rotation gates [16]. Additional control qubits can be added to the rotation gates to form controlled rotation gates (R_x, R_y, R_z) that have important usage in arbitrary state synthesis. The controlled phase shift gate, R_k [11] is a similar 2-qubit gate that applies a phase shift $e^{i\phi}$ based on the control qubit, where $\phi = \frac{2\pi}{2^k}$ is the phase shift with period 2π. The symbols and matrices for these rotation gates are shown in Fig. 3.

2.2 Quantum Algorithms

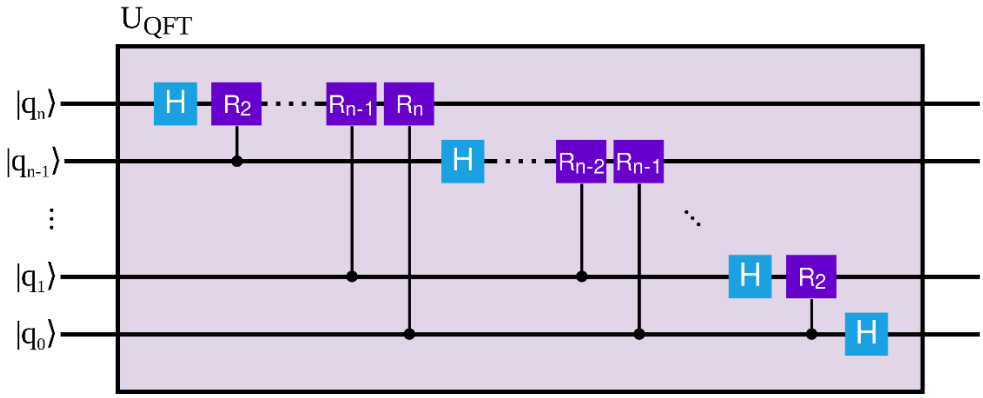
2.2.1 Quantum Fourier Transform

Quantum Fourier Transform (QFT) is an integral part of many larger quantum algorithms such as Shor's algorithm for integer factoring [11]. QFT is the quantum equivalent of the classical Discrete Fourier Transform (DFT). The input data samples for QFT are encoded as the basis state coefficients/amplitudes of a superimposed quantum state. When performed on a quantum computer, QFT can achieve exponential speedup over its classical counterpart. The mathematical model and quantum circuit for QFT can be determined from the classical DFT as demonstrated in [32]. The QFT algorithm transforms an arbitrary superposition of computational basis states to a corresponding superposition of Fourier basis states and is represented by (6a), where $|\psi\rangle$ is the quantum input state vector, and *n* is the number of qubits. The input signal samples are encoded as a normalized amplitude sequence given by (6b). A generalized *n*-qubit QHT circuit composed

of H gates, R_k and SWAP gates, is shown in Fig. 4. The QFT transformation can be represented using a single unitary matrix, U_{QFT}, of size $N \times N$, as shown in Fig. 4, where $\omega_n = e^{\frac{2\pi i}{n}}$.

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{q=0}^{2^n-1} f(q\Delta t)|q\rangle \xrightarrow{QFT} \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \sum_{q=0}^{2^n-1} f(q\Delta t) e^{2\pi i (\frac{qk}{2^n})} |k\rangle \tag{6a}$$

$$\sum_{q=0}^{2^n-1} |f(q\Delta t)|^2 = 1 \tag{6b}$$



$$U_{QFT} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{N-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(N-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \dots & \omega_n^{3(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{N-1} & \omega_n^{2(N-1)} & \dots & \omega_n^{(N-1)(N-1)} \end{bmatrix}$$

Fig. 4: Quantum circuit and corresponding transformation matrix for n-qubit QFT.

2.2.2 Quantum Wavelet Transform

The wavelet-transform (WT) decomposes signals/data into its spatio-temporal spectral components [33]. Unlike Fourier-transform, WT uses a set of non-sinusoidal functions, called mother-wavelets, which are localized spatially and temporally [33], resulting in the preservation of data spatial-locality. The computational speed of WT is higher than other transforms [33]

making it highly effective and commonly implemented in image processing applications. The first and simplest WT was introduced by mathematician Alfred Haar [34] and is thus named the Haar wavelet transform. The Haar mother wavelet function can be constructed using a unit step function, as shown in (7a). The discretized version of the Haar wavelet function is defined in (7b).

$$\Psi(t)_{Haar} = u(t) - 2u\left(t - \frac{1}{2}\right) + u(t-1) \tag{7a}$$

$$\Psi_D^* = \left(\frac{i}{N}\right) = \begin{cases} +1, & 0 \le i \le \frac{N}{2} \\ -1, & \frac{N}{2} \le i \le N \\ 0, & \text{otherwise} \end{cases} \tag{7b}$$

The discrete wavelet transform can be implemented as quantum Wavelet transform (QWT) [35] in the quantum domain. The general QWT can be expressed [36] by:

$$|\psi\rangle = \sum_{q=0}^{N-1} f\left(q.\Delta t\right)|q\rangle, \text{ where} \sum_{q=0}^{N-1} |\, f\left(q.\Delta t\right)|^2 = 1 \tag{7c}$$

$$\begin{aligned} |\psi\rangle_{QWT} &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{q=0}^{N-1} f\left(q.\Delta t\right) \Psi\left(\frac{q-j}{K}\right) |\, j\rangle \\ |\psi\rangle_{QWT} &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{q=0}^{N-1} f\left(q.\Delta t\right) \Psi\left(\frac{q-j}{K}\right) |\, j\rangle \end{aligned} \tag{7d}$$

where Ψ is the mother wavelet function in complex conjugate form, Δt is the sampling period, K is the wavelet-window-size in samples, $N = 2^n$ is the number of data samples represented as the total number of quantum states, *n* is the number of qubits, $|\psi\rangle$ is the input state, and $|\psi\rangle_{QWT}$ is the output state. The expression for quantum Haar transform (QHT) can thus be derived using the Haar wavelet function Ψ_{Haar}, see (7d).

The Haar wavelet function can be generalized by quantum operations using n qubits, and a d-dimension kernel. The Haar unitary transformation, U_{QHT}, using d entangled H gates and $n - d$

entangled I gates is shown in (7e), where H is the Hadamard gate and I is the identity matrix. For two-dimensional QHT (2D-QHT) with $d = 2$, the transformation matrix can be derived as shown in (7f).

$$U_{QHT} = I^{\otimes(n-d)} \otimes H^{\otimes d}$$

$$\text{where, } H^{\otimes d} = \underbrace{H \otimes H \otimes \ldots \otimes H}_{d}, \quad I^{\otimes(n-d)} = \underbrace{I \otimes I \otimes \ldots \otimes I}_{n-d}, \tag{7e}$$

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \; I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$U_{QHT}^{2D} = I^{\otimes(n-2)} \otimes H^{\otimes 2}$$

$$\text{where, } H^{\otimes 2} = H \otimes H = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{7f}$$

2.2.3 Quantum Grover's search

Grover's search is a quantum algorithm that can be used to search over an unsorted list of N elements [16]. The objective of this search algorithm is to find an element $s*$ that satisfies $f(s^*) = 1$ and holds (8) true, where s^* belongs to the set $S = \{s_1, s_2, s_3, ..., s_N\}$, N is the cardinality of S, and f is a Boolean function such that $f(x) \to \{0,1\}$.

$$f(x) = \begin{cases} 1, \text{ if } x = s* \\ 0, \text{ if } x \neq s* \end{cases} \tag{8}$$

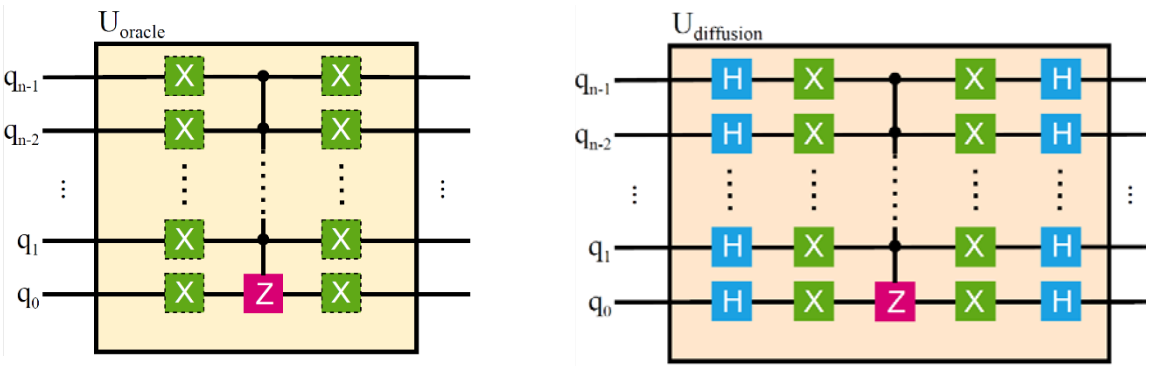
A quantum computer running Grover's algorithm can perform the search in $\sqrt{N}$ queries compared to the best classical search algorithm, resulting in a quadratic speedup [5]. Grover's algorithm can also be used to find multiple items/patterns from a list. A pattern is defined here as a string of bits. To find multiple patterns, the total number of solutions must be known ahead of running the algorithm [16]. When searching for multiple patterns, Grover's algorithm will find any

of the target patterns with equal probability [16]. The inputs to Grover's algorithm are the patterns encoded as the basis states of a superimposed quantum state. Initially, the input state is in equal superposition, i.e., their coefficients/amplitudes are equal, and therefore the probabilities of locating any item in the list are also equal. To obtain this input state an *H* gate is applied to the ground or zero state of each qubit. Two operations are then performed for an optimal number of iterations on this initial state, namely, *oracle* (also called *phase inversion* and *diffusion* (also called *inversion about the mean*) [16].

The *oracle* step or *phase inversion* operation [16], takes the input quantum state and inverts the coefficients/amplitudes of the basis states representing the patterns for which we are searching [16]. To see how this function works, let our *oracle* be denoted as U_{oracle}, as shown in (9). If $x \neq s*$, then $f(x) = 0$ and $|x\rangle$ will have no change. Otherwise, $|x\rangle$ will be multiplied by -1, resulting in a phase inversion for $|x\rangle$.

$$U_{oracle}|x\rangle = (-1)^{f(x)}|x\rangle$$
$$U_{diffusion} = I - 2|x\rangle\langle x| \tag{9}$$

The general quantum circuit of the *oracle* is shown in Fig. 5(a). The *X* gates in Fig. 5(a) have a dashed border indicating that the gate may or may not be needed for its corresponding qubit, depending on the target pattern. An *X* gate should be placed if the basis state for the target qubit is a $|0\rangle$. For example, if the sought pattern is $|0.....0\rangle$ then an *X* gate should be placed on every qubit. If $|1.....1\rangle$ is the target pattern, then there should be no corresponding *X* gates. Therefore, the conventional oracle circuit must be modified every time the target input pattern changes.



(a) Circuit for oracle operation (b) Circuit for diffusion operation

Fig. 5: Quantum circuits for Grover's search algorithm.

In the *diffusion* step, also called *inversion about the mean* operation or *amplitude amplification* operation, the inverted coefficients/amplitudes will be amplified, and the other coefficients/amplitudes will be attenuated [16]. This operation, denoted as $U_{diffusion}$ in (9), is done by finding the mean value of all the amplitudes and inverting each amplitude about the mean. This causes the positive amplitudes, which are close to but greater than the mean, to be attenuated while the negated amplitudes, which are less than the mean, to be amplified [16]. The general quantum circuit of the diffusion operation is shown in Fig. 5(b).

Repeating the two steps, i.e., *oracle*, and *diffusion*, consecutively will increase the target amplitudes of the target patterns (basis states), thus increasing the probability that the quantum state collapses to the target basis state(s). Boyer et al. [37] derived the optimal number of iterations, m, required for maximizing the probability of successful search, see (10), where N is the size of the unsorted list of elements, $N_{patterns}$ equals the number of solutions/patterns being searched for such that $N_{patterns} \leq N$, and $k = 1, 3, 5, 7, I$ is an odd number. For single-pattern Grover's algorithm, the target pattern's amplitude will be close to 1 at the end of m iterations. For multi-pattern Grover's algorithm, each of the target patterns (basis states) will have equal but higher amplitudes compared to the remaining basis state coefficients at the end of m iterations.

$$m = \left\lfloor \frac{\pi \cdot k}{4 \sin^{-1}\left(\sqrt{\frac{N_{patterns}}{N}}\right)} \right\rfloor \quad (10)$$

2.3 CPU/GPU-based Software Simulators

The work in [22] demonstrated a massively parallel quantum simulator implemented on different supercomputing platforms. Simulation of up to 48 qubits was performed, however, high number of resources (1 petabyte memory) were consumed, and no software optimizations were reported. A GPU-accelerated simulation of quantum annealing, and the Quantum Approximation Optimization Algorithm (QAOA) was presented in [24], showing simulation of up to 40 qubits. For this simulation, a large-scale supercomputer with around 4K Tensor Core GPUs distributed over 936 nodes was utilized. The authors in [20] proposed a GPU-based simulator, showing implementation of up to 25 qubits, while a minimal quantum circuit was used in the simulation case study. The work in [38] also used GPUs and achieved simulation of entangled Hadamard gates up to 21 qubits. In [23], the authors demonstrated simulation of up to 38 qubits using a GPU accelerated platform. However, cost-prohibitive amounts of computing resources (2048 nodes and 24 cores/node) were dedicated in the simulation. One of the recent works on quantum simulation [21] used a cluster supercomputing platform supported by the Alibaba group. In that work, the authors demonstrated simulation of up to 144 qubits with circuit depth of 27 gate levels using 131,072 processors and 1 petabyte memory. However, they have not investigated any quantum algorithms and the circuits consist of random gates. Furthermore, their simulator was shown to evaluate only one out of all possible output states. The existing CPU/GPU-based quantum simulators are costly since they consume large amounts of resources in terms of required number

of processors and system memory. The proposed FPGA-based emulation framework in this work is much less resource-intensive and is therefore highly cost-effective compared to existing CPU/GPU-based simulators.

2.4 FPGA-based Hardware Emulators

An assortment of work has also been done on hardware-based emulation of quantum circuits using FPGAs. In [25] the authors presented a quantum processor that abstracted quantum circuit operations into binary logic. The proposed system was shown to emulate up to 75 qubits. However, the modeling methodology of the quantum operations was highly inaccurate due to the use of low precision (1 bit) for the representation of state coefficients. Moreover, hardware cost in terms of resource utilization was not reported. In [27] the authors implement an emulator based on a library of quantum gates. The gate operations were implemented using fixed-point arithmetic, and a low operating frequency of 82.4 MHz was reported for the emulation of 3-qubit QFT and Grover's search algorithm. In [39] the authors proposed a similar fixed-point emulator, reporting up to 3-qubit QFT, but details regarding both their approach and the mapping of the quantum algorithm to the proposed architecture are missing. Moreover, quantum entanglement was also missing in their model. In [28] and [29] the authors present hardware architectures emulating QFT and Grover's search circuits. In their work, a maximum fixed-point precision of 24-bits was used to emulate up to 5-qubit QFT and 7-qubit Grover's search on a single FPGA. Scalability of their design is limited and there is no proposed solution to the problem of scalability. In [30] the authors propose a high-level synthesis (HLS) based emulation framework for QFT, but here also, the scalability of their design is limited and the authors did not address that limitation. In a related work, ProjectQ [26] compared simulation and emulation results trying to showcase the superiority of quantum computer emulators in terms of performance.

While modular and hierarchical modeling approaches in previous works improved re-usability, the modeling of each quantum gate as an individual component consumes greater resources, reduces accuracy, and limits scalability. In this work, we propose and evaluate emulation models that significantly reduce the resource utilization and emulation times, thus improving scalability and allowing the use of floating-point precision for improved accuracy. In this work, we report the highest number of fully entangled qubits on a single FPGA among related work. Lastly, fully pipelined designs of the hardware architectures resulted in higher operating frequency and throughput compared to existing emulators. The proposed emulation framework is also the first among existing related work, to integrate C2Q and Q2C methods with emulation of quantum algorithms.

Chapter 3: Classical-to-Quantum Encoding

3.1 Related Work

Existing methods of encoding classical data to a quantum representation are of three types: (a) basis encoding, (b) angle encoding, and (c) amplitude encoding [40]. Basis encoding involves encoding the binary representations of the data points as basis states. Typically, this technique is costly in terms of number of qubits. The authors in [41] presented an optimized basis encoding technique for image processing, where pixels are represented by the tensor product of their color and position. As a result of their optimization, the qubit cost was lowered, however their technique incurred greater circuit depth. Angle encoding represents one data point per qubit, with each data point encoded as a normalized rotation in the Bloch sphere. The authors in [42] investigated angle encoding and presented a quantum method for image edge detection. However, their method might render being unrealistic because each image pixel requires one qubit for encoding. In amplitude encoding, each data point is represented as the amplitude/coefficient of a basis state in a superimposed quantum state. The work in [43] proposed circuits with depth complexity $O(n)$, where n is the number of qubits. However, the number of qubits required is of the order $O(N)$, where $N = 2^n$ is the data size, therefore the proposed circuits are not feasible for current or near-future quantum processors.

Among the data encoding methods, amplitude encoding can represent the largest number of data points with the least number of qubits, but the technique requires complex quantum circuits to implement. Over the years, a number of methodologies based on amplitude encoding have been proposed for quantum state-preparation, also known as arbitrary state synthesis [19], [44], [45], [46], [47]. The most efficient methods have a spatial complexity of $O(2^{n+2})$, where n is the number of qubits of the corresponding state-preparation circuit. In each work, the synthesis method

has been evaluated by counting the total number of quantum gates (gate count) in the synthesis circuit. However, there has been insufficient emphasis on quantum circuit depth [48] for state synthesis. The circuit depth is defined as the number of gates or time-steps in the longest path of a circuit. The circuit depth is closely related to the temporal complexity [48] and can be used to determine whether a quantum circuit can be run within the decoherence constraints of a particular quantum system.

Song and Williams in [44] presented methodologies for synthesizing any n-qubit pure state or mixed state. For synthesizing a pure state, their algorithm involves first applying Gram-Schmidt procedure on a matrix that contains the input data as the leftmost column, to produce a unitary matrix. The unitary matrix is then synthesized to a quantum circuit using a recursive algebraic method that has a complexity of $O(2^{2n})$ [16], where n is the number of qubits. The authors in [46] presented transformations for one arbitrary state $|a\rangle$ to another $|b\rangle$ using uniformly controlled rotations. From their presented circuit transformation from $|a\rangle$ to $|b\rangle$, it can be inferred that transformation from $|a\rangle$ to $|0\rangle$ (or to any basis state) would require half the reported gate count. No analysis of circuit depth was provided in their work. To compare with our proposed circuits, we considered their circuit transforming state $|a\rangle$ to $|0\rangle$ and calculated the corresponding gate count and circuit depth to be $2^{(n+2)} - 6$. The work in [19] presented a method based on disentangling a qubit, i.e., producing a basis state $|0\rangle$ or $|1\rangle$ on the lowest significant qubit. The authors state that this disentangling method, which requires $2^n - 2$ CNOT gates for an n-qubit circuit, can be used recursively to transform any state to a desired basis state. They reported that the resulting final transformation circuit uses $2^{n+1} - 2n$ CNOT gates, however, no detailed analysis was provided. Furthermore, only the CNOT gate count was provided, while their proposed circuit also requires single-qubit rotation gates that would double the total gate count to $2^{n+2} -$

$2n$. In [47] the proposed methodology is based on applying $n-1$ rotation steps with permutations on the amplitudes in-between each rotation where additional gates are required in the intermediate permutations. The total gate count reported is $2^{n+2}+4n-9$ with no circuit representation of their methodology. To be consistent with our analyses, we calculated their circuit depth to be $2^{n+2}+3n-8$.

In this work, we have defined the process of encoding classical data onto the quantum domain as classical-to-quantum (C2Q) data encoding. We propose two C2Q methods based on amplitude encoding and the corresponding state-preparation/state-synthesis circuits that results in a lower circuit complexities than existing methods. We present the analytic expression for circuit gate depths that were not considered in prior work. We also present the full and optimized quantum circuits corresponding to our methods, and experimentally evaluate our circuits using simulation, emulation on FPGA, and hardware implementation on a real quantum device from IBM Quantum (IBM Q) [14]. In addition, the state fidelity of the proposed circuits is reported for the simulations on the quantum device.

3.2 Proposed Methods

We propose two methods and corresponding quantum circuits for C2Q data encoding. Given a classical dataset of $N=2^n$ elements, where *n* is the number of required qubits to represent the classical dataset, we propose a quantum circuit denoted as U^{C2Q-1} that synthesizes a corresponding quantum state with encoded classical data. U^{C2Q-1} is parameterized by the global scale *r*, global phase *t*, azimuth angle φ, and elevation angle θ. We also present an optimized state synthesis circuit U^{C2Q-2} that is characterized by unity global scale, i.e., $r=1$. The steps of the proposed methodology in the formation of the circuits U^{C2Q-1} and U^{C2Q-2} are elaborated in the next subsections.

3.2.1 Method 1: State Synthesis with Global Scale and Phase

A quantum register of n qubits that is in ground state is defined as $|\psi_0\rangle$=$|0\rangle^{\otimes n}$. Given a classical data set of $N = 2^n$ elements, the objective is to synthesize a target input quantum state given by $|\psi\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle$. Every i^{th} element from the classical data set will be encoded as a basis state coefficient α_i in the quantum state $|\psi\rangle$. For state-preparation or state-synthesis, it is required to find a quantum circuit, U^{C2Q-1}, that transforms the ground state $|\psi_0\rangle$ to the target state $|\psi\rangle$, i.e., $|\psi\rangle = U^{C2Q-1} \cdot |\psi_0\rangle$.

Any arbitrary single-qubit gate can be decomposed as a series of R_z and R_y gates known as the ZYZ or Pauli decomposition [16] [19]. Therefore, a qubit in ground state $|0\rangle$ can be transformed to any arbitrary state $|\psi\rangle$ by applying a rotation of angle θ about y-axis, followed by a rotation of angle ϕ about the z-axis, followed by a global scale and phase shift, see Fig. 6 and (11):

$$|\psi\rangle = R_z(\phi) \cdot R_y(\theta) \cdot re^{i\frac{t}{2}} \cdot |0\rangle \tag{11}$$

where $re^{i\frac{t}{2}}$ is an unobservable global quantity [16] for the single qubit, r is the global scale parameter [16], and t is the unobservable global phase shift [16]. If the coefficients of the target state $|\psi\rangle$ are α and β, such that $|\psi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}$ and $|0\rangle = \begin{bmatrix}1\\ 0\end{bmatrix}$, then the parameters $r, t, \theta,$ and ϕ for the transformation given in (11) could be determined by substituting the transformation matrices of R_z and R_y, see Fig. 3 , in (11), and are given by:

$$\begin{aligned} r &= \sqrt{\alpha^2 + \beta^2}, \qquad t = \angle\beta + \angle\alpha \\ \theta &= 2\tan^{-1}\left(\frac{|\beta|}{|\alpha|}\right), \quad \phi = \angle\beta - \angle\alpha \end{aligned} \tag{12}$$

where,

$$|\alpha| = \sqrt{\mathrm{Re}^2(\alpha) + \mathrm{Im}^2(\alpha)}, \quad \angle\alpha = \tan^{-1}\left(\frac{\mathrm{Im}(\alpha)}{\mathrm{Re}(\alpha)}\right),$$

$$|\beta| = \sqrt{\mathrm{Re}^2(\beta) + \mathrm{Im}^2(\beta)}, \quad \angle\beta = \tan^{-1}\left(\frac{\mathrm{Im}(\beta)}{\mathrm{Re}(\beta)}\right)$$

Using the Pauli decomposition described by (11) and the parameters obtained by (12), we derive a method for transforming any n-qubit register in the ground state $|\psi_0\rangle = |0\rangle^{\otimes n}$ to an arbitrary state $|\psi\rangle$, see Fig. 7. To synthesize the j^{th} pair of coefficients, or $|\psi_j\rangle$ in the state vector of $|\psi\rangle$, U_j is applied on a ground state $|\psi_0\rangle$, where $j = 0,1,2,\dots,(2^{n-1}-1)$. However, U_j cannot be applied to a single qubit in the n-qubit register to synthesize the corresponding pair of coefficients without also affecting the other coefficients in $|\psi\rangle$. Hence, each transformation U_j needs to be applied conditionally to synthesize the j^{th} pair of coefficients in the output state. The resulting conditional quantum circuit can be represented by a block-diagonal matrix U_{block}, of which each diagonal block is a 2×2 transformation matrix U_j, see Fig. 6 and (13). The elements of U_j are calculated using the parameters r_j, t_j, θ_j, and ϕ_j obtained from the j^{th} pair of coefficients using (12).

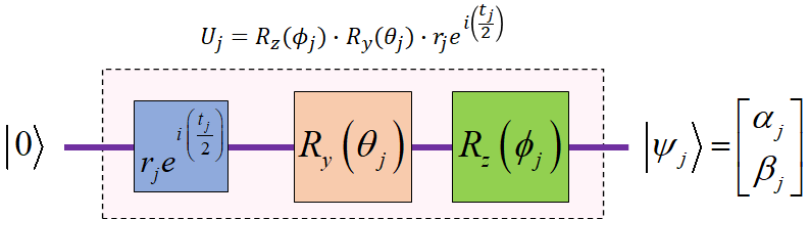


Fig. 6: Pauli decomposition for single-qubit state synthesis.

$$\begin{aligned} U_{block} &= U_0 \oplus U_1 \oplus \dots U_j \dots \oplus U_{(2^{n-1}-1)} \\ &= diag\left(U_0, U_1, \dots U_j, \dots, U_{(2^{n-1}-1)}\right) \end{aligned} \tag{13}$$

A block-diagonal matrix such as U_{block} can be implemented as a quantum multiplexer [44] [19] with n qubits of which $(n-1)$ are control qubits acting on the least significant target qubit.

The corresponding circuit is shown in Fig. 7. For each combination of the control qubits, the corresponding U_j transformation is applied on the target qubit q_0, where $j = 0,1,2,\dots,(2^{n-1}-1)$. To produce all combinations on the control qubits with equal probability, a set of H gates must be applied on the $(n-1)$ control qubits before applying the U transformation. The desired final state $|\psi\rangle$ is produced at the output with the target coefficients as a result of uniformly applying each U_j transformation on the least significant qubit. The overall transformation, U^{C2Q-1}, from ground state $|\psi_0\rangle = |0\rangle^{\otimes n}$ to $|\psi\rangle$ can be expressed by (14).

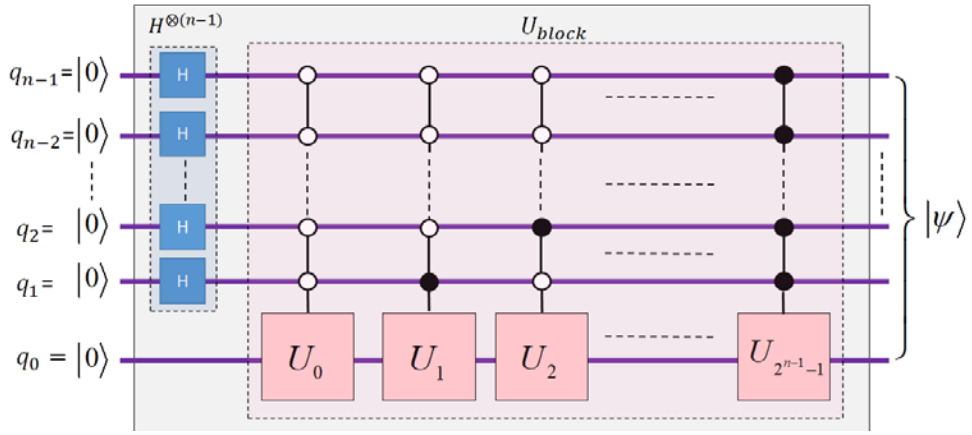


Fig. 7: Conditional logic-based quantum circuit for arbitrary state synthesis. The white and black circles on the control qubits represent bit values of zero and one respectively.

$$|\psi\rangle = U_{init} \cdot |\psi_0\rangle = U^{C2Q-1} \cdot |0\rangle^{\otimes n}, \text{ where}$$

$$U^{C2Q-1} = \left(\sqrt{2}\right)^{n-1} \cdot U_{block} \cdot \left(H^{\otimes(n-1)} \otimes I\right), \text{and} \qquad (14)$$

$$I \text{ is a } 2 \times 2 \text{ identity matrix}$$

Each U_j block is a sequence of a phase and scale shift, followed by y-rotation, followed by z-rotation as shown in Fig. 8, and U_j is calculated from the corresponding set of parameters $\{r_j, t_j, \theta_j, \phi_j\}$ obtained by (12). Since each set of operations are mutually exclusive from each other, we can separate them into uniformly controlled groups of phase and scale shifts, y-rotations, and z-rotations as shown in Fig. 9.

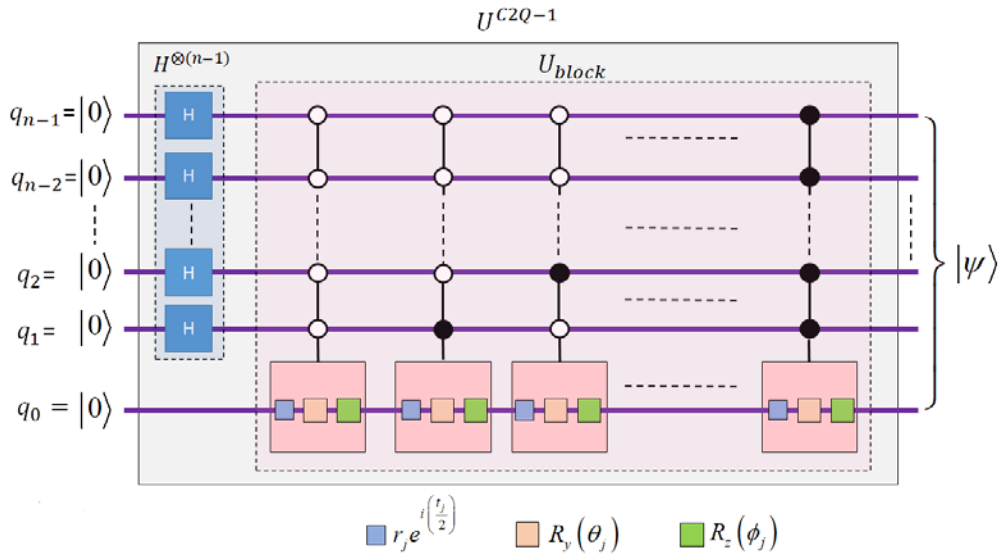


Fig. 8: Factorization of the U_j transformation.

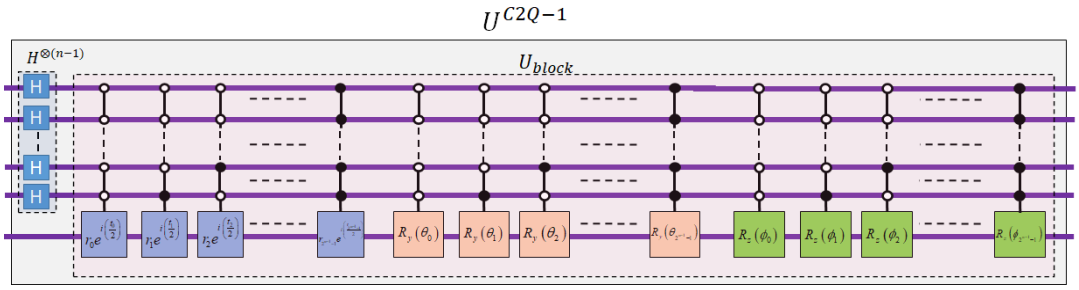


Fig. 9: Expanded full quantum circuit for arbitrary state synthesis.

To represent uniformly controlled operations as a single gate operation, we use a notation previously used in [19], where the sequence of different combinations on the control qubits are replaced with a `square box' notation indicating multi-control, and the parameterized operations for each combination are replaced by a single box denoting the general operation. We use this notation to simplify the circuit in Fig. 9 and the resulting circuit representation is shown in Fig. 10.

The uniformly controlled R_y and R_z rotation operations in Fig. 10 can be decomposed into a sequence of primitive CNOT and one-qubit rotation gates. A systemic decomposition method was presented in [45] which we leverage for our methodology. The method involves taking the binary reflected gray code of the control bit sequence to determine the control qubit positions of the

CNOT gates. As a demonstrative example, the decomposition for a 3-qubit controlled R_y operation with rotation angles θ_j is shown in Fig. 11. To calculate the new set of rotation angles $\widetilde{\theta_j}$ for the one-qubit rotations in Fig. 11, a transformation matrix $M_{ij}^{k} = (-1)^{b_{i-1} \cdot g_{j-1}}$ was formulated in [49]. The exponent of this matrix is the bit-wise inner product of the binary vectors for standard binary representation, b_{i-1}, and gray code representation g_{j-1}. Applying the inverse of M_{ij}^{k} on the vector of angles θ_j consequently produces a vector of angles $\widetilde{\theta_j}$. The decomposition for one uniformly controlled rotation operations takes 2^n gates (2^{n-1} CNOTs and 2^{n-1} one-qubit rotations) in total [19] [47] [49]. We apply this decomposition for the uniformly controlled R_y and R_z rotation operations in Fig. 10.

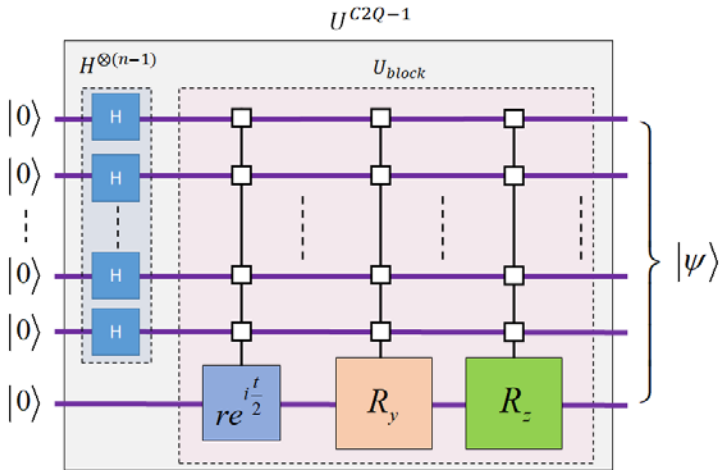


Fig. 10: Simplified full quantum circuit for arbitrary state synthesis.

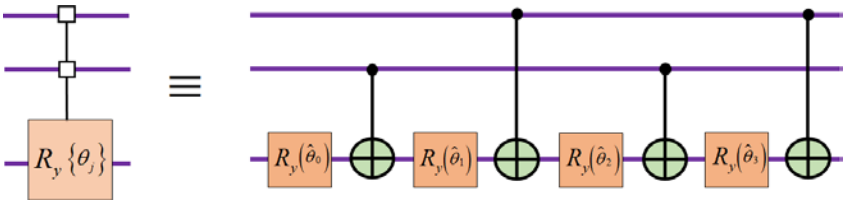


Fig. 11: Decomposition of a uniformly controlled 3-qubit R_y rotation operation.

3.2.2 Method 2: State Synthesis with Unity Global Scale

The proposed C2Q circuit in Fig. 10 considers the global scale and phase shift parameters, which are unobservable in actual quantum systems. We propose a more practical approach for synthesizing a quantum state in which the global scale is unity, i.e., $r = 1$. The corresponding optimized state-preparation circuit U^{C2Q-2} that is characterized by unity global scale is also presented.

Given $N = 2^n$ data points, we want to synthesize a target *n*-qubit quantum state $|\psi\rangle$ with *N* coefficients, $C_0, C_1, \ldots, C_{N-1}$, such that the coefficients are equal to the data points. From the given coefficients, we calculate an array of intermediate probabilities $P_{i,j}$ as shown in (15), for $0 < j < n-1$ and $0 < i < k_j - 1$, where $k_j = 2^{n-1-j}$.

$$P_{i,j} = \begin{cases} |C_{2i}|^2 + |C_{2i+1}|^2 & j = 0, 0 \le i < 2^{n-1} \\ P_{2i,j-1} + P_{2i+1,j-1} & 1 \le j < n, 0 \le i < 2^{n-1-j} \\ 0, & 2^{n-1-j} \le i < 2^{n-1} \end{cases} \tag{15}$$

Using the $P_{i,j}$ probabilities, we calculate the coefficient pairs $\alpha_{i,j}$ and $\beta_{i,j}$, as shown in (16) and (17).

$$\alpha_{i,j} = \begin{cases} \dfrac{C_{2i}}{\sqrt{P_{i,j}}} & P_{i,j} \neq 0, j = 0, 0 \le i < 2^{n-1} \\ \sqrt{\dfrac{P_{2i,j-1}}{P_{1,j}}} & P_{i,j} \neq 0, 1 \le j < n, 0 \le i < 2^{n-1-j} \\ 1, & P_{i,j} = 0 \end{cases} \tag{16}$$

$$\beta_{i,j} = \begin{cases} \dfrac{C_{2i}}{\sqrt{P_{i,j}}} & P_{i,j} \neq 0, j = 0, 0 \le i < 2^{n-1} \\ \sqrt{\dfrac{P_{2i+1,j-1}}{P_{1,j}}} & P_{i,j} \neq 0, 1 \le j < n, 0 \le i < 2^{n-1-j} \\ 0, & P_{i,j} = 0 \end{cases} \tag{17}$$

From the new coefficient pairs $\alpha_{i,j}$ and $\beta_{i,j}$, we calculate the parameters $\theta_{i,j}$, $\varphi_{i,j}$, $t_{i,j}$, and $r_{i,j}$, as shown in (18), required for transforming each qubit and synthesizing the target quantum state. This methodology results in values of $r_{i,j}$ being unity.

$$\begin{aligned} r_{i,j} &= \sqrt{\left|\alpha_{i,j}\right|^2 + \left|\beta_{i,j}\right|^2}, \qquad t_{i.j} = \angle\beta_{i,j} + \angle\alpha_{i,j} \\ \theta_{i,j} &= 2\tan^{-1}\left(\frac{\left|\beta_{i,j}\right|}{\left|\alpha_{i,j}\right|}\right), \quad \phi_{i,j} = \angle\beta_{i,j} - \angle\alpha_{i,j} \end{aligned} \tag{18}$$

where,

$$\left|\alpha_{i,j}\right| = \sqrt{\mathrm{Re}^2\left(\alpha_{i,j}\right) + \mathrm{Im}^2\left(\alpha_{i,j}\right)}, \;\; \angle\alpha_{i,j} = \tan^{-1}\left(\frac{\mathrm{Im}\left(\alpha_{i,j}\right)}{\mathrm{Re}\left(\alpha_{i,j}\right)}\right),$$

$$\left|\beta_{i,j}\right| = \sqrt{\mathrm{Re}^2\left(\beta_{i,j}\right) + \mathrm{Im}^2\left(\beta_{i,j}\right)}, \;\; \angle\beta_{i,j} = \tan^{-1}\left(\frac{\mathrm{Im}\left(\beta_{i,j}\right)}{\mathrm{Re}\left(\beta_{i,j}\right)}\right)$$

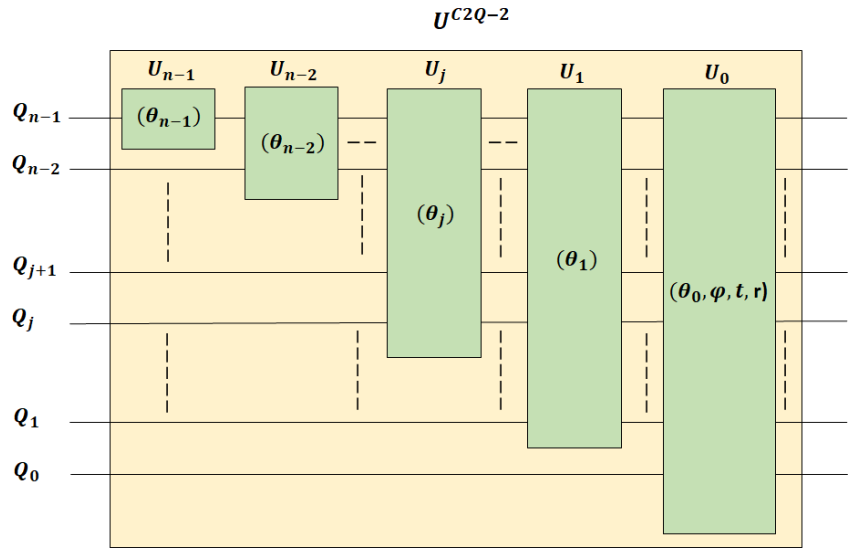
and,

$$j = 0, 1, \ldots, (n-1),$$

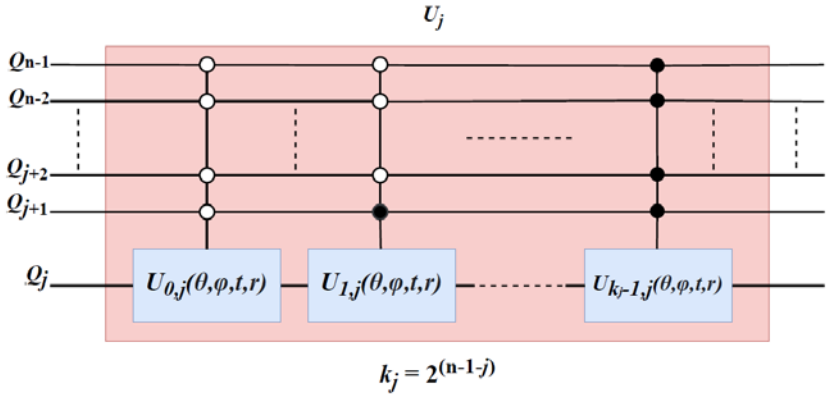
$$i = 0, 1, \ldots, \left(k_j - 1\right),$$

$$k_j = 2^{n-1-j}$$

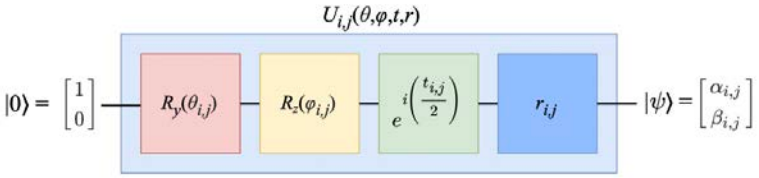
The corresponding quantum circuit U^{C2Q-2} consists of a series of operations U_j where $j = 0, 1, \ldots, (n-1)$, see Fig. 12. A set of $R_y(\theta_j)$ rotations is applied in the first $(n-1)$ operations U_j, where $j = (n-1), (n-2), \ldots, 2, 1$, followed by a set of $R_y(\theta_0)$ and $R_z(\varphi_0)$ rotations, and global phase shift t in U_0, see Fig. 12(a). Each U_j is a uniformly-controlled operation, shown in Fig. 12(b). As shown in Fig. 12(c), a number $(k_j = 2^{n-1-j})$ of $U_{i,j}$ rotation operations are applied for each U_j, where $i = 0, 1, \ldots, \left(k_j - 1\right)$, and $0 \leq j < n$. Each $U_{i,j}$ is a Pauli decomposition [16] operation requiring a 4-tuple of parameters (θ, φ, r, t) which are calculated from given classical data set $|\psi\rangle$ using the steps described previously in (15) to (18).



(a) Quantum circuit for C2Q data encoding with unity global scale.



(b) Uniformly-controlled operations.



(c) Pauli decomposition for single-qubit state synthesis.

Fig. 12: Quantum circuits for C2Q data encoding with unity global scale.

3.2.3 Analysis of Circuit Depth for Proposed Methods

To determine the circuit depths for the proposed C2Q circuits, we present two types of depth analysis: (a) considering uniformly-controlled gates, and (b) considering primitive 2-qubit CNOT

and 1-qubit rotation gates. For each type of analysis, we also consider two types of input data: (a) complex data, and (b) positive real data. The derived circuit depths for the proposed C2Q methods are summarized in Table 1.

Table 1: Analysis of circuit depths for proposed C2Q methods.

	C2Q Circuit Depth			
	Method 1		**Method 2**	
	Uniformly-controlled gates	**Primitive gates**	**Uniformly-controlled gates**	**Primitive gates**
Complex data	$2^{n+1}+1$	$2^{n+1}+\chi_r+\chi_t+1$	$2^{n+1}-1$	$2^{n+1}+2^n+\chi_t-3$
Positive real data	2^n+1	$2^n+\chi_r+1$	2^n-1	$2^{n+1}-3$

$\chi_r \equiv$ Gate depth of uniformly-controlled global scale gate
$\chi_t \equiv$ Gate depth of uniformly-controlled global phase gate

For method 1, the proposed U^{C2Q-1} circuit in Fig. 10 consists of four uniformly-controlled operations. Each operation has $n-1$ controls, therefore, there are 2^{n-1} combinations in each uniformly-controlled operation and depth of each is 2^{n-1}. The total depth for complex data is $4\times 2^{n-1}+1=2^{n+1}+1$ with the additional level for the H gates, as shown in Table 1. For positive real data, the uniformly-controlled global phase and R_Z operations are not required, i.e., R_Z becomes identity matrix, and therefore the total depth is reduced to $2\times 2^{n-1}+1=2^n+1$. If we consider decomposing the circuits into primitive 2-qubit CNOT and 1-qubit rotation gates, then each uniformly-controlled R_y or R_z operation can be decomposed into 2^n CNOTs and rotations. For this analysis, we denote the depth of the uniformly-controlled global scale and phase operations as χ_r and χ_t. Therefore, for the U^{C2Q-1} circuit in Fig. 10, the total depth for complex data is $2\times 2^n+\chi_r+\chi_t+1$, see Table 1. For positive real data, the total depth is reduced to $2^n+\chi_r+1$, see Table 1.

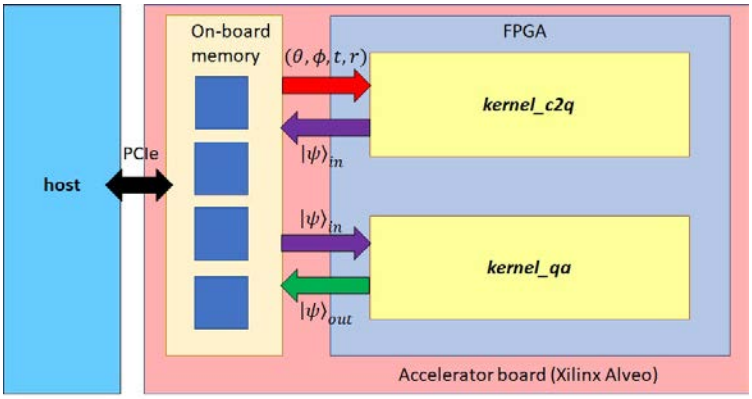
For method 2, the proposed U^{C2Q-2} circuit in Fig. 12 consists of $n \times U_j$ operations each having $(n-j-1)$ control qubits. The final block U_0 contains three uniformly-controlled operations, each of depth 2^{n-1}. The total depth can be derived as $1 + 2^1 + 2^2 + \cdots + 2^{n-1} + 2 \times 2^{n-1} = 2^{n+1} - 1$. For positive real data, the reduced depth is $2^n - 1$. For analysis with primitive gates, we denote the depth of uniformly-controlled global scale and phase operations as χ_r and χ_t as before. The decomposition of each U_j into primitive gates is 2^{n-j}. Therefore, the total depth is derived as $1 + 2^1 + 2^2 + \cdots + 2^{n-1} + (2^n + 2^n - 2 + \chi_t) = 2^{n+1} + 2^n + \chi_t - 3$, see Table 1. For positive real data, the reduced depth is $2^{n+1} - 3$, see Table 1.

We analyzed the complexities of the methods presented in prior works related to arbitrary state synthesis. A quantitative comparison of those methods with our proposed C2Q methods, in terms of the theoretical circuit depth, is shown in Table 2. In general, the previous methods proposed using uniformly controlled rotation operations to recursively disentangle each qubit, which results in larger gate count and depth. Our proposed methods result in reduction of circuit depth by at least a factor of two, see Table 2.

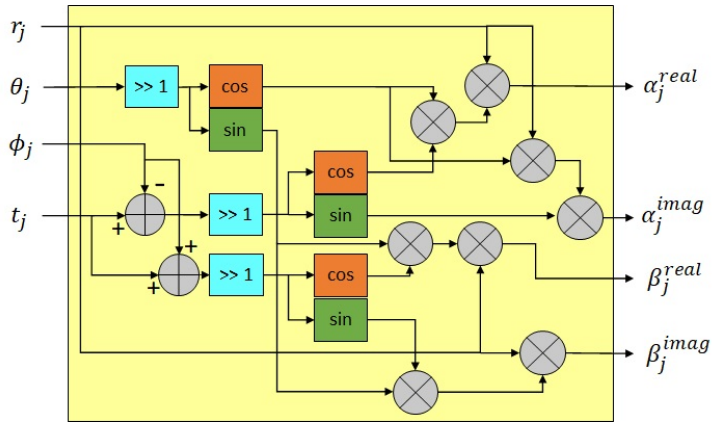
Table 2: Comparison of C2Q methods in terms of circuit depth.

Method	**Circuit Depth**
Mottonen [46], 2004	$2^{n+2} - 6$
Shende [19], 2006	$2^{n+2} + 2n$
Niemann [47], 2016	$2^{n+2} + 3n - 8$
Proposed C2Q Method 1	$2^{n+1} + 1$
Proposed C2Q Method 2	$2^{n+1} - 1$

3.3 Hardware Architectures for Emulating Classical-to-Quantum Encoding



(a) System architecture for emulation of quantum algorithms integrated with C2Q.



(b) Kernel architecture for C2Q-1

Fig. 13: Hardware architectures for emulating C2Q data encoding.

To evaluate the proposed C2Q methods and corresponding circuits, a hardware-based emulation model is proposed, and the hardware system architecture is shown in Fig. 13(a). The emulation model consists of two components: modeling C2Q data encoding, and modeling a quantum algorithm. These components are modeled in the architecture as reconfigurable hardware kernels, *kernel_c2q* and *kernel_qa* respectively, see Fig. 13. The sets of input parameters, $\theta_j, \phi_j, r_j, t_j$ where $j = 0, 1, 2, \ldots, \frac{N}{2} - 1$, and input/output state vectors $|\psi_{in}\rangle, |\psi_{out}\rangle$ are stored in the on-board memory and transferred to the kernel reconfigurable regions during computation. 32-bit floating point precision is used for storage and computation. The host machine controls memory

transfers and kernel execution commands. In this emulation model, the *kernel_c2q* is executed first, which operates on the input parameters and synthesizes the input quantum state $|\psi_{in}\rangle$. The parameter extraction from given data set is done by the host machine. The input quantum state vector is then transferred to the *kernel_qa* which performs the operations of the quantum algorithm and produces an output state vector $|\psi_{out}\rangle$ that is stored in the on-board memory. The hardware architecture of *kernel_c2q* is presented in Fig. 13(b) which emulates the operation of the proposed method 1 for C2Q, see Fig. 10. This architecture synthesizes the j^{th} pair of complex coefficients from input parameters $\theta_j, \phi_j, r_j, t_j$.

The proposed hardware architecture focuses on ease-of-use and maximizing scalability of the emulation. As state vectors are stored in the on-board memory, it allows emulation of larger quantum circuits and algorithms. On the other hand, I/O between the FPGA and memory can cause degradation of emulation speed. Faster emulation can be traded off for less memory space by storing the state vectors in the FPGA on-chip memories and directly buffering the vectors between the kernels. The latter method, however, results in a larger and more complex architecture that requires strict synchronization between hardware kernels.

Chapter 4: Quantum Algorithm Emulation

To emulate behavior of quantum algorithms, we investigated and developed different emulation models and techniques. We analyze the area and speed trade-offs for each model and discuss the advantages and disadvantages of the underlying techniques used.

4.1 Gate-based Emulation Model

Our primary objective was to develop a gate-based, modular framework which can be easily re-used for emulating large-scale quantum algorithms. The framework consists of a library of components of single qubit gates, e.g., Hadamard, Pauli X, Pauli Z, etc. [11], and multi qubit gates, e.g., CNOT, SWAP, Controlled Phase Gate, etc. [11]. In modeling qubits, we used the data structure shown in Fig. 14, where α and β are complex coefficients. To build an accurate emulator that matches the precision of software simulators, we use 32-bit floating-point numbers in our calculations to represent the real and imaginary components of each complex coefficient.

α_{real}(32-bit)	α_{imag}(32-bit)	β_{real}(32-bit)	β_{imag}(32-bit)

Fig. 14: Data structure for storing information for a single qubit.

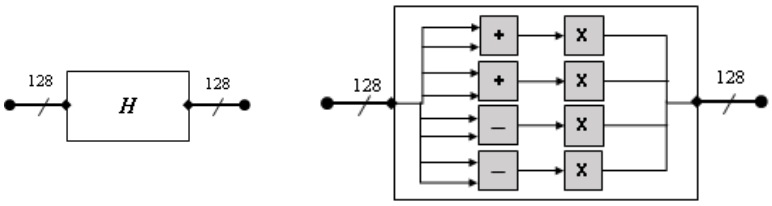
4.1.1 Modeling Quantum Gates

To demonstrate our modeling approach, we take the *H* gate as an example. The matrix representation of the *H* gate is given in Fig. 3 and the vector representation of a qubit is given in (2). The operation of the *H* gate on a qubit can be defined by matrix multiplication as described in (19).

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}.\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}\alpha+\beta\\ \alpha-\beta\end{bmatrix} \tag{19}$$

The *H* gate component is provided two 128-bit data bus for input and output respectively, as shown in Fig. 15(a). The dataflow operations for the *H* gate is shown in Fig. 15(b). The component performs unpacking operations to extract the real and imaginary parts of the coefficients from the

data bus. It then performs the necessary math operations on the real and imaginary components, in this case, four additions and four multiplications. It finally performs packing operations to prepare the output data in the same format and sends it out via the output bus. This structure has been used as a template for creating components for other quantum gates.

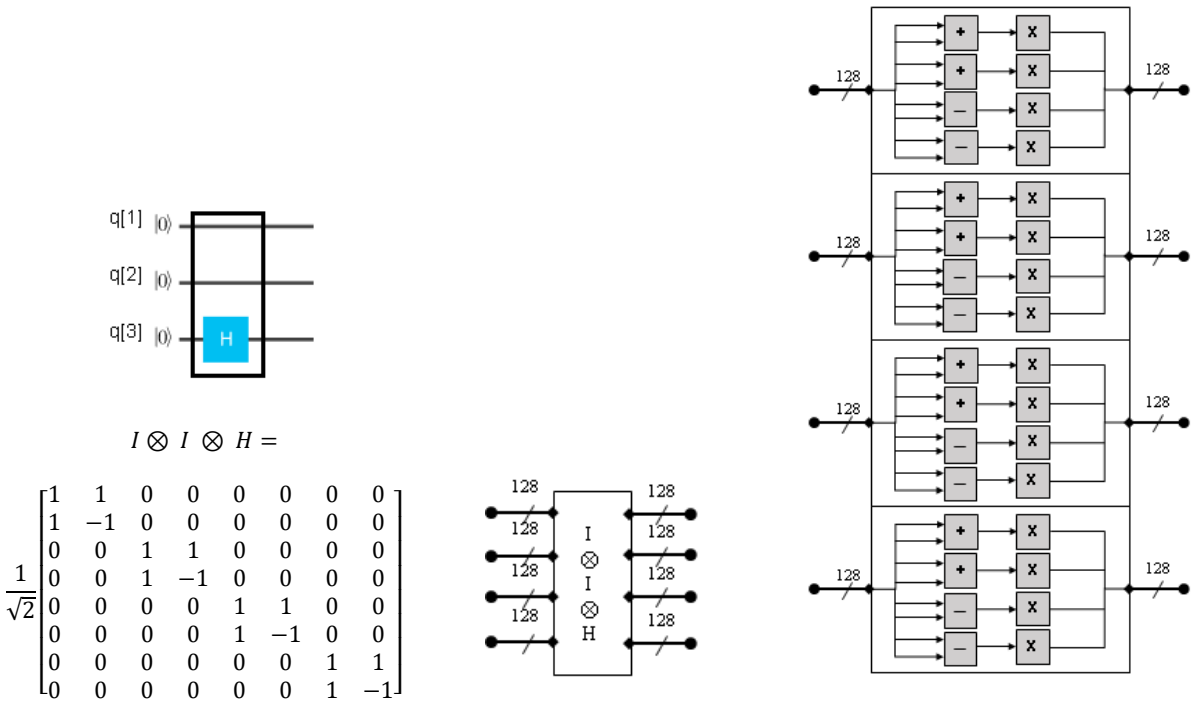


(a) *H* gate component (b) Dataflow model for *H* gate

Fig. 15: Emulating Hadamard (*H*) gate.

4.1.2 Modeling Tensor Operations

To model quantum operations, we also need components representing tensor operations. For example, for the tensor operation shown in Fig. 16(a) for 3 qubits, the corresponding hardware design is shown in Fig. 16(b). The $I \otimes I \otimes H$ operation, defined by an 8×8 matrix, is a transformation on the coefficients, which are represented as a state vector, of any quantum state. Hence the component for this operation will take 8 complex coefficients as input and produce 8 complex output coefficients. To accommodate this many coefficients, four 128-bit data buses are used as inputs and outputs. Accordingly, from the transformation matrix, see Fig. 16(a), the dataflow model is derived as shown in Fig. 16(c).



(a) Transformation on entangled qubits represented as a tensor product.

(b) Component for tensor operation I ⊗ I ⊗ H

(c) Dataflow model for I ⊗ I ⊗ H operation

Fig. 16: Emulating tensor operations.

4.1.3 Modeling Quantum Circuits

The size of the equivalent hardware model of a quantum circuit will grow exponentially with the increase in the number of qubits [29]. Hence, we can only emulate a limited number of qubits using a single FPGA. In our work, we have used full 32-bit floating-point precision which increases the resource consumption on the FPGA compared to previous fixed-point implementations [27] [39] [28], [29] [30]. If there is no scope of further design optimization, then to improve scalability of the design, the resources can be distributed among multiple FPGA nodes. The design model is partitioned, and data can be passed between partitioned systems or nodes via a low-latency and high-bandwidth network, as shown in Fig. 17. A number of reconfigurable nodes

can be stacked according to the requirements and size of the model. We discuss modeling of a 5-qubit QFT circuit in order to demonstrate this approach.

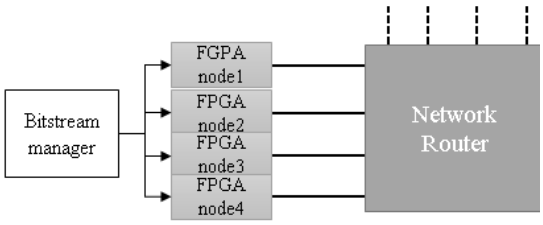
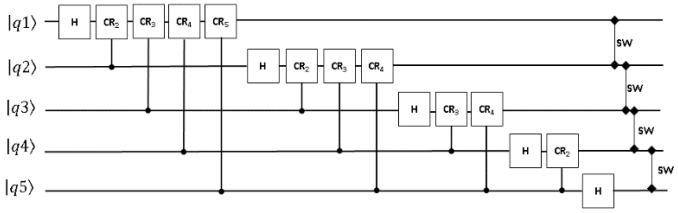


Fig. 17: Hardware architecture for design space sharing.



$$T1 = H \otimes I \otimes I \otimes I \otimes I$$
$$T2 = CR_2 \otimes I \otimes I \otimes I$$
$$T3 = (I \otimes SW \otimes I \otimes I).(CR_3 \otimes I \otimes I \otimes I).(I \otimes SW \otimes I \otimes I)$$
$$T4 = (I \otimes I \otimes SW \otimes I).(I \otimes SW \otimes I \otimes I).(CR_4 \otimes I \otimes I \otimes I).(I \otimes SW \otimes I \otimes I).(I \otimes I \otimes SW \otimes I)$$
$$T5 = (I \otimes I \otimes I \otimes SW).(I \otimes I \otimes SW \otimes I).(I \otimes SW \otimes I \otimes I).(CR_5 \otimes I \otimes I \otimes I).(I \otimes SW \otimes I \otimes I).(I \otimes I \otimes I \otimes SW)$$

Fig. 18: Modeling a 5-qubit Quantum Fourier Transform circuit using tensor operations.

Fig. 18 shows the quantum circuit for 5-qubit QFT. Derivation of the QFT circuit can be found among previous works [28] [29] [30]. The circuit consists of Hadamard gates (*H*), Controlled-Phase shift gates (CR_2, CR_3, CR_4, and CR_5) and SWAP gates (SW) [24]. The *H* gate puts the qubit in a superposition state. The CR_2, CR_3, CR_4, and CR_5 gates shift the phase of the qubit by $\frac{\pi}{2}, \frac{\pi}{4}, \frac{\pi}{8}$, and $\frac{\pi}{16}$ respectively, depending on the control qubit. The SW gate simply swaps the coefficients of two qubits. According to the gate-based quantum computing approach, the circuit can be modeled as a series of transformations. The first five transformations are shown in Fig. 18. It should be noted that for *T3*, *T4*, and *T5*, additional SW gate operations are required to enable controlled-phase shift operations on adjacent qubits [28]. To model this circuit for hardware, the tensor components designed in our library are used to build a dataflow model representing the series of

transformations. Once a complete dataflow model consisting of multi components is developed, it can be conveniently split into necessary number of partitions for implementation on the multi-node architecture in Fig. 17. Fig. 19 shows the dataflow model for 5-qubit QFT split into three partitions for a tentative three-node architecture.

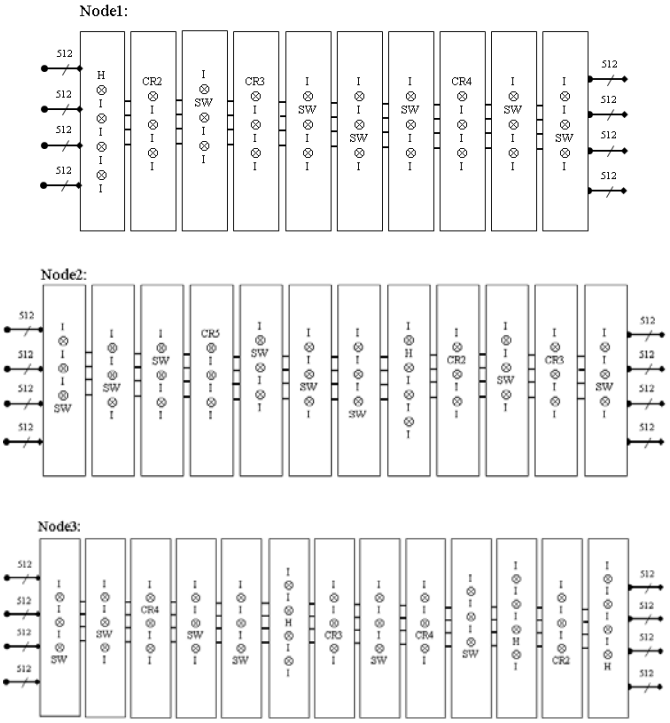


Fig. 19: Space-shared (partitioned) hardware models for 5-qubit QFT circuit.

For the case of quantum circuits such as QFT, there is limited scope of design optimization and resource sharing as each part of the circuit has a different set of operations relative to each other. In other words, QFT is not inherently regular or uniform in terms of hardware structures among temporal evaluation iterations. Therefore, it is not possible to exploit or apply temporal resource sharing for QFT. However, for the case of Grover's search algorithm, the algorithm consists of multiple iterations of the same set of circuit operations. Hence, it is possible and essential to make use of both space and time-sharing techniques in order to minimize resource utilization for the design of larger-scale hardware models of Grover's algorithm while also

maximizing its throughput. In general, we design a hardware architecture for modeling quantum circuits that require temporal iterations of one or more functions. This is illustrated in Fig. 20 for a generic algorithm that consists of more than one function or stage.

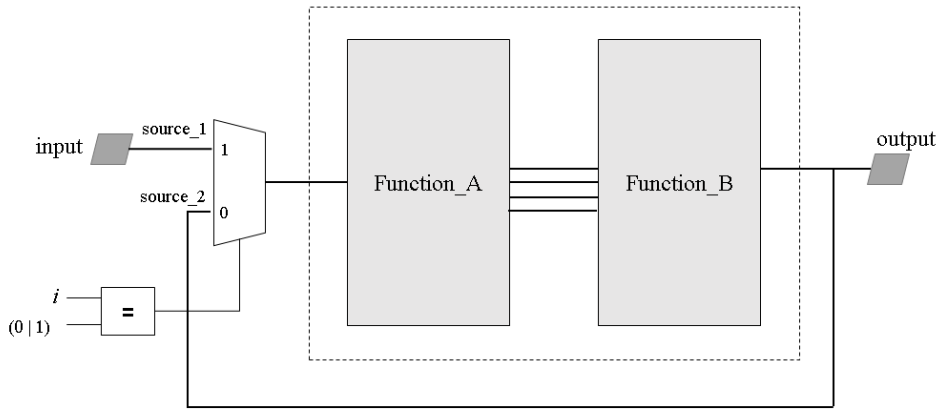


Fig. 20: Hardware architecture for space-time sharing.

To elaborate the working concept of the architecture we take the example of Grover's search algorithm. As discussed in Chapter II, Grover's search algorithm consists of two prime stages which are repeated for $\sqrt{N}$ iterations to reach the solution. The stages phase inversion and inversion about mean are mapped to *function_A* and *function_B* respectively, see Fig. 5 and Fig. 20. The input data to the circuit is selected by a multiplexer between *source_1*, which is the external input, and *source_2*, which is the feedback from the output of a previous iteration of evaluation. The select signal to the multiplexer comes from a comparator that compares an iteration variable *i* (that is incremented every clock cycle) to 0 or 1. This is done to ensure that *function_A* will accept input from source_1 during the first two clock cycles of every iteration cycle, while for other values of *i*, the input will come from *source_2*. This technique of non-linear pipelining ensures that at each clock cycle both stages are doing useful work, thus maximizing throughput and pipeline efficiency [50]. This is illustrated by the function reservation table [50] shown in Table 3. As shown in Table 3, in the first clock cycle, *Cycle1*, *function_A* takes data *D1** from source_1 while *function_B* is

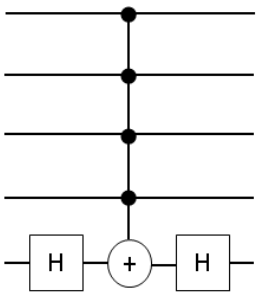
idle. In the next cycle *Cycle2*, *function_B* processes D1 from *function_A*, while *function_A* accepts a new set of data, *D2**. From each cycle onwards, the two stages continue to work these sets of data until the number of iterations required by the algorithm is completed. Here it is assumed that each iteration is equivalent to two clock cycles for each stage.

Table 3: Reservation Table of Non-Linear Pipelined Architecture

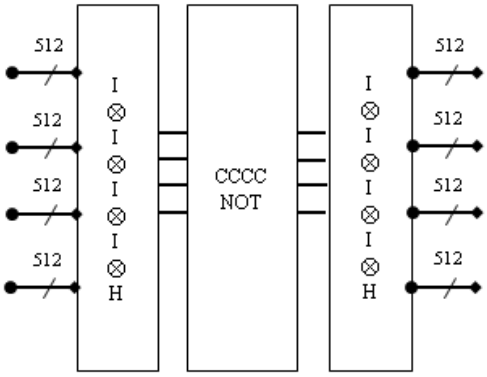
	Cycle1	*Cycle2*	*Cycle3*	*Cycle4*	*Cycle5*	*Cycle6*	*Cycle7*
function_A	D1*	D2*	D1	D2	D1	D2	D3*
function_B		D1	D2	D1	D2	D1	D2
	iteration 0 (D1)		*iteration 1 (D1)*		*iteration 2 (D1)*		···

In modeling the stages of Grover's circuit, we propose using a hybrid approach. We model the oracle function using pure quantum gates, while the Grover diffusion function is modeled using arithmetic functions. This hybrid approach ensures true quantum behavior is emulated in the oracle search function while significant computational resources are saved in using the arithmetic model for the diffusion function. Fig. 21(a) shows a quantum gate-based oracle circuit for detecting the binary string "*11111*". We develop the equivalent hardware model using components as shown in Fig. 21(b). The quantum gates used in this model are *H* gates and a quadruple controlled NOT (*CCCCNOT*) gate derived from a triply controlled Not (*CCCNOT*) gate [51]. We choose the gate or component-based design approach for the oracle function so that it can be conveniently modified for detecting different patterns for other functions. Fig. 21(c) shows the data flow model of the *inversion about mean* stage, otherwise known as the Grover diffusion function. The design is parameterized by the number of qubits *n*. α_1 to α_N are the complex coefficients of the qubits of the system, where $N = 2^n$. The model operation consists of summing the complex coefficients and right-shifting the sum by *n-1* bits to obtain 2μ, where μ is the mean. The final part of the transformation

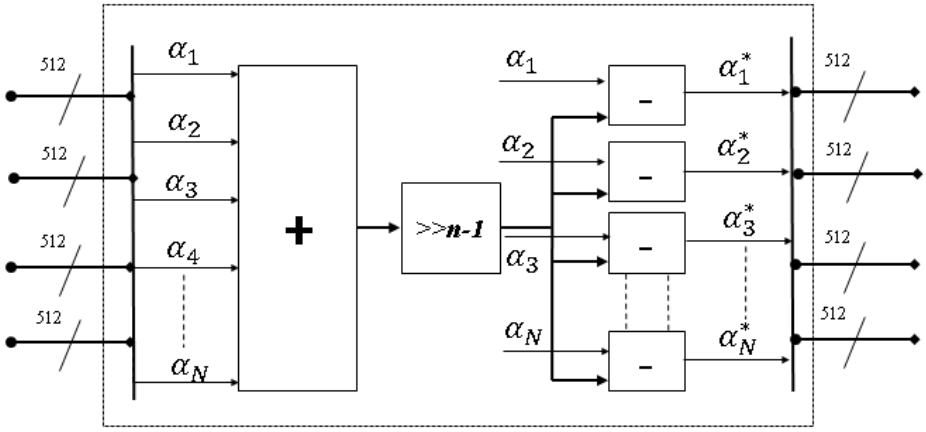
is $2\mu - \alpha_i$, see Fig. 4. The design depth is determined by N while the design width remains unchanged for any number of qubits.



(a) Oracle circuit for detecting "11111".



(b) Component-level view of oracle hardware model



(c) Data flow for implementing Grover diffusion function.

Fig. 21: Hardware models for 5-qubit Grover's search algorithm.

4.2 CMAC-based Emulation Model

The gate-based emulation model results in low scalability, as the hardware resource utilization increases exponentially with circuit size, i.e., number of qubits and number of stages (cascaded gates). We investigated and developed a more scalable, generalized emulation model that is optimized in terms of resource utilization and emulation time. A quantum algorithm is a series of transformations on the entangled quantum state of the qubits. The series of transformations can be represented as a single unitary complex-valued matrix, U_{ALG} [16]. An input quantum state, $|\psi_{in}\rangle$,

can be represented by a state vector comprising of the complex coefficients of the basis states of the quantum state. A complex vector-matrix multiplication of the input vector with the algorithm matrix produces the output quantum state vector, $|\psi_{out}\rangle$, whose coefficients represent the basis states of the output quantum state. We use this approach, illustrated in Fig. 22, as a model for designing hardware architectures for the proposed quantum emulation framework. This model is generalized and can be used to emulate any quantum algorithm/circuit that can be reduced to a single unitary operation, i.e., the transformation matrix can be pre-computed (lookup) generated dynamically, or streamed in from an external memory source.

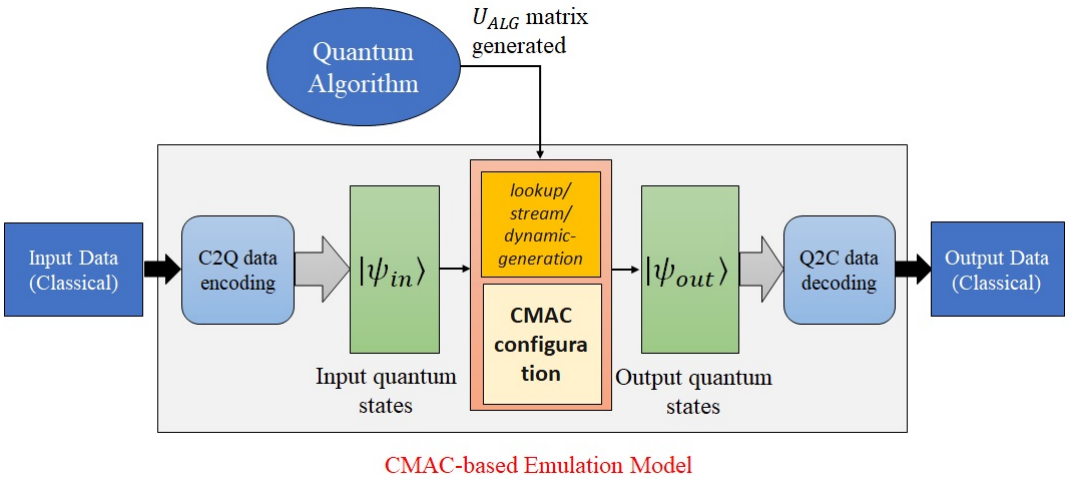


Fig. 22: CMAC-based emulation model

By reducing the algorithm/circuit to a single transformation and performing the necessary vector-matrix product, the corresponding hardware implementation becomes independent of the circuit depth, resulting in a space- and time-efficient emulation architecture. This methodology assumes that the algorithm matrix is known and pre-computed, or can be dynamically generated. A limitation of this methodology is that for some algorithms, pre-computing and storing the algorithm matrix may not be feasible as the matrix dynamically changes with the algorithm input, for example, Shor's algorithm [4]. Dynamically generating the matrix is also difficult for

algorithms with no pattern in the matrix elements, but it is certainly doable. To mitigate the limitations of pre-computing or dynamically generating the matrix and account for dynamically changing algorithm inputs and matrices, we incorporate data *streaming* techniques for emulation as elaborated in the next sections.

4.2.1 CMAC Architectures

To implement complex-valued vector-matrix multiplications on hardware (FPGA), we use a generic complex multiply-and-accumulate (CMAC) unit, as shown in Fig. 23. The inputs of the unit are complex values, i.e., elements of the input state vector, $|\psi_{in}(j)\rangle$, and of the algorithm matrix, $U_{i,j}$. The complex values are represented using 64 bits, with 32 floating-point bits for each of the real and imaginary parts. The benefit of using a CMAC is that different computation techniques, each with space and time trade-offs, can be applied during computation. To operate on complex values, the internal components of the CMAC (such as the multiplier and adder) have been designed for complex operations. The CMAC operations are described in (20). One CMAC unit performs, in total, four multiplications and four additions, see Fig. 23.

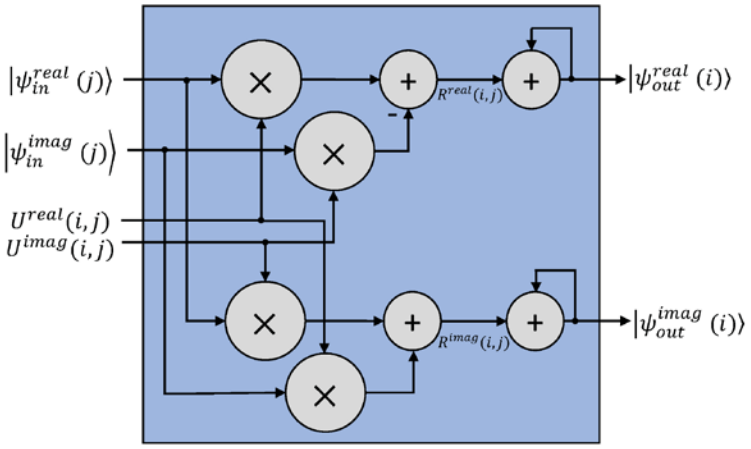


Fig. 23: Complex multiply-and-accumulate unit.

$$\psi_{out}^{real}(i) = \sum_{j=0}^{N-1} R^{real}(i,j)$$
$$\psi_{out}^{imag}(i) = \sum_{j=0}^{N-1} R^{imag}(i,j) \quad (20)$$

where,

$$i = 0,1,2,\dots,(N-1)$$
$$R^{real}(i,j) = \left(\psi_{in}^{real}(j) \times U^{real}(i,j)\right) - \left(\psi_{in}^{imag}(j) \times U^{imag}(i,j)\right), \text{and}$$
$$R^{imag}(i,j) = \left(\psi_{in}^{imag}(j) \times U^{real}(i,j)\right) + \left(\psi_{in}^{real}(j) \times U^{imag}(i,j)\right)$$

We explored different hardware architectures, as listed in Table 4, by varying the number of CMAC instances. The purpose of this design space exploration was to implement either fully resource-optimized or fully latency-optimized designs to find an optimized CMAC configuration for developing a scalable hardware emulation framework. Space and time complexities for these architectures are also summarized in Table 4.

Table 4: Space and Time Complexities of CMAC Architectures

CMAC Architecture	**Complexity**	
	Space (O_s)	**Time (O_t)**
Single	$O(1)$	$O(N^2)$
N-concurrent	$O(N)$	$O(N)$
Dual-sequential	$O(1)$	$O(N^2)$

Single-CMAC Architecture: For a fully resource-optimized design, we instantiate only one CMAC unit and feed it with one algorithm matrix element and one input quantum state vector element for each clock cycle. This is repeated for all N^2 items in the U_{ALG} matrix. For this architecture, the time complexity is $O(N^2)$, as shown in (21), where T_{clock} is the clock period. The hardware takes N cycles to store each input coefficient and N^2 cycles to process each element of the algorithm matrix, in addition to some initial latency L_1. The space complexity is $O(1)$, as shown in (22), since a single CMAC instance is being used.

$$O_{time} = (L_1 + N + N^2) \times T_{clock} = O(N^2) \quad (21)$$

$$O_{space} = 1 \times CMAC = O(1) \tag{22}$$

N-Concurrent-CMAC Architecture: In a fully parallel implementation, N CMAC instances are used to operate concurrently, for processing each row of the U_{ALG} matrix. The time complexity of this design, as shown in (23), is effectively $O(N)$ as it takes N cycles to store the input states, and N more cycles to concurrently process all N rows of the algorithm matrix, along with initial latency L_2. The space complexity now becomes $O(N)$, due to the N instances of CMAC units, as shown in (24).

$$O_{time} = (L_2 + 2N) \times T_{clock} = O(N) \tag{23}$$

$$O_{space} = N \times CMACs = O(N) \tag{24}$$

Dual-sequential-CMAC Architecture: In this implementation, two CMAC instances are utilized sequentially. After the initial latency L_3, the first CMAC processes the first row of the matrix while the input vector is being stored, and the second CMAC instance continues the subsequent processing of the remaining rows using the stored inputs. This implementation has double the resource requirements of the first architecture but has the benefit of improvement in execution time. The time complexity is determined as in (25) and the space complexity is given by (26).

$$O_{time} = (L_3 + N^2 - 1) \times T_{clock} = O(N^2) \tag{25}$$

$$O_{space} = 2 \times CMACs = O(1) \tag{26}$$

4.2.1 CMAC Computation Techniques

To investigate trade-offs in area and speed of the emulator, we leverage three computation techniques: *lookup, dynamic generation,* and data *streaming,* and apply them for the CMAC architecture. Benefits and drawbacks of each technique along with memory consumption analysis is discussed in the following sections.

Lookup-based CMAC: Look-up-tables (LUTs) simplify hardware design by replacing complex parts of computation with simple array-indexed operations. It is generally implemented as an array in memory that stores pre-calculated values which results in low resource requirements. In the CMAC architecture, we use the process of *lookup* to fetch pre-computed algorithm matrix values from memory during complex computation operations. A limitation of this technique is that for some algorithms, the algorithm matrix changes dynamically with the inputs, hence this method will not be feasible in those cases. We combine this *lookup* approach with the *dual-sequential-CMAC architecture* to optimize the design in terms of speed. The total memory, M_L, required using this combination is derived in (27), assuming 32-bit floating point numbers are used for each real and imaginary component of the complex matrix and vector elements.

$$M_L = M_{vec} + M_{mat} = 8N + 8N^2 = 2^{n+3} + 2^{2n+3} \tag{27}$$

Dynamic Generation-based CMAC: The *lookup* approach is optimized for speed, but storing the algorithm matrix consumes resources that increase exponentially with circuit size. For resource utilization optimization and improved scalability, we propose integrating the *dual-sequential-CMAC* architecture with a dynamic approach that involves generating the algorithm matrix values at runtime, storing only input vectors in memory. The advantage of this method is that it significantly reduces the total memory utilization, M_{DG} of the simulation, as shown in (28). The algorithm matrix U_{QFT}, see Fig. 4, is generated as part of the architecture using dedicated hardware units as shown in Fig. 24. The operations of this architecture are summarized in (29).

$$M_{DG} = M_{vec} = 8N = 2^{n+3} \tag{28}$$

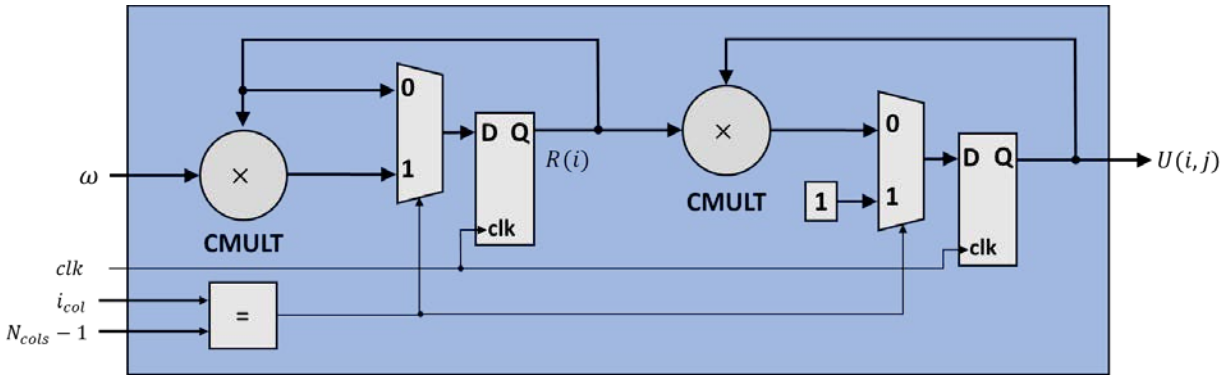


Fig. 24: Hardware architecture for dynamic generation of the QFT algorithm matrix

$$R(i) = R(i-1) \cdot \omega \qquad (29)$$
$$U(i,j) = U(i,j-1) \cdot R(i)$$

where,

$$\omega = e^{\hat{\imath}\frac{2\pi}{N}} = \cos\left(\frac{2\pi}{N}\right) + \hat{\imath} \cdot \sin\left(\frac{2\pi}{N}\right),$$

$$i, j = 0 \rightarrow N-1,$$

$$\text{and } R(0) = 1, \qquad U(0,0) = 1$$

The drawback of this technique is that the dynamic generation hardware can introduce pipeline latencies depending on the complexity of the algorithm, degrading the speed of the overall emulation. Furthermore, designing a dedicated hardware generation unit also requires us to find and exploit some pattern in the algorithm matrix, which might not be possible in every case. Generally, it is hard to efficiently generate the matrix values if the algorithm matrix doesn't have a special structure, therefore the generation hardware would be complex, and the approach may not be feasible for particular algorithms.

Stream-based CMAC: While the *lookup* approach improves speed, it sacrifices area, and similarly, while *dynamic generation* improves area, it sacrifices speed. We investigate a more optimal approach that sustains both speed and area improvements and improves scalability and latency of

the emulator. Instead of being stored into on-chip resources (OCR) or on-board memory (OBM), or dynamically generated during computation, the algorithm matrix elements are streamed in during computation as an input stream from an external control processor. The cost of streaming is typically the I/O channel latency between the control processor and the FPGA, which is negligible relative to the compute time necessary for processing the algorithm matrix. The contribution of this technique is that it greatly reduces the constraint on memory requirement compared to the *lookup*-based technique, while also avoiding the hardware cost and bottleneck of using the *dynamic generation* technique. As a result, a significantly higher number of qubits can be emulated on the same FPGA area. The total memory requirement, M_S, using this method is equivalent to M_{DG} and shown in (30). The top-level view of the emulator design using the data streaming technique is shown in Fig. 25.

$$M_S = M_{DG} = M_{vec} = 8N = 2^{n+3} \tag{30}$$

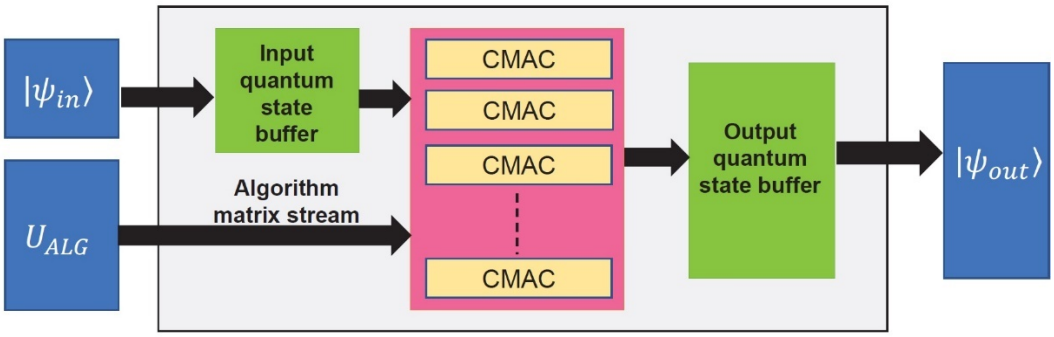


Fig. 25: Architecture of the stream-based CMAC quantum emulator.

4.3 Kernel-based Emulation Model

While the CMAC emulation model is suitable for modeling dense algorithm matrices, a faster, kernel-based model can be applied for more sparse matrices that involve a repeated set of core operations. The core operations are modeled as a kernel using either quantum gates or classical logic/arithmetic. The total input states are divided into groups and the kernel operation is applied

iteratively across all groups, one group every clock cycle. The hardware model for emulation using this approach is shown in Fig. 26. As an example, the one-dimensional Quantum Haar Transform (1D-QHT) algorithm is modeled for emulation.

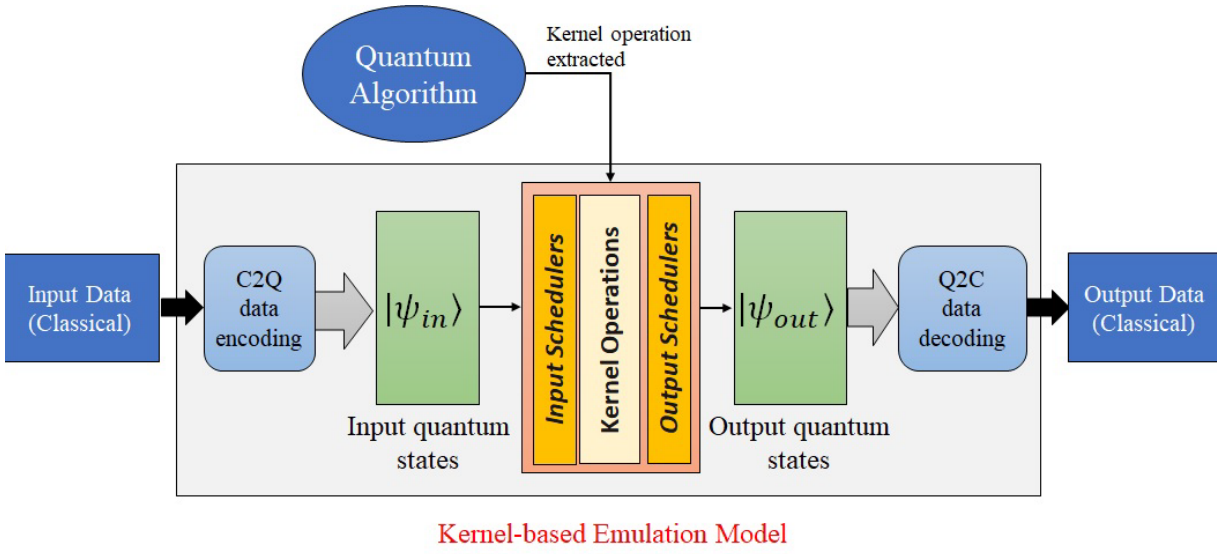


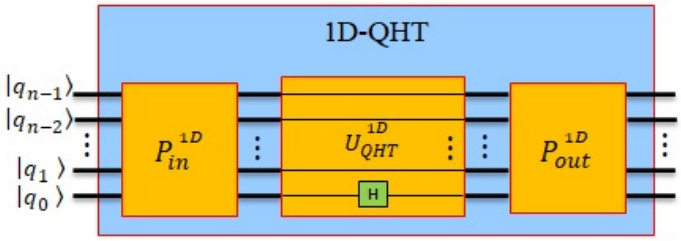
Fig. 26: Kernel-based model for quantum algorithm emulation.

In the gate-based emulation model, QHT is emulated using Hadamard and SWAP gate models. Alternatively, the proposed kernel-based model can be applied in the emulation of QHT to significantly reduce hardware resource utilizations, hardware latencies, and improve the time-complexity. We propose simplified hardware kernels for implementing 1D-QHT and inverse 1D-QHT (1D-IQHT).

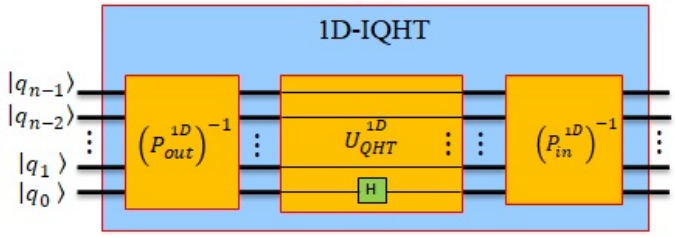
$$QHT^{1D} = P_{out}^{1D} \cdot U_{QHT}^{1D} \cdot P_{in}^{1D}$$
$$IQHT^{1D} = (P_{in}^{1D})^{-1} \cdot U_{QHT}^{1D} \cdot (P_{out}^{1D})^{-1} \quad (31)$$

For modeling 1D-QHT, the core operation, i.e. the Haar wavelet function, represented by U_{QHT}^{1D}, see (7e), is reduced to a common kernel operation described by H, which will be applied iteratively across groups of state coefficients. The Haar wavelet function is preceded and followed by permutations on the state coefficients as shown in (31), where P_{in}^{1D} is a perfect shuffle permutation

[35] on the input states, and P_{out}^{1D} is a perfect shuffle permutation operation on the output states. The inverse 1D-QHT operation (1D-IQHT) can be achieved by using inverse permutations, and the same Haar wavelet function, also shown in (31). Hardware emulation models for 1D-QHT and 1D-IQHT are presented in Fig. 27. The steps of our proposed algorithm for 1D-QHT operation are elaborated in Algorithm 1 in the appendix. For emulation, we model 1D permutations for hardware using classical logic to save resources, instead of quantum CNOT gate models, as done in related works. The 1D-QHT kernel is modeled using basic arithmetic operations and integrated with the input and output permutations, which are modeled using dedicated input/output hardware schedulers, see Fig. 26. Detailed architectures of the kernel and hardware schedulers are discussed in later chapters.



(a) Emulation model for 1D-QHT.



(a) Emulation model for 1D-IQHT.

Fig. 27: Algorithm and hardware kernel architectures for emulation of 1D-QHT.

Chapter 5: Quantum-to-Classical Decoding

In this chapter, the existing approaches for Quantum-to-Classical (Q2C) data decoding are discussed. A new methodology for Q2C, based on using Quantum Haar Transform (QHT), is also proposed and discussed.

5.1 General Approach

In classical systems, measurement of a state is predictable and deterministic. However, in quantum systems, the outcome of measurement of a quantum state is unpredictable and non-deterministic. A quantum state resides in a superposition of its basis states and measurement collapses the superposition and projects the quantum state to one of the basis states. For example, when the quantum state $|\psi\rangle$ described in (3) is measured, the probability of the outcome being the basis state $|i\rangle$ is $|C_i|^2$. This kind of measurement for which the result is one out of a set of basis states is called a *von Neumann* measurement [52]. In any quantum system, there are measurement gates which observe and project the state of a qubit or qubits onto a classical bit or register. Generally, to extract useful classical information, a quantum circuit is executed multiple times and the output is sampled for each execution by performing a measurement. The greater the number of measurements, the more accurate is the extracted information.

Classica data for quantum computation is generally encoded as the set of amplitudes $\{C_i\}$ of the basis states $|i\rangle$. Quantum-to-Classical (Q2C) data decoding is the process of reconstructing the amplitudes of the final quantum state by measurement. The general approach for Q2C is to determine the set of probabilities $\{|C_i|^2\}$ of measuring the basis states $|i\rangle$, from which the set of amplitudes $\{C_i\}$ can be obtained. The quantum circuit is iteratively sampled multiple times, and the basis state outcome is measured for every iteration. The frequency of occurrence of each basis

state is recorded and using that count, a probability distribution or a histogram is constructed. The basis state amplitudes can then be calculated from the probability distribution.

In applications such as quantum image processing, decoding image data from the final quantum state requires high number of samples or iterations of the quantum circuit. Usually, the number of iterations required to accurately recover the image data is in the range of thousand samples. Performing a large number of circuit executions introduces significant overhead and has an adverse effect on the execution time. In addition, the accuracy or fidelity of the reconstructed amplitudes is low due to statistical errors that arise as a result of the finite number of measurements/sampling of the non-deterministic outcomes of the quantum system.

5.2 Quantum-to-Classical Decoding Using Quantum Fourier Transform

An alternative approach to Q2C data decoding for image processing applications was proposed in [16]. Instead of reconstructing all the amplitudes from the quantum state, the proposed approach was extracting a collective property from the amplitudes stored in the quantum state. For example in classical image processing, one can obtain useful information about any image by applying Fourier transform and projecting it in the frequency domain. Similarly, in quantum image processing, by applying the quantum Fourier transform (QFT) we can observe the frequency components of an image, without having to measure all the pixel data. For example, consider an image represented as a one-dimensional time series of pixels and encoded as the amplitudes of a quantum state. A QFT circuit is applied and then the output is measured. It is likely that the resulting basis state outcome will correspond to a peak in the Fourier transformed image. This suggests that the corresponding frequency component is strongly represented in the Fourier transform of the image [16]. For specific image processing applications, this measurement in the Fourier bases gives us information about properties of the transformed image without having to

decode the actual image pixels. The number of iterations/samples of the quantum circuit will be relatively less than those required in the general approach, as we are no longer reconstructing the exact and complete probability distribution, but are only interested in measuring the Fourier transformed basis states. The QFT-based methodology for Q2C decoding is illustrated in Fig. 28.

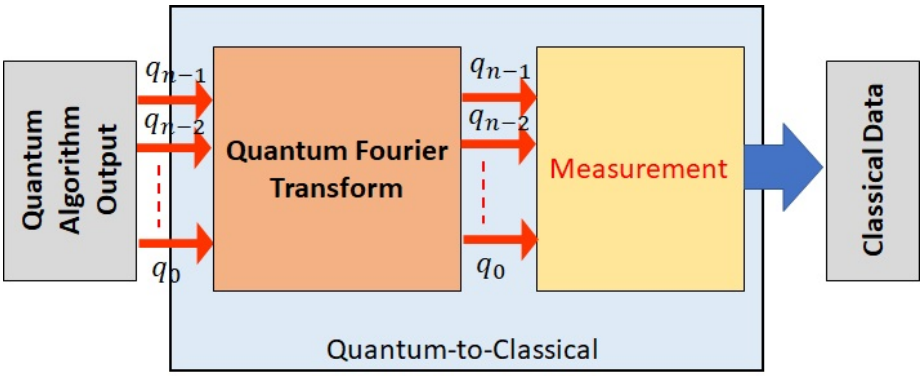


Fig. 28: Methodology overview for QFT-based quantum-to-classical data decoding

In this methodology, the output of a quantum algorithm/circuit, represented by n qubits, is passed through a QFT circuit. The quantum circuit for n-qubit QFT is shown in Fig. 4. The QFT changes the basis of the qubits to the Fourier bases, and the amplitudes of the basis states represent data in the frequency domain. The QFT output is passed to a measurement unit that repeats the circuit execution for a specified number of iterations, samples the output for every iteration, and produces a probability distribution. From the probability distribution, the basis states with the highest probabilities correspond to the frequency components present in the Fourier transformed data. The QFT-based method for Q2C data decoding is particularly interesting for applications involving image or audio processing, where properties of the data such as frequency content and/or bandwidth are useful for analyzing the output. The drawback of this method is that it does not decode the actual data encoded in the quantum state, but only reveals a collective property or feature of the data.

5.3 Quantum-to-Classical Decoding Using Quantum Haar Transform

An important feature of the Quantum Haar Transform (QHT) is that it preserves the spatial and temporal locality of data. In addition, QHT is also decomposable for multiple levels. These features make QHT an effective tool for dimension reduction, which is the process of reducing the number of features of a data set while retaining some form of spatial and/or temporal variation [49]. We propose a methodology for Q2C data decoding that incorporates use of the QHT algorithm and dimension reduction. By applying multi-level decomposable QHT, data represented by n qubits can be transformed to data represented by a lower number of qubits $k = (n - l \cdot d)$, where $k < n$, l is the number of decomposition levels, and d is the dimensionality of the data. The objective of performing dimension reduction by QHT is to use less qubits to represent the data and therefore reduce the time taken during measurement while maintaining higher fidelity. Fig. 29 shows the proposed methodology for QHT-based Q2C data decoding.

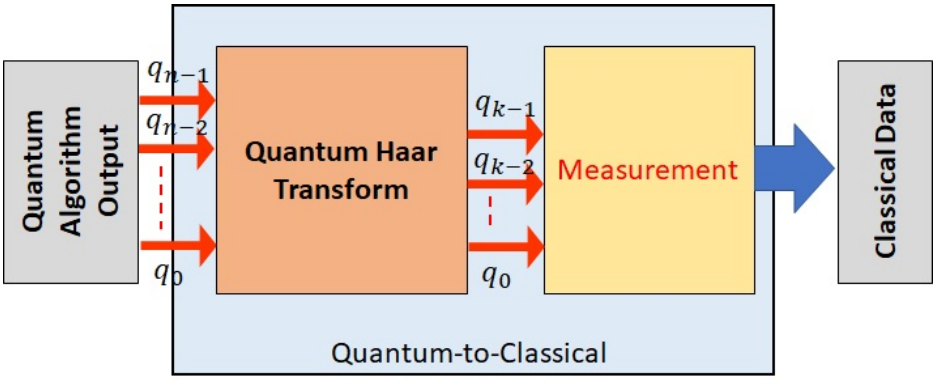


Fig. 29: Methodology overview for QHT-based quantum-to-classical data decoding

In our proposed methodology, QHT is applied to the output of a quantum algorithm, represented by n qubits. The QHT algorithm splits the data into bands with low and high frequencies. For example, a one-level two-dimensional (2D) QHT splits the data into four frequency quadrants: low-low (LL), low-high (LH), high-low (HL), and high-high (HH). The spectrum density of the LL frequency components is higher compared to the others, and contains

the most relevant and useful information that approximates the original data. At the output of the QHT circuit, the k qubits representing the low frequency bands are measured. It should be noted that dimension reduction reduces features of the data, so the amplitudes decoded from the k qubits will not be the exact amplitudes represented by the original n qubits. However, the data will retain its spatial and/or temporal locality and will have resemblance in structure with the original data. The proposed QHT-based method for Q2C data decoding will be useful in applications such as quantum image processing for efficiently visualizing transformed images.

Chapter 6: Proposed Use Cases

In this chapter, we propose three use cases of quantum algorithms such as Quantum Wavelet (Haar) Transform and Quantum Grover's search. Specifically, we propose *dimension reduction* using multi-level multi-dimensional Quantum Haar Transform (QHT) and present the corresponding depth optimized QHT circuits. We also propose dynamic *multi-pattern search* using Quantum Grover's Search and present the corresponding methodology and quantum circuits. Finally a novel quantum application is presented: efficient Quantum Pattern Recognition based on dimension reduction techniques, using both Quantum Haar Transform and Quantum Grover's Search. These use cases are evaluated using the proposed emulation framework and all corresponding hardware architectures for emulation are presented.

6.1 Dimension Reduction using Quantum Wavelet (Haar) Transform

Dimension reduction is a process of reducing the number of features of a data set while retaining some form of spatial or temporal variation from the original data set [53]. The classical wavelet transform (WT) has been shown to achieve dimension reduction efficiently [53] and can be used in various applications that use hyperspectral data, for example: remote sensing hyperspectral imagery, mineralogy, surveillance, etc. The WT uses a set of non-sinusoidal functions, usually called mother wavelets, that are both spatially and temporally localized [34]. This results in a very important feature unique to WT which is preservation of spatial locality of data. In other words, WT gives information about both time and frequency of input data. Depending on the type of data and the application in which this data is being used, multi-level and multi-dimensional WT (e.g., 1D wavelet transform (1D-WT) and 2D wavelet transform (2D-WT)) can be used for dimension reduction. For example, while the data in remote sensing hyperspectral imagery is in the form of large 3D data cubes, 1D-WT was previously proposed [53] for efficient

dimensionality reduction of such data cubes. In the experimental work in [33], five levels of wavelet decomposition were used on images of size 217×512 pixels by 192 bands to achieve $\times 32$ reduction in data volume.

In current and future large-scale applications, the volume of data can be overwhelming. For example, hyperspectral image cubes are typically hundreds of pixels in width and height [53], with 220-240 frequency bands [33]. Hence, it is necessary to investigate and apply newer paradigms of information processing and storage for supporting future applications at full bandwidths. In quantum information processing, exponentially greater amount of information can be held in the state of quantum system compared to a classical binary system. Thus, we propose using quantum information processing techniques such as Quantum Wavelet (Haar) Transform (QHT) for the processing of high volumes of data in large-scale applications. In the next sections, we elaborate our methodology in which we propose multi-level, multi-dimensional QHT to achieve dimension reduction. We propose depth-optimized quantum circuits for dimension reduction using QHT and present the corresponding emulation hardware architectures.

6.1.1 Methodology Overview

Our proposed methodology for dimension reduction using QHT is shown in Fig. 30. Input image data first undergoes a d-dimensional QHT, e.g., one-dimensional QHT (1D-QHT) or two-dimensional QHT (2D-QHT) operation. The d-dimensional QHT operations can have multiple decomposition levels and the input image is separated into a number of low frequency and high frequency replications, depending on the number of decomposition levels. The lowest frequency image replication retains the principal components of the input data without significant data loss. More importantly, the mirror images have reduced dimensionality and thus can be used for

reducing pre-processing overhead or communication bandwidth congestion. Multi-level multi-dimensional inverse quantum Haar transform (e.g., 1D-IQHT or 2D-IQHT) is then applied to reconstruct the original data.

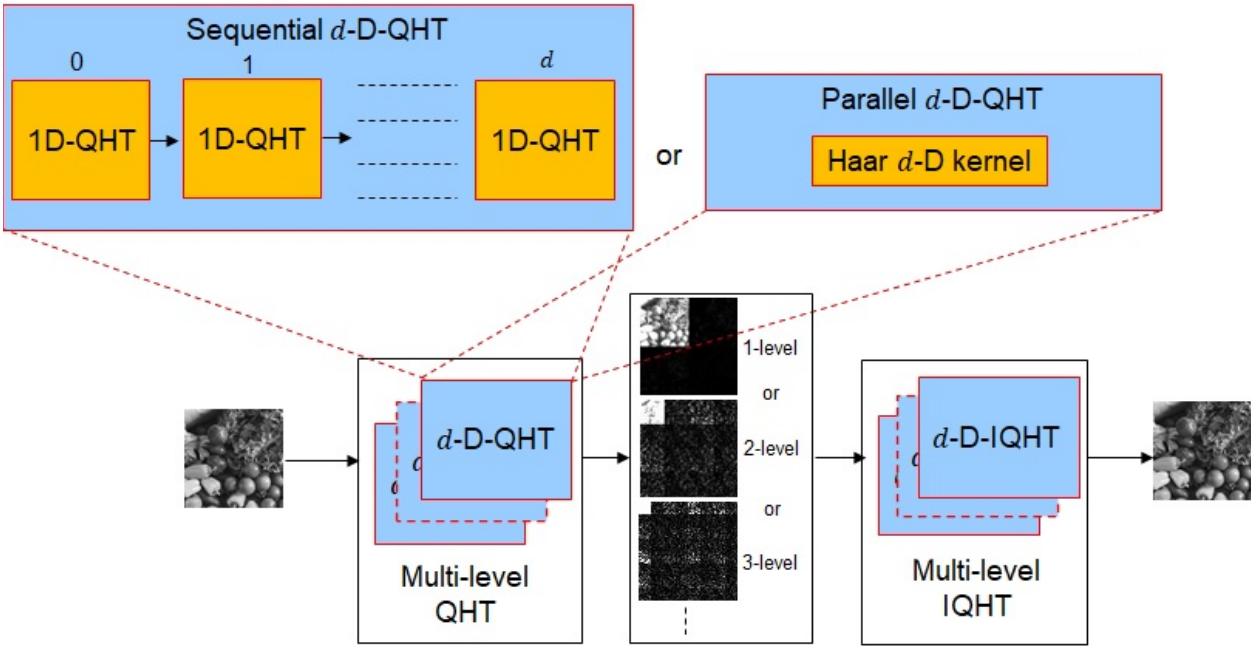


Fig. 30: Dimension reduction using multi-level, multi-dimensional QHT and IQHT.

As shown in Fig. 30, we propose performing d-dimensional QHT and IQHT operations in two ways: (1) Sequential QHT, i.e., by cascading 1D operations and multiple 1D permutation sets, and (2) Parallel QHT, i.e., applying a single d-dimensional Haar kernel. We also show that Sequential and Parallel QHT/IQHT circuit variants are decomposable, to perform multi-level-decomposable QHT/IQHT, see Fig. 30. For example, using this methodology, a $64K \times 64K$ image can be reduced to a smaller resolution of 32×32 using a 32-qubit, 12-level QWT decomposition. The data (pixels) are encoded as the coefficients of N basis states of a quantum state, where $N = 2^n$ and n is the number of qubits, i.e., 32. The quantum circuits for performing Sequential/Parallel QHT and their packet/pyramid-decomposable forms are presented in the next sections, as well as our proposed optimizations that significantly reduce the circuit depths. The corresponding

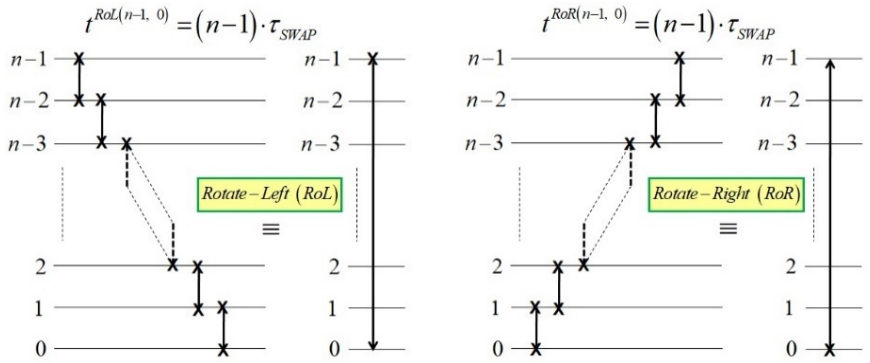
algorithms and hardware architectures for emulation of multi-dimensional, multi-level QHT are also presented.

6.1.2 Optimized Quantum Circuits

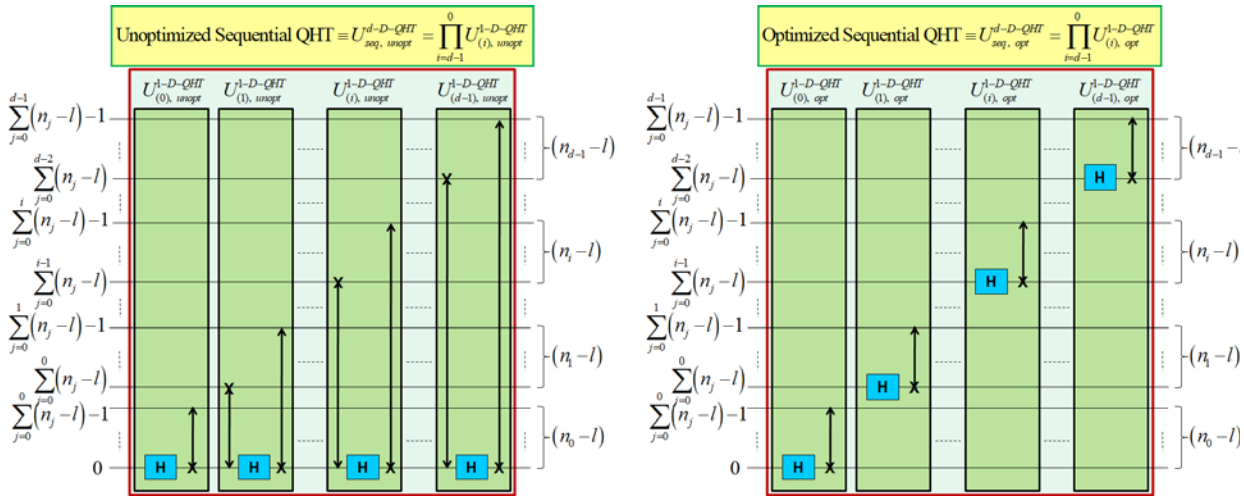
We denote a general d-dimensional QHT operation as $U^{d-D-QHT}$. The $U^{d-D-QHT}$ operation consists of three parts: (1) input permutations applied to the input state vector, (2) Haar-transform operations, and (3) output permutations applied to produce the output state vector. These operations are described for 1D, 2D, and 3D-QHT in Algorithms 1, 2, and 3 respectively in the Appendix. In the quantum domain, the Haar-transform operations are performed using *d H* gates. The input/output permutations are performed using perfect-shuffle-permutation (PSP) operations. PSPs are fundamental in classical signal and image processing [54]. Quantum PSPs can be described directly in terms of their effect on the ordering of qubits [35] [55] [56]. We present two quantum PSP operations that will be used in building our proposed quantum circuits, i.e., Rotate-Left (RoL) and Rotate-Right (RoR), see (32) and (33) respectively. RoL($n-1$, 0) and RoR($n-1, 0$) operations are essentially circular (left/right) shifts of qubits. RoL/RoR can be implemented with networks of SWAP gates, see Fig. 31(a). The number of levels of SWAP gates required for RoL($n-1, 0$)/RoR($n-1, 0$) is simply $n-1$. The gate symbols we have used for RoL/RoR in our proposed circuits are shown in Fig. 31(a).

$$RoL(n-1,0)\colon |q_{n-1}q_{n-2}\dots q_1q_0\rangle \rightarrow |q_{n-2}\dots q_1q_0q_{n-1}\rangle \tag{32}$$

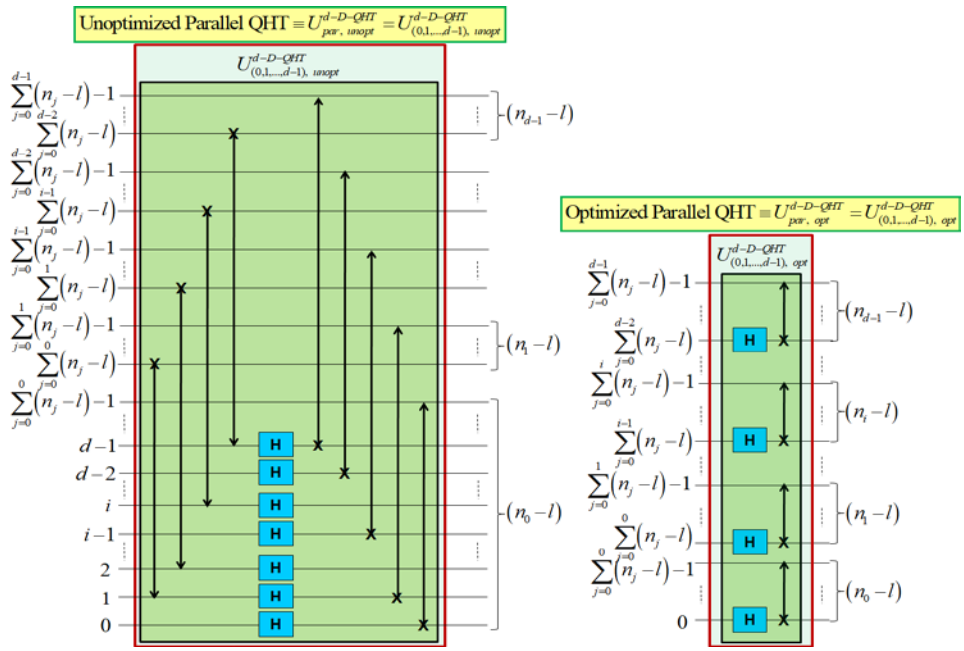
$$RoR(n-1,0)\colon |q_{n-1}q_{n-2}\dots q_1q_0\rangle \rightarrow |q_0q_{n-1}q_{n-2}\dots q_1\rangle \tag{33}$$



(a) Quantum circuits for Perfect Shuffle Permutations.



(b) Optimizations for Sequential QHT.



(c) Optimizations for Parallel QHT

Fig. 31: Quantum circuits for Sequential and Parallel QHT.

We present two generalized circuit variants that perform the operation $U^{d-D-QHT}$: Sequential (d-stage) d-dimensional QHT, and Parallel (1-stage) d-dimensional QHT. We also present unoptimized and optimized circuits for each variant. The $U^{d-D-QHT}$ operation can be performed by any of the unoptimized and/or optimized circuit variants presented in Figs. 31(b) and 31(c) for which expressions for time-delay are also derived. We also show how the Sequential and Parallel circuit variants are decomposable in packet and pyramidal forms, see Figs. 32(a) and 32(b). The notations that have been used in the time-delay expressions are defined as:

$N \equiv$ Number of data samples

$d \equiv$ Number of data dimensions

$l \equiv$ Number of decomposition levels

$l_{max} \equiv$ Maximum number of decomposition levels

$N_i \equiv$ Number of data samples in dimension i

$n_i = \lceil \log_2 N_i \rceil \equiv$ Total number of qubits representing dimension i

$n = \sum_{i=0}^{d-1} n_i \equiv$ Total number of qubits

$n_{max} = \max_{0 \le i < d-1} (n_i) =$ Maximum number of qubits across all d dimensions

$n_{min} \equiv \min_{0 \le i < d-1} (n_i) =$ Minimum number of qubits across all d dimensions

$l_{lossless} = n_{min} =$ Maximum number of levels for lossless decomposition

$l_{max}^{pkt} = \left\lfloor \frac{n}{d} \right\rfloor \equiv$ Maximum number of levels for packet decomposition (34)

$l_{max}^{pyr} = \left\lfloor \min\left(\frac{n}{d}, 1 + \frac{n - n_0}{d-1}\right) \right\rfloor \equiv$ Maximum number of levels for pyramidal decomposition

$\tau_{SWAP} \equiv$ Time delay of the SWAP gate

$\tau_H \equiv$ Time delay of the Hadamard gate

$t_{total} \equiv$ Total time delay

$U^{d-D-QHT} \equiv$ Generic d-dimensional QHT operation

$U_{pkt,l}^{d-D-QHT} \equiv d$-dimensional QHT at l level of packet decomposition

$U_{pyr,l}^{d-D-QHT} \equiv d$-dimensional QHT at l level of pyramidal decomposition

$U_{packet}^{d-D-QHT} \equiv$ Overall d-dimensional QHT of packet decomposition

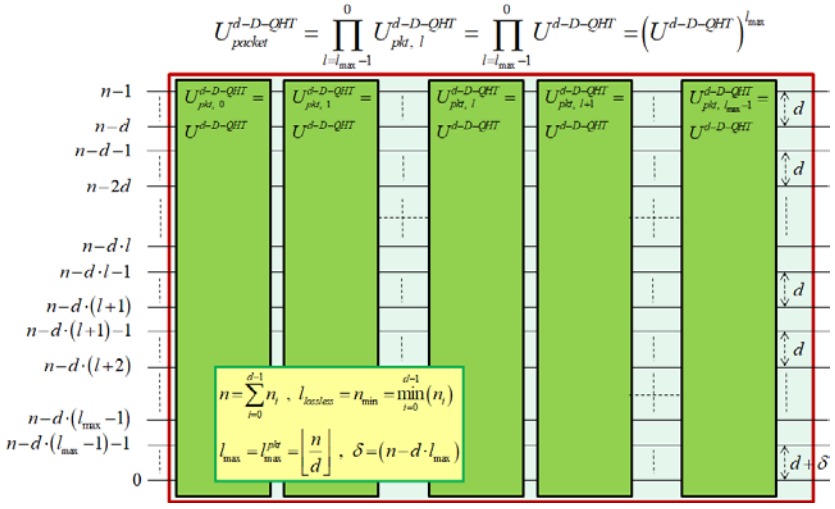
$U_{pyramidal}^{d-D-QHT} \equiv$ Overall d-dimensional QHT of pyramidal decomposition

Packet and Pyramidal decompositions

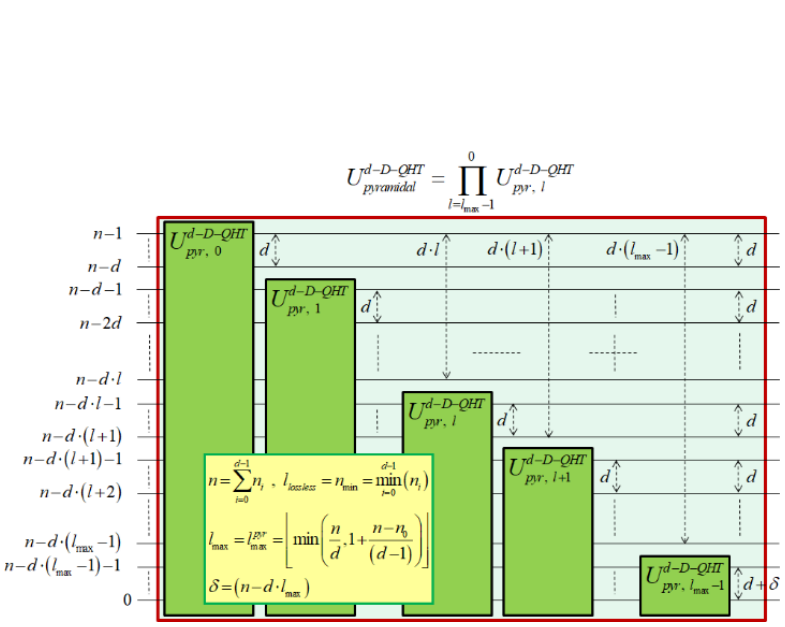
We show that d-dimensional QHT is packet/pyramidal decomposable for l levels. In packet decomposition, see Fig. 32(a), $U^{d-D-QHT}$ is repeatedly applied for every level on all the data (qubits) and all data qubits are required throughput the process. The maximum number of levels for packet decomposition, l_{max}^{pkt} see (34), depends on the total number of qubits n and the number of data dimensions d. However, for lossless decomposition, i.e., no data dimensions are lost during decomposition, the maximum number of levels is equal to the minimum number of qubits n_{min} across all d dimensions.

In pyramidal decomposition, see Fig. 32(b), for each level of decomposition, the d-dimensional QHT operates on fewer data qubits. Specifically, d qubits (1 qubit per each dimension) are discarded after every decomposition level, see 31(b). Similar to packet decomposition, the maximum number of levels for lossless pyramidal decomposition is also n_{min}. The expression for the maximum number of possible pyramidal decomposition levels, l_{max}^{pyr}, is given in (34). Pyramidal decomposition has certain advantages over packet in that the size and depth of the QHT circuit is reduced after every decomposition level. However, one drawback is that more inter-level permutations are required, see Fig. 32(c). The time-delay for inter-level pyramidal permutations could be derived using Fig. 32(c) and is given in (35).

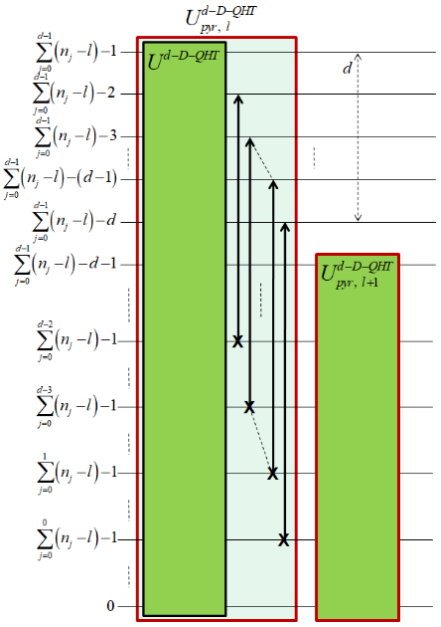
$$t^{pyr-perm} = \left(n - n_0 - (d-1)\frac{l}{2}\right)(l-1)\cdot\tau_{SWAP} \tag{35}$$



(a) Packet decomposition



(b) Pyramidal decomposition



(c) Inter-level pyramidal permutations

Fig. 32: Multi-level decomposition of *d*-dimensional QHT.

Sequential and Parallel QHT

For Sequential QHT, *d*-dimensional QHT can be performed by cascading d 1D-QHT transforms, see Fig. 31(b). Each 1D-QHT, U^{1D-QHT}, consists of RoL, *H* gates, and RoR gates, see Fig. 31(a). The 1D-QHT is consecutively repeated for every dimension indexed from 0 to $d-1$.

The (unoptimized) sequential QHT circuit could be decomposed in multi-level packet or pyramidal forms. The total time-delays for packet and pyramidal decompositions with l levels is provided in (36).

$$t_{total}^{seq,unopt.,pkt} = \left(\left((2d-1)n - 2\sum_{i=0}^{d-1} i \cdot n_i - d\right) \cdot \tau_{SWAP} + d \cdot \tau_H\right) \cdot l$$
$$t_{total}^{seq,unopt.,pyr} = t_{total}^{seq,unopt.,pkt} + t^{pyr-perm} - \frac{d^2 \cdot l(l-1)}{2} \cdot \tau_{SWAP} \quad (36)$$

The d-stage sequential QHT circuit can be optimized as shown in Fig. 31(b). For each 1D-QHT operation, the RoL operation is eliminated, and the kernel (H gate) is shifted up to its corresponding dimension i, where $i = 0,1,I.,d-1$. This reduces the consecutive RoR operation (less depth) and thus reduces the overall circuit depth. The total time-delays for the optimized sequential packet and pyramidal decomposable d-dimensional QHT circuits are given in (37).

$$t_{total}^{seq,opt.,pkt} = \left((n-d) \cdot \tau_{SWAP} + d \cdot \tau_H\right) \cdot l$$
$$t_{total}^{seq,opt.,pyr} = t_{total}^{seq,opt.,pkt} + t^{pyr-perm} - \frac{d \cdot l(l-1)}{2} \cdot \tau_{SWAP} \quad (37)$$

In Parallel QHT the Haar operation (H gates) is applied in parallel (1-stage) instead of in sequence on each of the d dimensions, see Fig. 31(c). The RoR and RoL operations are grouped into sets of preceding and proceeding permutations respectively, see Fig. 31(c). This circuit variant can be used in packet or pyramidal decomposition. The total time-delays of the packet and pyramidal decomposable circuits are given in (38).

$$t_{total}^{par,unopt.,pkt} = \left(\left(2n - n_{d-1} - (2d-1)\right) \cdot \tau_{SWAP} + \tau_H\right) \cdot l$$
$$t_{total}^{par,unopt.,pyr} = t_{total}^{par,unopt.,pkt} + t^{pyr-perm} - \frac{(2d-1) \cdot l(l-1)}{2} \cdot \tau_{SWAP} \quad (38)$$

The parallel (1-stage) QHT is optimized further by positioning the H gates separated by n_i qubits, where $i = 0,1,I.,d-1$, see Fig. 31(c). Due to this shift, no preceding permutations (RoL gates) are required. The proceeding RoR operations are also reduced in depth and can be applied

in parallel as they are independent of each other. The total time-delay for the optimized parallel decomposable (packet and pyramidal) *d*-dimensional QHT circuits is given in (39).

$$t_{total}^{par,opt.,pkt} = \left((n_{max} - 1) \cdot \tau_{SWAP} + \tau_H\right) \cdot l$$
$$t_{total}^{par,opt.,pyr} = t_{total}^{par,opt.,pkt} + t^{pyr-perm} - \frac{l(l-1)}{2} \cdot \tau_{SWAP} \quad (39)$$

<u>*Packet vs Pyramidal QHT*</u>

We define the general time-delays for packet and pyramidal decomposable circuits as t^{pkt} and t^{pyr}, irrespective of the type of QHT circuit or the optimization (serial/parallel or optimized/unoptimized). Therefore, a general time expression can be derived as (40) from the equations given in (30)-(33).

$$t^{pyr} = t^{pkt} + t^{pyr-perm} - \Delta t \quad (40)$$

where,

$$\Delta t = f(d) \cdot \frac{l(l-1)}{2}, \qquad \text{and } f(d) = \begin{cases} d^2 & \rightarrow seq.\ unopt \\ d & \rightarrow seq.opt \\ (2d-1) & \rightarrow par.unopt \\ 1 & \rightarrow par.opt \end{cases}$$

For pyramidal to be faster than packet, $t^{pyr} - t^{pkt} \leq 0$ should be true, i.e., $t^{pyr-perm} - \Delta t \leq 0$. Using the expression for $t^{pyr-perm}$ and Δt, we can derive an expression for the minimum number of decomposition levels, l_{min}^{pyr} required for pyramidal to be faster than packet, given in (41).

$$l_{min}^{pyr} = \frac{2(n - n_0)}{f(d) + d - 1} \quad (41)$$

6.1.3 Hardware Architectures for Emulating Quantum Haar Transform

We propose kernel-based emulation algorithms for multi-level 1D-QHT, 2D-QHT, and 3D-QHT and they are presented in the Appendix as Algorithms 1, 2, and 3 respectively. The algorithms perform multi-level decompositions of d-D-QHT operations based on a d-dimensional Haar

wavelet kernel. The kernel functionality can be represented by a set of operations applied to 2^d data points, and is preceded and followed by perfect shuffle permutation operations [35] on the input and output data points. The permutation operations are performed by means of index calculations and scheduling. Algorithm 1 (in the appendix) performs multi-level decompositions of 1D-QHT operations based on a multi-dimensional Haar wavelet kernel. The kernel functionality is described by a set of operations applied to input states/pixels, and is preceded and followed by 1D perfect shuffle permutation operations [35] on the input and output states/pixels. The permutation operations are performed by means of index calculations and scheduling. Algorithm 2 (in the appendix) performs 2D-QHT on a set of *N* input pixels *X* and produces an output pixel set *Y*. The first stage in the algorithm is to perform input permutations on the input pixels, followed by 2D Haar kernel operations on $2^d = 4$ neighboring pixels every cycle, and then finally output permutations are performed producing the output set of pixels. The 3D-QHT operation in Algorithm 3 (in the appendix) is very similar, performing a 3D Haar kernel on $2^d = 8$ neighboring pixels every cycle, preceded and followed by input and output permutations on the pixels.

The hardware architectures equivalent to Algorithm 1 (in the appendix) for emulation of 1D-QHT are shown in Figs. 33(a), (b), and (c). The first stage in Algorithm 1 is the input permutation P_{in}^{1D} . The permutation can be emulated by gate models of RoR and RoL operations but that incurs high resource utilization in the corresponding hardware architecture. For this reason, classical models are used that involve simple index scheduling and the corresponding emulation architecture is shown in Fig. 33(a). The input is a vector of quantum state coefficients which are written to a memory array in the index order 0 to $N - 1$. Two coefficient values are then read out each clock cycle, with the scheduler generating the read indices $i_{X_{00}}$ and $i_{X_{10}}$ according to the input permutation, see Algorithm 1 (in the appendix). The scheduler calculates a row index i_{row} and a

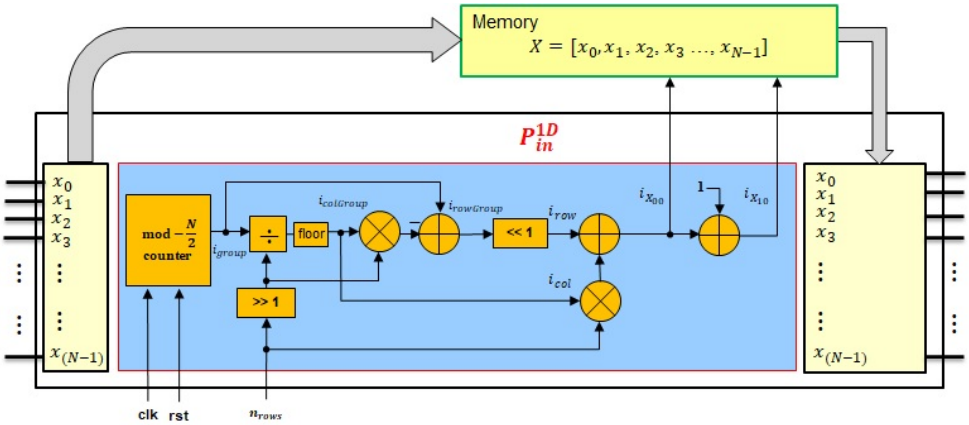
column index i_{col}, to determine the output indices. These are used to write the output state into an output buffer. Optimizations such as replacing multiplications and divisions by powers of two with logical shifts are done for more time and resource efficient hardware emulation. A floor operation module is also implemented for the scheduler.

As shown in Fig. 31, the Haar transformations, are modeled using Hadamard gates. The Hadamard operation reduces to kernel operations on a set of 2^d coefficients and the kernel operation is iterated over all data points or states. The emulation architecture for the 1D Haar kernel is shown in Fig. 33(b). The design takes in 2 input coefficients, applies the kernel operations which involve addition and division, and outputs four coefficients per clock cycle. Conventional operator sharing techniques and logical shifts are applied to optimize for speed and area.

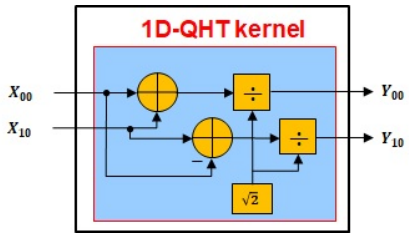
The final stage in Algorithm 1 (in the appendix) is the output permutation, P_{out}^{1D}. The corresponding emulation architecture is shown in Fig. 33(c) and works similarly to the input permutation scheduler. The input vector of coefficients is written to a memory array, 2 values per clock cycle, with the scheduler generating the write indices $i_{Y_{00}}$ and $i_{Y_{10}}$ according to the output permutation described in Algorithm 1 (in the appendix). The permuted coefficients are then read out from memory 2 values per clock cycle.

The architectures equivalent to Algorithm 2 (in the appendix) for emulation of 2D-QHT are shown in Figs 33(d), (e), and (f). The operations of the architectures are similar to those of 1D-QHT. Four coefficient values are read each clock cycle, with the input scheduler generating the read indices $i_{X_{00}}$, $i_{X_{01}}$, $i_{X_{11}}$, and $i_{X_{10}}$ see Fig. 33(d) and Algorithm 2 (in the appendix). The input scheduler calculates a row index i_{row} and a column index i_{col}, to determine the output indices which are used to write the output state into an output buffer. 2D Haar kernel operation, see Fig.

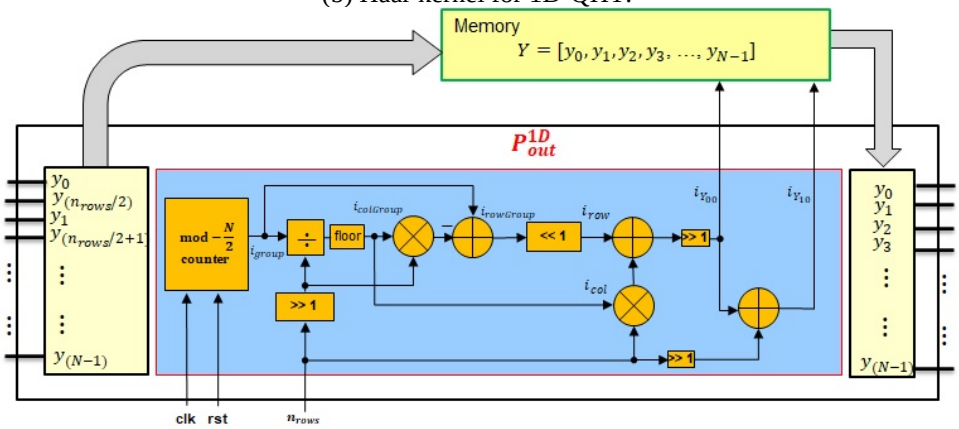
33(e), takes in 4 input coefficients, applies the kernel operations which involve addition and division, and outputs four coefficients per clock cycle. The 2D output permutations scheduler, see Fig. 33(f), works similar to that for 1D, and operates on 4 coefficient values per clock cycle.



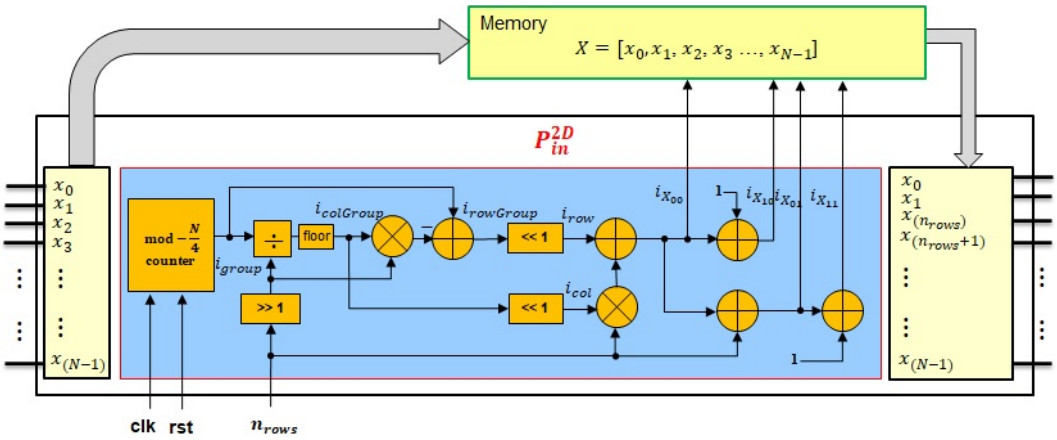
(a) Input permutations scheduler for 1D-QHT.



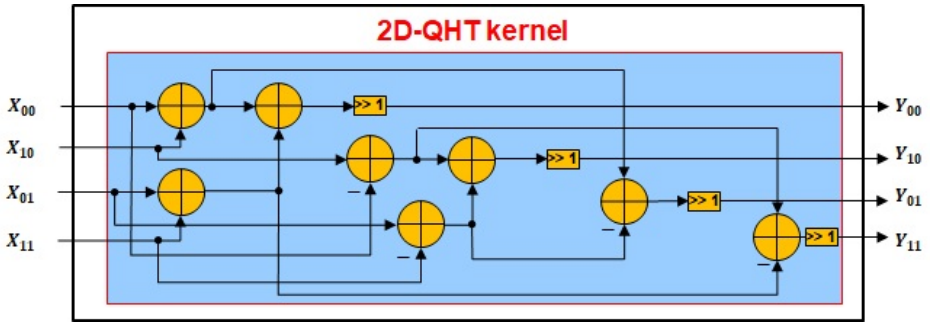
(b) Haar kernel for 1D-QHT.



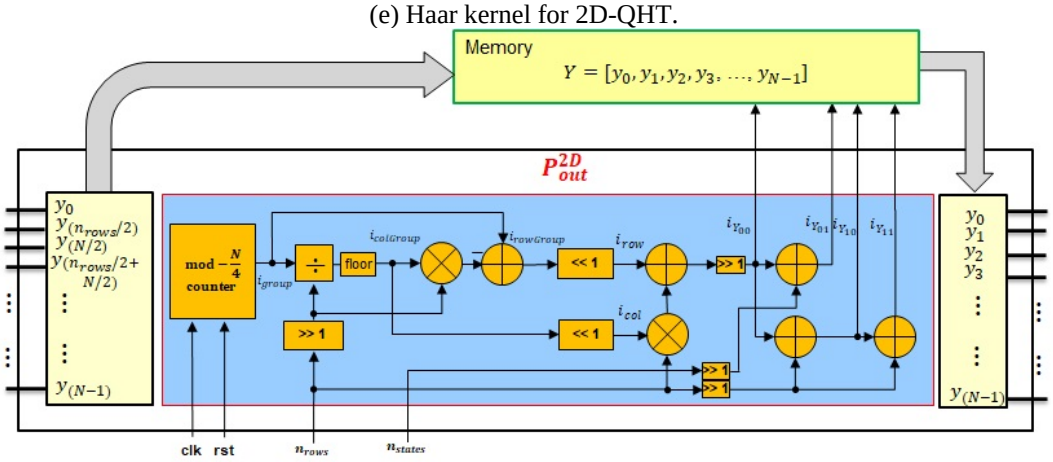
(c) Output permutations scheduler for 1D-QHT.



(d) Input permutations scheduler for 2D-QHT.



(e) Haar kernel for 2D-QHT.



(f) Output permutations scheduler for 2D-QHT.

Fig. 33: Hardware architectures for emulation of 1D-QHT and 2D-QHT.

6.2 Dynamic Multi-Pattern Search using Quantum Grover's Search

Generally, Grover's algorithm consists of two main steps, the *oracle* (also called phase inversion) and *diffusion* (also called inversion about the mean) [16]. For both traditional single-

pattern as well as multi-pattern Grover's algorithm, the oracle circuit has to be statically set up before computations for every input search pattern, which is inconvenient for fast and dynamic search. Therefore, we present a modified Grover's algorithm capable of fast, dynamic searches with multiple patterns.

6.2.1 Proposed Methodology

Our proposed methodology for dynamic multi-pattern Grover's search is shown in Fig. 34 and consists of two modifications compared to the conventional single-pattern Grover's search. Our first modification adds a dynamic *oracle* circuit U_{oracle} that locates items at the first $N_{patterns}$ indices of the search list. This is followed by a conventional Grover's *diffusion* circuit $U_{diffusion}$ that increases the probabilities of locating the pattern(s). The *oracle* and *diffusion* quantum circuits are repeated for *m* iterations to produce an output quantum state, where *m* is the optimal number of iterations given by (10). Our second modification adds a permutation $U_{permute}$ of the basis of the quantum state, which is critical for successfully locating the target pattern(s).

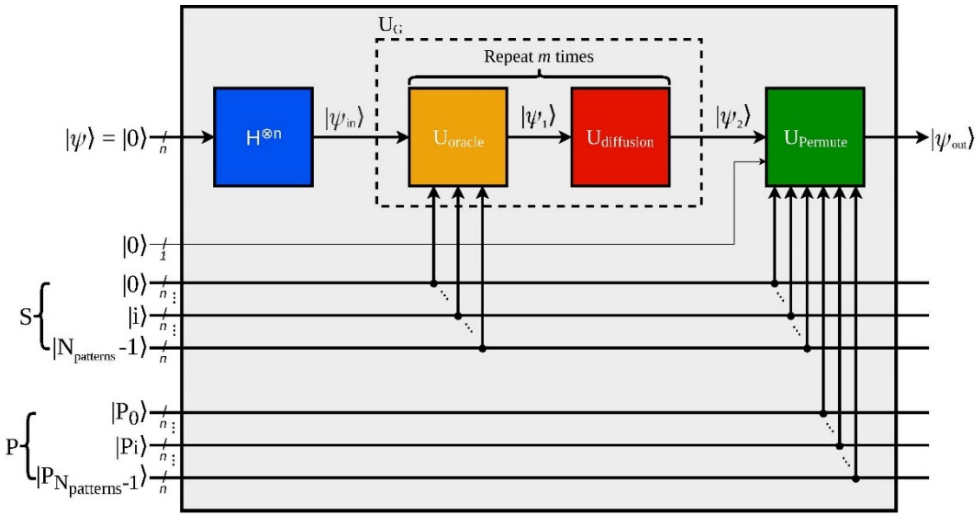


Fig. 34: Proposed/modified quantum circuit for multi-pattern Quantum Grover's Search.

Separate from the *oracle* and *diffusion* stages, the permutation can be performed using either classical or quantum gates. The proposed permutation circuit uses ancilla bits/qubits, in which the target pattern(s) are encoded, to assign the probability coefficients to the correct basis states.

6.2.2 Implementation

The proposed quantum circuit for multi-pattern Grover's algorithm, see Fig. 34, has four sets of inputs: 1) a set of *n* compute qubits $|\psi\rangle$ which are all initialized to the ground state $|0\rangle$, i.e., $|\psi\rangle = |0\rangle^{\otimes n}$, 2) a single flag ancilla qubit initially set to the ground state $|0\rangle$, 3) a set of $N_{patterns}$ entries of statically initialized *n* ancilla qubits, i.e., $S = \{|0\rangle, |1\rangle, \dots, |N_{patterns-1}\rangle\}$, and 4) a set of $N_{patterns}$ entries of dynamically changing *n* ancilla qubits that represent the input search patterns in $|\psi_{in}\rangle$, i.e., $P = \{|P_o\rangle, |P_o\rangle, \dots, |P_{N_{patterns-1}}\rangle\}$. For example, if $P = \{|3\rangle, |7\rangle\}$ then the basis states to be located and amplified in the initial quantum state are $|3\rangle$ and $|7\rangle$ and $N_{patterns} = 2$.

The first step, as shown in Fig. 34, is to initialize the input state of the qubits, $|0\rangle$, to a uniform superposition state, $|\psi_{in}\rangle$. This is accomplished by applying an *H* gate to each one of the *n* qubits, i.e., $H^{\otimes n} = H \otimes H \otimes \dots \otimes H$, which sets equal amplitudes to all states in $|\psi_{in}\rangle$ ($|0\rangle, |1\rangle, \dots, |N-1\rangle$). The input qubit state vector $|\psi_{in}\rangle$ after the application of the *H* gate is expressed mathematically in (42) as:

$$|\psi_{in}\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle \tag{42}$$

Once the input qubits are in a uniform superposition state, a modified dynamic oracle operation U_{oracle} and an unmodified diffusion operator $U_{diffusion}$, are applied *m* consecutive times, amplifying the first $N_{patterns}$ states. As only the first $N_{patterns}$ states will be amplified, a permutation step, $U_{permute}$, is performed to assign the high amplitudes to the target states based

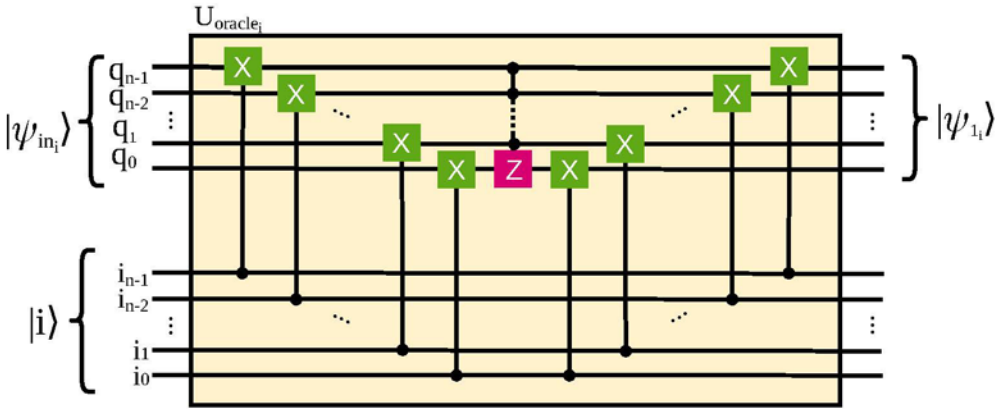
on the input patterns *P*. These iterations produce the final output state, $|\psi_{out}\rangle$. This process can be represented using a single unitary matrix $U_G = \left(U_{diffusion} \cdot U_{oracle}\right)^m$, and m is the optimal number of iterations given by (10). The probability, $P_{success}$, of successfully finding a desired pattern in the final output state $|\psi_{out}\rangle$ is expressed in (43) [37], where *N* is the size of the unsorted list of elements and $N_{patterns}$ equals the number of solutions/patterns being searched for such that $N_{patterns} \leq N$.

$$P_{success} = \left(\sin\left((2m+1) \times \theta\right)\right)^2$$
$$\theta = \sin^{-1}\left(\sqrt{\frac{N_{patterns}}{N}}\right), \text{and } 0 < \theta \leq \frac{\pi}{2} \tag{43}$$

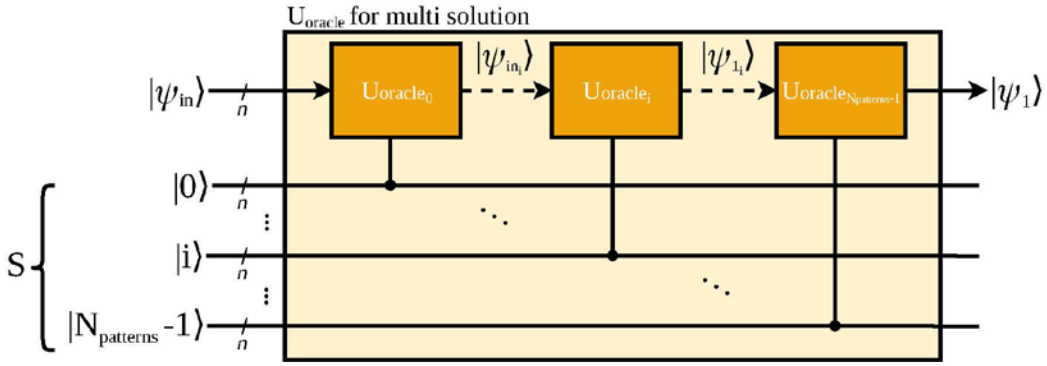
6.2.3 Modified Oracle and Diffusion Circuits

Our proposed oracle model, U_{oracle}, uses *cX* gates to dynamically modify the target pattern as seen in Fig. 34. Dynamic modification of the search pattern allows us to extend and generalize the algorithm to dynamically search for any pattern with a single quantum circuit. In the conventional Grover's algorithm, a different oracle is needed for every search pattern, requiring a new quantum circuit for each pattern. In our modified Grover's algorithm, the *cX* gates in each oracle are controlled by ancilla qubits that are set to the current pattern that is being amplified, $|i\rangle$, as seen in Fig. 35(a). To generalize the circuit further, only the first $N_{patterns}$ amplitudes are inverted. Therefore, in single-pattern search, the oracle ancilla qubits are set to $|0\rangle$ so that only the amplitude on the first state will be inverted. For multi-pattern search, we apply cascaded, incremental single-pattern oracle quantum circuits to invert the first $N_{patterns}$ amplitudes as seen in Fig. 35(b). As oracle circuits are mutually exclusive, i.e., each oracle circuit only inverts a single state and does not affect any other state, they could be sequentially cascaded in any arbitrary order, e.g., an

ascending order as shown in Fig. 35(b). The output from the oracle, $|\psi_1\rangle$, is subsequently provided to the next stage, i.e., Grover's diffusion, for amplification.



(a) Modified Grover's oracle for a single solution/pattern.



(b) Modified Grover's oracle for multiple solutions/patterns

Fig. 35: Modified oracle circuits for the proposed multi-pattern Quantum Grover's Search

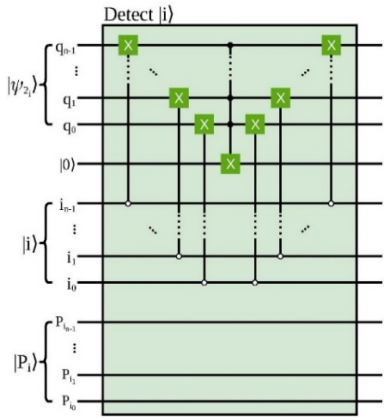
The diffusion circuit, $U_{diffusion}$, is identical to the conventional diffusion circuit used in Grover's algorithm, see Fig. 5(b). The diffusion circuit only amplifies the states negated by the oracle, such that their resultant amplitudes are greater than their values before the oracle operation was performed. Because the oracle only inverts the first $N_{patterns}$ states and the diffusion circuit only amplifies inverted states, the first $N_{patterns}$ states are amplified while the remaining state amplitudes are attenuated. The U_{oracle} and $U_{diffusion}$ stages are then iterated over m times to maximize the target amplitudes.

6.2.4 Quantum State Permutation

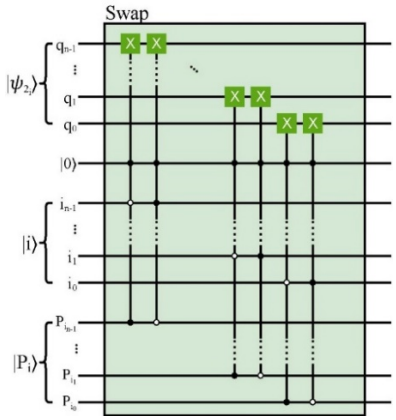
Our modified design of Grover's algorithm amplifies the first $N_{patterns}$ states, and a permutation step is added to assign the amplified amplitudes to the target basis states in the output state $|\psi_{out}\rangle$. Similar to our oracle implementation, the permutation step consists of cascaded mutually exclusive single permutation operations, as seen in Fig. 36. The individual permutation step swaps two selected states based on a static index *I* and a dynamic input pattern P_i, where $P_i \in P = \left\{P_0, P_1, \dots, P_{N_{patterns}-1}\right\}$. As each individual permutation step only swaps a single state with another state, a total of $N_{patterns}$ permutation steps are needed to permute each high state with one target basis state. Here, each permutation step, denoted $U_{permute_i}$, swaps the state in $|i\rangle$ with the state located at P_i. In other words, $U_{permute_0}$ means that state $|\psi_0\rangle$ is swapped with the first index of *P*, i.e., P_0, which can be any state in $|\psi_2\rangle$. This is then followed by the second state $|1\rangle$ with the second pattern in *P*, i.e., P_1, and so on until the last state $N_{patterns} - 1$ is swapped with the last pattern in *P*, i.e., $P_{N_{patterns}-1}$, as shown in Fig. 36(f) and described by (44).

$$U_{permute} = U_{permute_{N_{patterns}-1}} \cdot \dots \cdot U_{permute_i} \cdot \dots \cdot U_{permute_0}$$
$$U_{permute_i} = U_{permute_i}^{toggle\ flag} \cdot U_{permute_i}^{swap} \cdot U_{permute_i}^{detect|P_i\rangle} \cdot U_{permute_i}^{swap} \cdot U_{permute_i}^{detect|i\rangle} \quad (44)$$

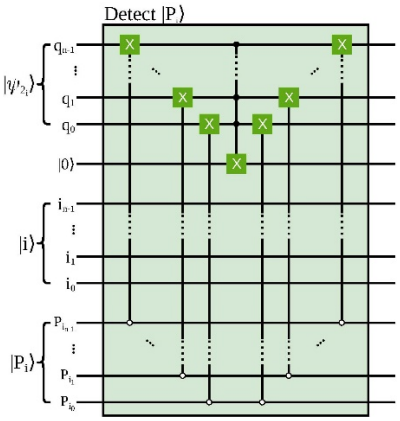
Once all $N_{patterns}$ permutations are performed, the output state $|\psi_{out}\rangle$ will have high amplitudes only in the desired states. The quantum circuit that performs the individual permutation operation consists of five consecutive steps as shown in Fig. 36(d) and described by (44).



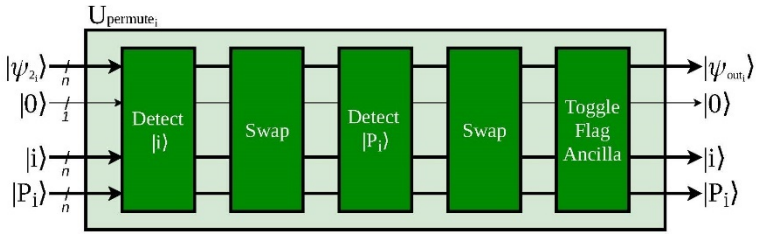
(a) Detect $|i\rangle$ and toggle flag ancilla operation $U_{permute_i}^{detect|i\rangle}$



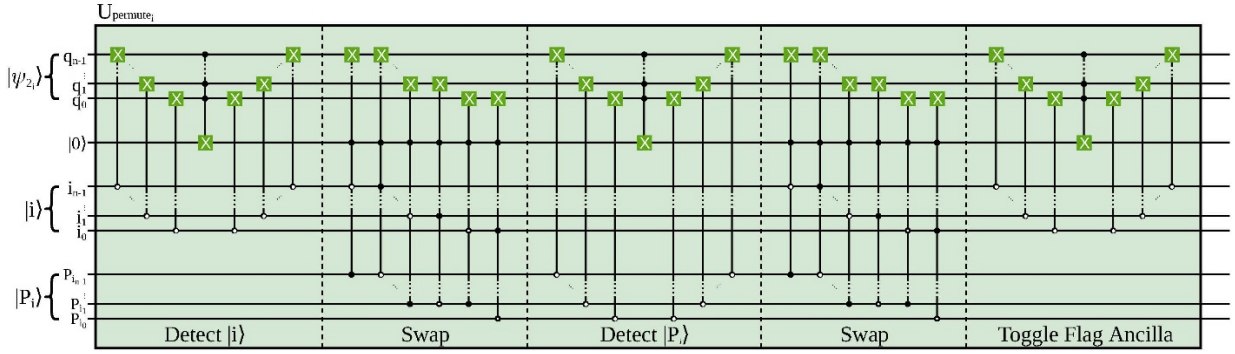
(b) Swap operation $U_{permute_i}^{swap}$



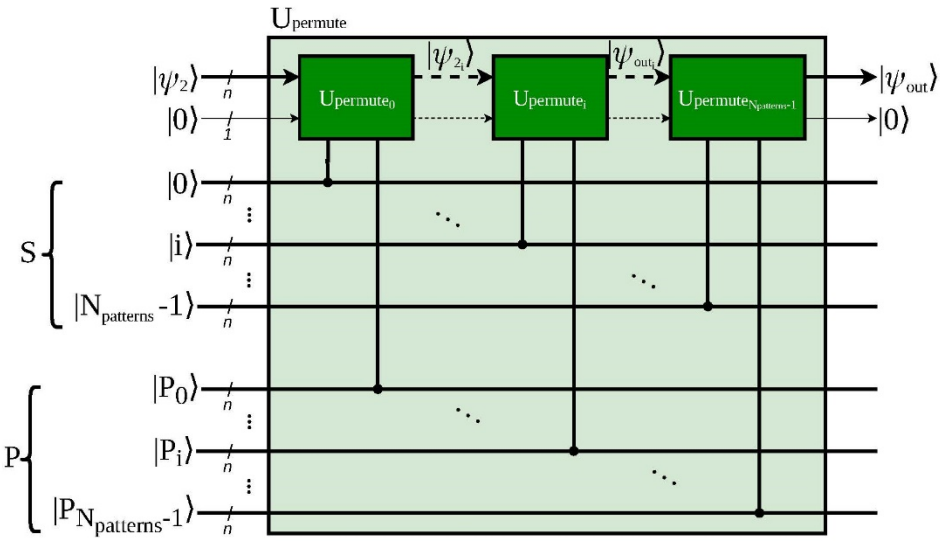
(c) Detect P_i and toggle flag ancilla operation $U_{permute_i}^{detect|P_i\rangle}$



(d) Permutation circuit $U_{permute_i}$ for a single pattern



(e) Detailed permutation circuit $U_{permute_i}$ for a single pattern



(f) Permutation circuit $U_{permute}$ for multiple patterns

Fig. 36: Permutation circuits for multi-pattern Quantum Grover's Search.

The first step, shown in Fig. 36(a), detects the index at $|i\rangle$ and toggles (sets) the flag ancilla qubit to $|1\rangle$ whenever $|\psi_{2_i}\rangle = |i\rangle$, but leaves it unchanged at $|0\rangle$ for all other states of $|\psi_{2_i}\rangle$. This singles out the desired coefficient to swap in $|\psi_{2_i}\rangle$ and allows it to be manipulated without affecting the other coefficients in $|\psi_{2_i}\rangle$. The second step, shown in Fig. 36(b), is labeled as the *swap* operation and swaps the desired flagged coefficient at $|i\rangle$ to the correct index of $|P_i\rangle$. To do this, *X* gates are applied only when the ancilla qubit is flagged to $|1\rangle$ and $|i\rangle \oplus |P_i\rangle = |1\rangle$. This will swap the coefficient value at $|i\rangle$ with that at $|P_i\rangle$ leaving the flagged coefficient at the right index, while the ancilla qubit remains flagged to $|1\rangle$. The third step, shown in Fig. 36(c), is similar to the first step where it detects the index at $|P_i\rangle$ and toggles the flag ancilla qubit. However, instead of flagging the ancilla qubit when $|\psi_{2_i}\rangle = |i\rangle$, this step toggles (resets) the flag on the ancilla qubit when $|\psi_{2_i}\rangle = |P_i\rangle$. At this point, the original flagged coefficient that was at index $|i\rangle$ in $|\psi_{2_i}\rangle$ has been swapped to index $|P_i\rangle$, and the coefficient at index $|P_i\rangle$ has yet to be swapped back. In steps four and five, we swap back the coefficient at $|P_i\rangle$ by applying an additional *swap* operation (step

four) and an additional *detect* operation (step five). Step five restores the flag ancilla qubit to its initial ground state $|0\rangle$. The circuit is shown in its entirety in Fig. 36(e). This permutation circuit allows for a more generalized and flexible design, as the oracle/diffusion circuit only needs to amplify the first $N_{patterns}$ and the permutation circuit can dynamically swap the necessary states.

6.2.5 Hardware Architectures for Emulating Quantum Grover's Search

For hardware implementation of the proposed multi-pattern quantum Grover's algorithm, our objective was to derive space-efficient emulation architectures while maintaining a high level of accuracy and throughput. High accuracy was achieved by using single-precision floating-point representations to model qubits and quantum operations. The complex coefficients describing qubits and quantum gates are represented using 64 bits, with 32 bits for the real and imaginary components respectively.

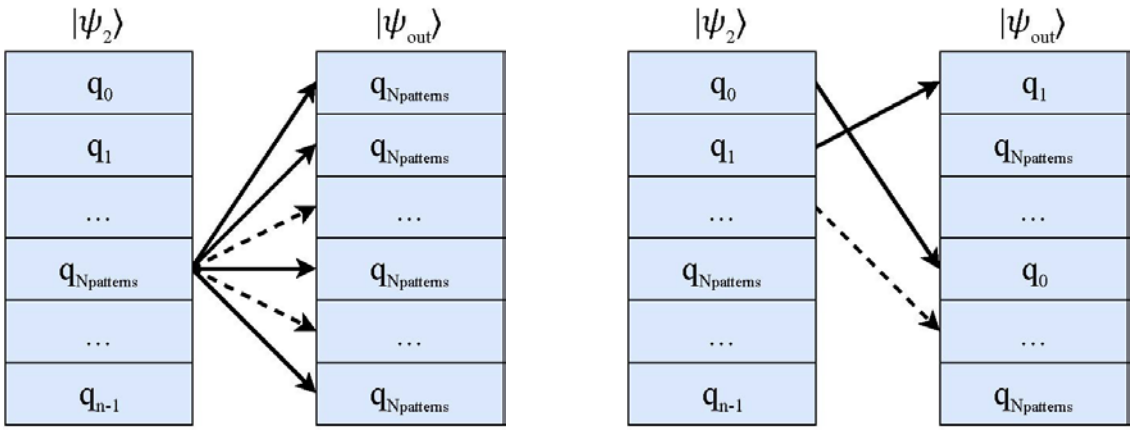
To achieve our goals of space-efficiency and high throughput, we conducted a thorough analysis of each stage of our proposed modified Grover's algorithm. The first stage of the proposed algorithm is qubit initialization and normalization, see Fig. 34. This was realized efficiently and simply on classical hardware by initializing an array of ones, and can also be achieved with the equivalent quantum circuit $H^{\otimes n}$. To implement the second stage U_G, shown in Fig. 34, we used the stream-based CMAC emulation approach, whose architecture is presented in Fig. 25. The emulator determines the output quantum state $|\psi_{out}\rangle$ given an input state $|\psi_{in}\rangle$ and the unitary operation of the quantum algorithm U_{ALG}, where $U_{ALG} = U_G$ for the case of Grover's algorithm. The inputs are streamed in and stored in buffers, the outputs are streamed out from an output buffer, and the dataflow architecture is fully pipelined.

The emulation architecture leverages well-known multiply-and-accumulate techniques. An efficient complex multiply-and-accumulate (CMAC) unit is designed to perform complex vector-

matrix and matrix-matrix multiplications. The number of CMAC instances that the emulator architecture uses can be varied from 1 to N, as a trade-off between circuit area and speed. Additionally, irrespective of the number of operations in the algorithm that is being modeled, the architecture of the CMAC remains fixed. In other words, a single CMAC will always have the same number of arithmetic units, allowing the use of the same number of CMAC unit(s) with increasing circuit size. This ensures a highly scalable and space-efficient design. This emulation model is also generalized and capable of emulating any quantum algorithm that can be reduced to a single unitary transformation, i.e., the algorithm matrix can be pre-computed and stored on a host machine and streamed into the emulator during the computation. The streaming technique accounts for algorithms such as Shor's, whose algorithm matrix changes dynamically as the algorithm inputs change. As a result of streaming, there is a communication latency overhead between the host machine and emulator, but the latency is negligible compared to computation time. The computational complexity of this emulation model is $O(N^2)$. However, pre-computing the algorithm matrix is generally challenging and can add to the complexity of the emulation, depending on the targeted quantum algorithm. In the case of some algorithms, such as Shor's and Quantum Approximation Optimization Algorithm (QAOA) [57], the pre-computation could limit the efficiency of the emulation model.

A single CMAC unit is shown in Fig. 23 and its operations are described in (20). A single CMAC unit works on the real and imaginary elements of the input state vector and algorithm matrix, performing four additions and four multiplications in total. Using this CMAC architecture, we store only the quantum state vectors and use fast input streams for the algorithm matrix U_{ALG}. This technique allows the emulation of a much higher number of qubits than existing emulator designs.

The final operation of the algorithm is $U_{permute}$, illustrated in Fig. 34. $U_{permute}$ is performed in two stages, both of which involve permuting the coefficients in the quantum state vector $|\psi_2\rangle$ to produce the output quantum state vector $|\psi_{out}\rangle$. In the first permutation stage, shown in Fig. 37(a), the output vector $|\psi_{out}\rangle$ is populated with the low coefficient value located at index $N_{patterns}$ of the $|\psi_2\rangle$ vector. In the second stage of permutation, shown in Fig. 37(b), the amplified indices (0 to $N_{patterns} - 1$) in $|\psi_2\rangle$ are driven to the target indices in $|\psi_{out}\rangle$ based on P.



(a) First stage of permutation (b) Second stage of permutation

Fig. 37: Stages of the permutation operation on a quantum state vector.

To perform these permutations, $U_{permute}$ can be emulated as a direct quantum circuit model of quantum gates as shown in Fig. 36(e) and Fig. 36(f). We propose a more space-efficient approach for emulation using classical methods like index scheduling, since quantum permutation is, in essence, the swapping of basis coefficients of the quantum state. For a hardware index scheduler, let *I* index values be defined as a function of *j* index values:

$$i = \begin{cases} 0, & if\ j \in P \\ N_{patterns}, & otherwise \end{cases}$$

The basis coefficients of the input and output states of $U_{permute}$ are stored at indices *I* and *j* as $C_i^{\psi_2}$ and $C_j^{\psi_{out}}$ respectively:

$$\psi_{out}(j) = C_j^{\psi_{out}}$$

$$\psi_2(i) = C_i^{\psi_2}$$

The quantum input and output states of $U_{permute}$, which are $|\psi_2\rangle$ and $|\psi_{out}\rangle$, respectively, are defined by the following expressions:

$$|\psi_2\rangle = \sum_{i=0}^{N-1} C_i^{\psi_2} |i\rangle$$

$$|\psi_{out}\rangle = \sum_{j=0}^{N-1} C_j^{\psi_{out}} |j\rangle$$

The permutation operation may then be described as:

$$\psi_{out}(j) = \psi_2(i) \tag{45}$$

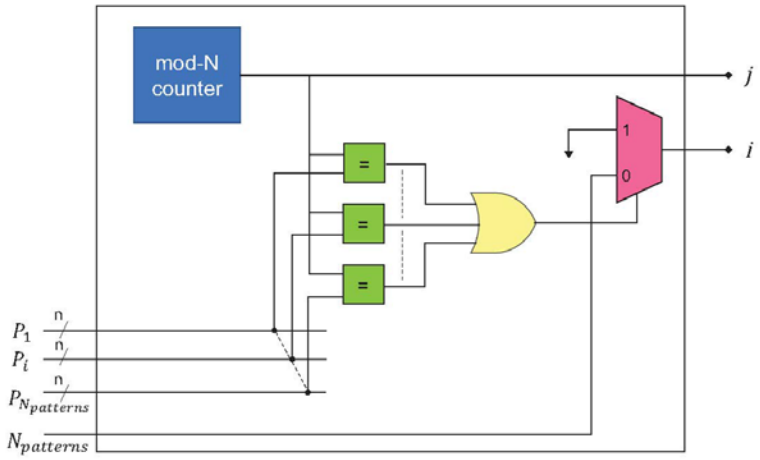


Fig. 38: Hardware index scheduler modeling quantum permutation for Grover's search.

Based upon the above discussions and mathematical model, we design an efficient hardware index scheduler to model the quantum permutation, $U_{permute}$. The scheduler hardware architecture is shown in Fig. 38. A mod-*N* counter generates the *j* index values. The set of *I* index values are calculated based on comparisons of each generated *j* index with the target patterns $\{P_0, P_1, \dots, P_{N_{patterns}-1}\}$. The outputs of the comparisons are OR-ed and connected to the control

of a selector/multiplexer (MUX). The MUX sets the value of the *I* index to 0 if there is a match between the *j* index and any of the patterns, or to $N_{patterns}$ if there is no match. The generated *I* and *j* indices are used to read from the $|\psi_2\rangle$ vector and write into the $|\psi_{out}\rangle$ vector, as described by (45).

6.3 Quantum Pattern Recognition

6.3.1 Methodology Overview

We present a quantum-algorithm-based methodology for dimension reduction and pattern matching in high-resolution hyperspectral data. The methodology has two main operations, (1) performing dimension reduction on the input data set while preserving its spatial locality as a pre-processing technique, and (2) searching for the dynamically changing target patterns in the data with reduced dimensionality. The first stage of operations, dimension reduction, is achieved by applying multi-dimensional QHT (1D-, 2D-, and/or 3D-QHT) in multiple decomposition levels, to convert the high spatial resolution of the input data to a desired, low spatial resolution. The multi-level QHT is implemented as cascaded packet wavelet decomposition [56]. A set of input patterns are also provided to the system and a pattern matching search is then performed on the low spatial resolution data set using multi-pattern Grover's search algorithm. In Grover's algorithm, a pattern generally means any binary string representing an integer.

Input classical data is encoded on *n* qubits $|q_0\rangle, q_1\rangle, \dots, q_{n-1}\rangle$, see Fig. 39, representing the *N* basis states of a superimposed quantum state, where $n = \lceil \log_2 N \rceil$. This can be achieved using classical-to-quantum encoding methods as described in [16], or the C2Q methods proposed in this work. For example, one of the methods described in [16] is pure state synthesis, i.e., the problem of encoding data in a quantum state reduces to the problem of synthesizing the state. The *SynthesizePureState* algorithm described in [16] and [44] has a complexity of $O(N^2)$, which is

similar to the cost of processing an $N \times N$ image using classical methods. Therefore, the classical-to-quantum encoding is a cost worth paying especially if the subsequent quantum algorithms provide substantial speedup compared to the classical equivalents.

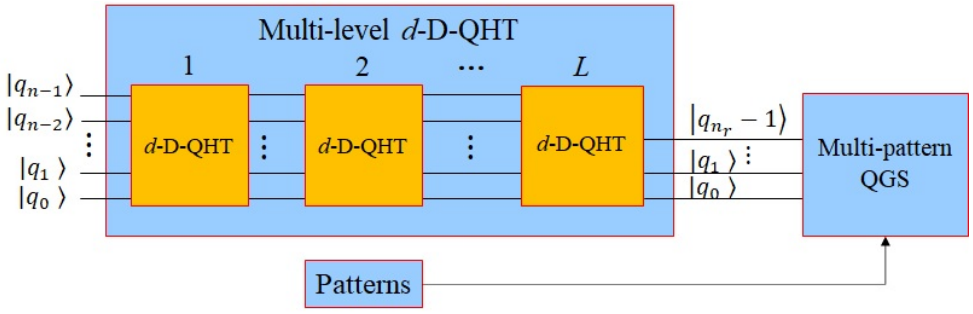


Fig. 39: Overview of methodology for pattern recognition using dimension reduction.

The input qubits, assuming the data has been encoded, undergo L decomposition levels, where $L = \left\lfloor \frac{1}{d} \log_2 \frac{N}{N_r} \right\rfloor$, where $d = 2$ for 2D-QHT, $d = 3$ for 3D-QHT, and N_R is a fixed and pre-determined number of states less than N that represents the size of the data with reduced dimensionality. The number of qubits needed to represent the data with reduced dimensionality decreases to $n_r = \lceil \log_2 N_r \rceil$. It is desired to perform multi-pattern quantum Grover's search (QGS) for a given $N_{patterns}$ number of patterns/basis states using the n_r qubits $|q_0\rangle$, $|q_1\rangle$, ... , $|q_{n_r-1}\rangle$, see Fig. 39. For the pattern search, m iterations [37] of multi-pattern quantum Grover's search (QGS) is applied. In the next sections, the QHT and QGS circuits that will be used for this methodology are discussed.

6.3.2 Quantum Circuits

Two variants of implementing QHT circuits (Sequential QHT and Parallel QHT) have been discussed previously, see Figs. 31(b) and 31(c) respectively. Considering the optimized Parallel QHT circuit variant in Fig. 31(c), we generalized the number of steps of swap gates required for

the permutations as a function of the number of qubits, *n*, the number of dimensions of the kernel, *d*, and the number of qubits representing the d^{th} dimension, n_d. For input and output permutations, the number of steps, or circuit depth is given by (46) and (47) respectively. The circuit depth for *d*-dimension Haar operation is always 1 since the circuit is one level of Hadamard gate(s). Therefore, total circuit depth for multi-level, multi-dimensional QHT, taking into account the number of levels of decomposition, *L*, is given by (48).

$$n_{depth}^{P_{in}} = (n - n_d) - (d - 1) \tag{46}$$

$$n_{depth}^{P_{out}} = n - n_d \tag{47}$$

$$\begin{aligned} n_{depth}^{QHT} &= \left(n_{depth}^{P_{in}} + 1 + n_{depth}^{P_{out}}\right) \cdot L \\ &= (2(n - d + 1) - n_d) \cdot L \end{aligned} \tag{48}$$

For dynamic multi-pattern Grover's search, we extended the conventional single-pattern Grover's search algorithm by modifying the *phase inversion* stage of the algorithm. Overview of the methodology for this process is shown in Fig. 34, where $|\psi_{in}\rangle$ is the output from 2D-QHT or 3D-QHT, *P* is the list of patterns to be searched for, and *S* is a series of indexes ranging from $|0\rangle$ to $|N_{patterns} - 1\rangle$. The process to find the circuit depth for multi-pattern Grover's search algorithm can be separated into the four separate stages that are shown in Fig. 34, and is described in (49) where n_{depth}^{QGS} is the number of time steps or circuit depth of Grover's algorithm and *m* is the amount of times U_{oracle} and $U_{diffusion}$ stages, see Fig. 34, are repeated.

$$n_{depth}^{QGS} = n_{depth}^{H} + m\left(n_{depth}^{oracle} + n_{depth}^{diffusion}\right) + n_{depth}^{permute} \tag{49}$$

In the $H^{\otimes n}$ stage, the depth is simply 1, i.e., $n_{depth}^{H} = 1$, as one *H* gate is applied to each qubit. This can be done in one time step as each *H* operation is independent from each other. In the *phase inversion* stage of Grover's algorithm, oracle circuits [58] are generally implemented using a *cZ*

gate, and multiple *X* gates. To make the pattern search dynamic, we proposed using *cX* or controlled *X*-gates, with the index at *S* acting as the controlling qubits. This modified U_{oracle} circuit for single-pattern Grover's search is shown in Fig. 35(a), where the input quantum state, formed by qubits $|q_0\rangle$, $q_1\rangle$, ... , $q_{n-1}\rangle$, is in equal superposition of its basis states after applying an *H* gate to each qubit as shown by the $H^{\otimes n}$ block in Fig. 34. The *X*-gates controlled by the search pattern dynamically changes the basis state that the oracle is searching for. The use of *cX*-gates also allows us to generalize the algorithm for multi-pattern search. Fig. 35(b) shows the proposed oracle for dynamic multi-pattern, dynamic Grover's search. To search for multiple patterns, $N_{patterns}$ single-pattern oracle circuits must be cascaded, with each oracle circuit controlled by the corresponding index qubits as described by:

$$U_{oracle} = U_{oracle_{N_{patterns}-1}} \cdot \ldots \cdot U_{oracle_i} \cdot \ldots \cdot U_{oracle_0} \tag{50}$$

In the single pattern oracle circuit, see Fig. 35(a), the *cX* gates operate independently from each other as each *cX* gate operates on a qubit pair with no overlap. From this, the depth of the single pattern oracle circuit is 3 with a *cX* step followed by a $c^{n-1}Z$ step and lastly an additional *cX* step. The depth for the multi-pattern circuit is simply $3 \times N_{patterns}$ as each pattern has its own single pattern circuit as shown in Fig. 35(b). The total depth of the U_{oracle} , $n_{depth}^{oracle} = 3 \times N_{patterns}$.

For the next stage, i.e., *inversion about mean*, the circuit we are using is identical to the traditional Grover's algorithm inversion about mean circuit [58]. Additionally, just like in traditional Grover's algorithm, the *phase inversion* and *inversion about mean* circuits are iterated *m* times as described in (10). The depth of the *inversion about mean*, $n_{depth}^{diffusion} = 5$. The circuit first applies an *H* gate to each qubit followed by an *X* gate again to each qubit, then there is a single $c^{n-1}Z$ gate which is followed again by an *X* and an *H* gate applied to each qubit, resulting in a

total of 5 time-steps. This method amplifies the first $N_{patterns}$ states, so a permutation stage dependent on the pattern being searched for is needed.

The permutation circuit has five sub circuits, see Fig. 36, in which the detect and toggle circuits have the same depth. The detect and toggle circuits each has a depth of 3 similar to the single pattern oracle circuit as the *cX* gates operate independently from each other on qubit pairs with no overlap. In the case of the swap circuits, each qubit has a $cc'cX$ and $ccc'X$ gate. These gates are mutually exclusive because if one gate is applied the other gate is guaranteed to not be applied due to the control qubits. However, the *cX* gates between each qubit are not mutually exclusive as they all depend on using the ancillary qubit. This results in a circuit depth of *n*. For multi-pattern cases the circuit depth is again multiplied by $N_{patterns}$ similar to the oracle circuit. Combining everything together gives the following circuit depth for the $U_{permute}$ stage,

$$n_{depth}^{permute} = (9 + 2n) \times N_{patterns} \tag{51}$$

Substituting in all four of the depth equations into (49) results in the final QGS circuit depth given in (52).

$$n_{depth}^{QGS} = 1 + m\left(3N_{patterns} + 5\right) + (9 + 2n)N_{patterns} \tag{52}$$

Using the proposed methodology and quantum circuits of QHT and QGS, it is possible to achieve polynomial speedup over classical methods and techniques. The best known classical search algorithm has complexity of $O(N)$ [5] [37], while QGS provides quadratic speedup with complexity of $O\left(\sqrt{N}\right)$ [5] [37]. Applying dimension reduction using QHT reduces the state space for Grover's search from N to N_r, thereby improving the complexity to $O\left(\sqrt{N_r}\right)$, where $N_r = \frac{N}{2^{dL}}$, where d is the number of data dimensions. Moreover, the use of QHT compared to a classical

method such as DWT also improves the complexity from $O(N)$ to $O(\log_2 N)$ because of encoding the classical data as state coefficients/amplitudes [16].

6.3.3 Considerations for Practical Quantum Pattern Recognition

In practical implementation of quantum circuits, decoherence [11] plays an important part and is a critical consideration in the design of quantum computers. Decoherence is the noise in quantum circuits that disrupts the desired evolution of the quantum state. For any quantum circuit, the duration of the longest possible quantum computation is the ratio of the system decoherence time, i.e., the total time the system remains quantum-mechanically coherent, to the time taken for basic two-qubit unitary transformations [11]. Estimates of the total number of operations possible on different technologies of quantum computers such as nuclear spin, ion trap, quantum dot, etc., are given in [11]. For example, an ion trap quantum computer has a decoherence time of around 10^{-1} seconds and a gate operation time of 10^{-14} seconds, and can therefore perform up to 10^{13} operations [11]. From our circuit analysis in previous sections, n_{depth}^{QHT} and n_{depth}^{QGS} can be used along with the technology gate operation time to determine the practical implementation of the proposed circuits.

Fidelity of quantum gates is another important practical consideration. In quantum information theory, fidelity is used to measure how close two quantum states are. It is the probability that one state will pass a test and identify as the other [11]. Fidelity threshold of quantum gates is dependent on the underlying quantum technology. For example, superconducting quantum gates have a per-step fidelity threshold of 99% [8]. On the other hand, silicon-based qubit technology have achieved gate fidelities exceeding 99.9% [59]. Quantum gate fidelity can be improved by using additional error-correcting qubits.

Chapter 7: Experimental Results and Analysis

7.1 Experimental Platforms

7.1.1 DirectStream

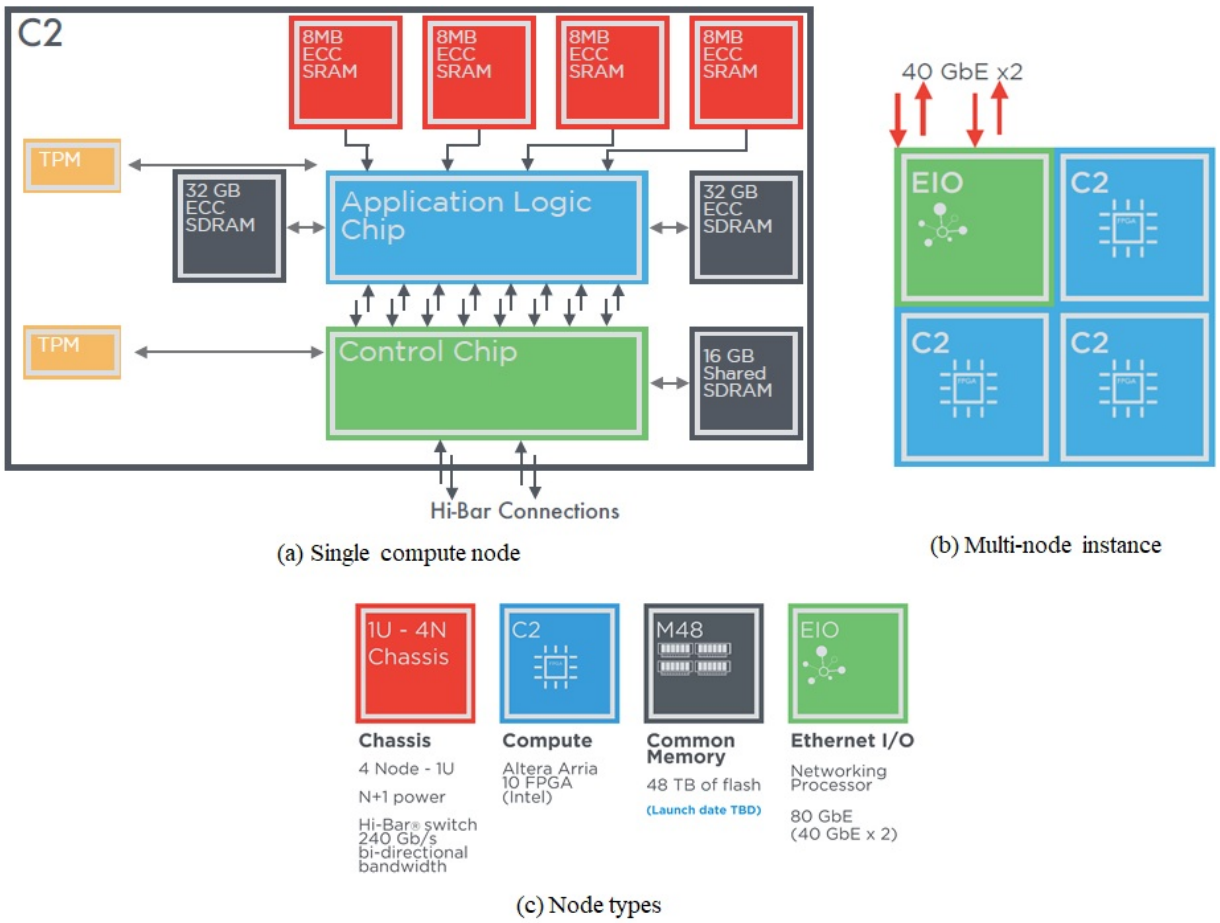


Fig. 40: DirectStream (DS8) system architecture.

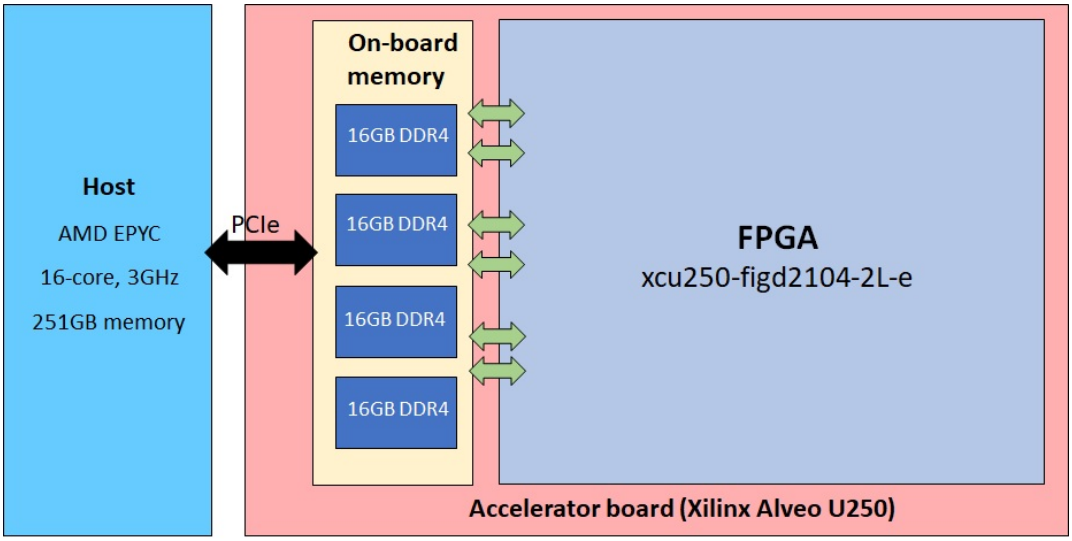
One of the evaluation platforms used for the experimental work was DS8, a state-of-the-art high-performance reconfigurable computing (HPRC) system provided by DirectStream. DS8 is a platform where developers can build applications onto systems ranging from single-node compute instances to multi-node chassis to multi-chassis racks, see Fig. 40. The DS8 system removes OS elements and is an FPGA-only hardware architecture. This has benefits such as reduced interconnection bottlenecks, reduced resource contention, and reduced cost and energy use, compared to conventional CPU+FPGA architectures. A single C2 compute node of the DS8

system is equipped with high-end Intel-Altera Arria 10 10AX115N4F45E3SG FPGA and on-board memory (OBM) SDRAM and SRAM modules, as shown in Fig. 40. The FPGA on-chip resources (OCR) consist of 427,200 Adaptive Logic Modules (ALMs), 2,713 Block RAMs (BRAMs), and 1,518 Digital Signal Processing (DSP) blocks, while the on-board memory (OBM) consists of 4×8MB SRAM banks and 2×32GB SDRAM banks. The DS8 hardware system is integrated with DirectStream's programming environment, which succeeds the previous Carte-C compiler [60]. DirectStream's environment uses a High-Level Language (HLL) which facilitates the development of complex, parallel, and reconfigurable codes in an efficient manner. The study in [61] showed that Carte-C has a highly productive environment, short acquisition time, short learning time as well as a short development time. The DS8 architecture provides a combination of high performance, high scalability, runtime reconfiguration, and ease of use.

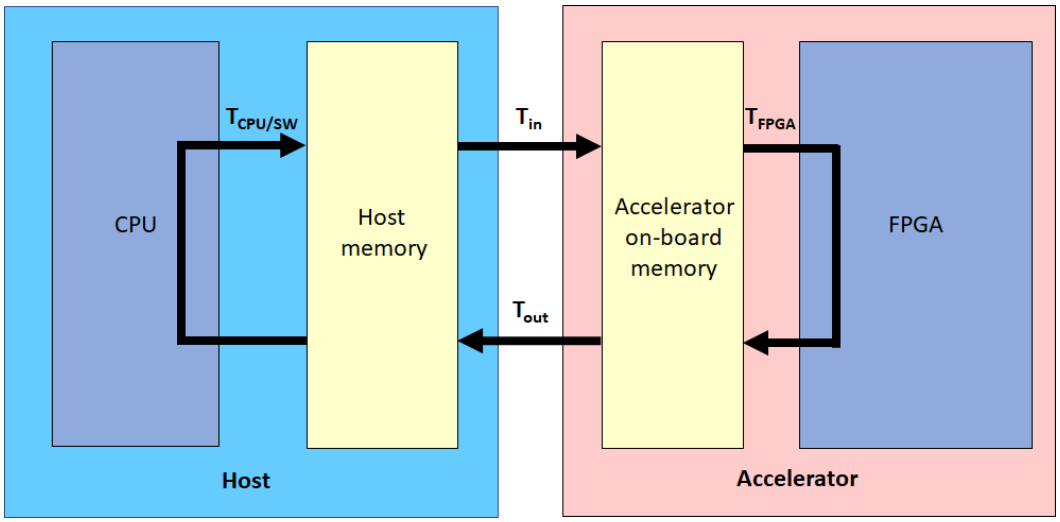
7.1.2 Xilinx Alveo

The second evaluation platform used for the experimental work was an HPRC system based on Xilinx Alveo U250 Data Center Accelerator [62] connected to a host PC, see Fig. 41. The host PC has the following configuration: 16-core, 3GHZ AMD CPU with 251GB system memory, and uses fast Gen3 PCIe for host-to-board configuration and data communications, see Fig. 41(a). The Alveo U250 board contains an XCU250 FPGA that uses Xilinx stacked silicon interconnect (SSI) technology. SSI technology allows for increased density by combining 4 super logic regions (SLRs). The deployment shell that handles device bring-up and configuration over PCIe is contained within a static region of the FPGA. The remaining dynamic region is available for developers to implement custom accelerators and kernels. The dynamic regions resources consist of 1341K look-up tables (LUTs), 2,749K registers, 2000×36KB block RAMs, and 11,508 DSP slices. In addition, the on-board memory resources consist of four 16GB 288-pin DDR4 DIMM

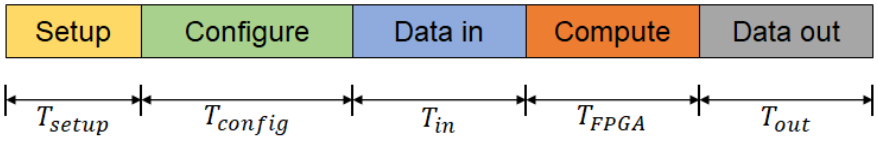
sockets populated with single rank DIMMs, see Fig. 41(a), with data transfer rates up to 2400 MegaTransfers per second.



(a) Xilinx Alveo System Architecture.



(b) Measured Execution Times on the Host and Accelerator.



(c) Timing profile for the Accelerator.

Fig. 41: Xilinx Alveo System Architecture and Timing Profile.

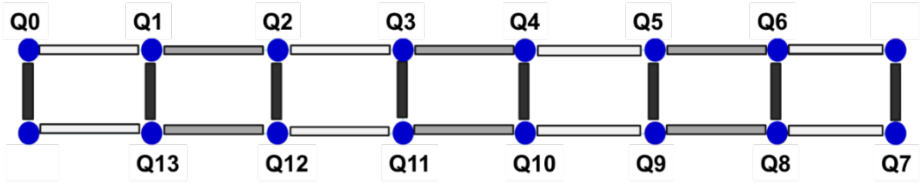
The measured execution times on the system for the host and the accelerator is shown in Fig. 41(b). The timing profile for the hardware accelerator is shown in Fig. 41(c). The time taken by the host to perform memory allocation, setup kernel objects, kernel queues, etc. is termed as T_{setup}. The time taken to program and configure the FPGA via PCIe is termed as T_{config}. The time taken to transfer data from the host memory to on-board memory of the FPGA is termed as T_{in}, and the time taken to transfer data from the on-board memory to the host memory is termed as T_{out}, see Figs. 41(b) and 41(c). The compute time spent in the kernel on the FPGA is termed as T_{FPGA} and it also includes the data transfer times between the FPGA and the on-board memory, see Figs. 41(b) and 41(c). The software time is denoted as T_{SW} or T_{CPU} and constitutes the total time taken by the host (including host memory transfers) to execute the architectures, see Figs. 41(b).

7.1.3 IBM Quantum

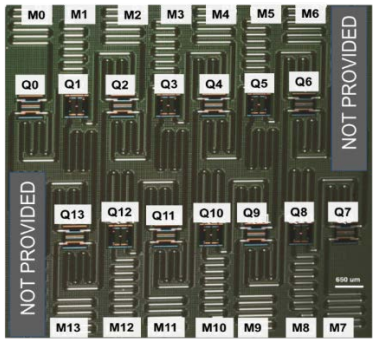
The third platform used for experimental evaluation was IBM Quantum [63]. The IBM Quantum is an integrated quantum computing system consisting of a number of custom components (a) the quantum chip or processor built using superconducting qubits, (b) a cryogenic system for consistent cold temperatures and isolation from environment, (c) high-precision control electronics to tightly control a large number of qubits within strict parameters, and (d) classical resources to provide secure cloud access and hybrid execution of quantum algorithms. IBM Quantum provides quantum processors of varying architectures (e.g. Falcon, Hummingbird, Eagle), varying scale (number of qubits), quality (quantum volume [64]), and speed (Circuit Layer Operations per Second or CLOPs [65]). IBM Quantum also provides a python-based framework called *Qiskit* [66] for programming quantum systems on the cloud, a GUI-based interface called

Composer for building quantum circuits graphically, and a general-purpose simulator with noise modeling called IBM *qasm* [66].

For our experiments we used an open-access 15-qubit quantum processor from IBM Quantum, called the *ibmq_16_melbourne* [66]. The qubits of *ibmq_16_melbourne* have, on average, an operating frequency of 4.98 GHz, T1 (amplitude damping) time of 58.28 μs and T2 (decoherence time) of 62.1 μs. The connectivity on the device is provided by total 22 coplanar waveguide (CPW) "bus" resonators, each of which connects two qubits. The connectivity configuration is shown in Fig. 42(a). The colored dots indicate qubits, and the colored bars indicate CPW bus resonators. Three different resonant frequencies are used for the bus resonators. The white bars indicate the buses with a resonant frequency of 6.25 GHz, the grey bars indicate 6.45 GHz, and the black bars indicate 6.65 GHz. Two of the qubits in the chip are not calibrated due to frequency instability and crosstalk issues [66]. Each qubit has a dedicated CPW readout resonator attached (labeled as R) for control and readout. Fig. 42(b) shows the chip layout.



(a) Connectivity configuration of the ibmq_16_melbourne processor



(b) Chip layout of the ibmq_16_melbourne processor

Fig. 42: The ibmq_16_melbourne processor connectivity and layout

7.2 Evaluation of Classical-to-Quantum Data Encoding

7.2.1 C2Q Method 1 Experiments

The proposed method 1 for C2Q data encoding has been evaluated using (1) MATLAB, for simulation using noise-free qubits, (2) IBM Quantum, for noisy qubits on a real Noisy Intermediate-Scale Quantum (NISQ) device, and (3) Xilinx Alveo, for hardware-accelerated, noise-free emulation. In the IBM Quantum environment, simulations were performed using the IBM *qasm* simulator, while real implementations were performed on the 15-qubit real quantum processor, *ibmq_16_melbourne*. Synthesis of two types of target data was performed: (1) complex randomized data, and (2) real grayscale image data.

Table 5: Simulation and Implementation of Proposed C2Q Circuits using IBM Q.

	Number of data items (N)	Number of qubits (n)	Simulation (MATLAB / ibm_qasm)		Implementation (ibmq_16_melbourne)	
			Gate Count	Circuit Depth	Gate Count	Circuit Depth
Complex randomized data	4.00E+00	2	7.00E+00	6.00E+00	1.70E+01	1.50E+01
	1.60E+01	4	3.30E+01	3.00E+01	9.20E+01	7.20E+01
	6.40E+01	6	1.31E+02	1.26E+02	3.38E+02	2.76E+02
	2.56E+02	8	5.17E+02	5.10E+02	NA due to hardware limitations of IBM-Q	
	1.02E+03	10	2.06E+03	2.05E+03		
	4.10E+03	12	8.20E+03	8.19E+03		
	1.64E+04	14	3.28E+04	3.28E+04		
Image data	2 x 2 pixels	2	3.00E+00	2.00E+00	1.30E+01	1.10E+01
	4 x 4 pixels	4	1.70E+01	1.40E+01	5.80E+01	4.70E+01
	8 x 8 pixels	6	6.70E+01	6.20E+01	2.38E+02	1.97E+02
	16 x 16 pixels	8	2.61E+02	2.54E+02	8.71E+02	7.46E+02
	32 x 32 pixels	10	1.03E+03	1.02E+03	NA due to hardware limitations of IBM-Q	
	64 x 64 pixels	12	4.11E+03	4.09E+03		
	128 x 128 pixels	14	1.64E+04	1.64E+04		

MATLAB and IBM Quantum Results: The experimental results from MATLAB and IBM Quantum are presented in Table 5. For complex randomized data, the circuit depths reach the theoretical upper bounds derived earlier in section 3.2.3 as the full synthesis circuit is required. For real image data, the gate counts and circuit depths of the circuits were reduced by at least a factor of two, as there are no imaginary components ($t_j = \phi_j = 0$) in the data, and thus both the uniformly-controlled R_Z operations, i.e., $R_z(\phi_j) = I$, and their corresponding CNOT operations

are eliminated. Results obtained from IBM *qasm* simulations of up to 14-qubit circuits were consistent with our theoretical expectations for circuit depth, see Table 1 and Table 5. Due to hardware constraints for the *ibmq_16_melbourne* device, gate counts and circuit depths were obtained for circuits up to only 6 qubits (complex randomized data) and 8 qubits (real image data). Several of the gates used in our proposed circuit, such as H, CNOT and R_y, are not physically realizable on the *ibmq_16_melbourne* device and are instead replaced in a transpilation process using a different subset of universal gates that are native to the IBM Q platform. The transpilation step resulted in higher gate counts and circuit depths for the implementations, compared to our theoretical expectations, see Table 1 and Table 5. For larger data sets that require a large number of qubits, and consequently larger synthesis circuits, the system decoherence time (T2) on *ibmq_16_melbourne* was exceeded, limiting implementations to only 6 qubits (complex randomized data) and 8 qubits (real image data). For simulations and implementations on IBM Q, the circuits were executed with 8000 shots (iterations) to measure the probability distributions of the output states.

To verify the correctness of the proposed C2Q methodology and circuits, the encoded images were reconstructed from the synthesized state coefficients and the fidelity of the synthesized state was calculated. The state fidelity is a measure for the similarity of the measured output state $|\psi_{measured}\rangle$, observed in simulation or implementation, to the theoretical or expected state $|\psi_{expected}\rangle$. The Uhlmann-Jozsa fidelity for pure states [67] [68], given in (53), is used for our experiments.

$$F = \left|\langle\psi_{expected}|\psi_{measured}\rangle\right|^2 \tag{53}$$

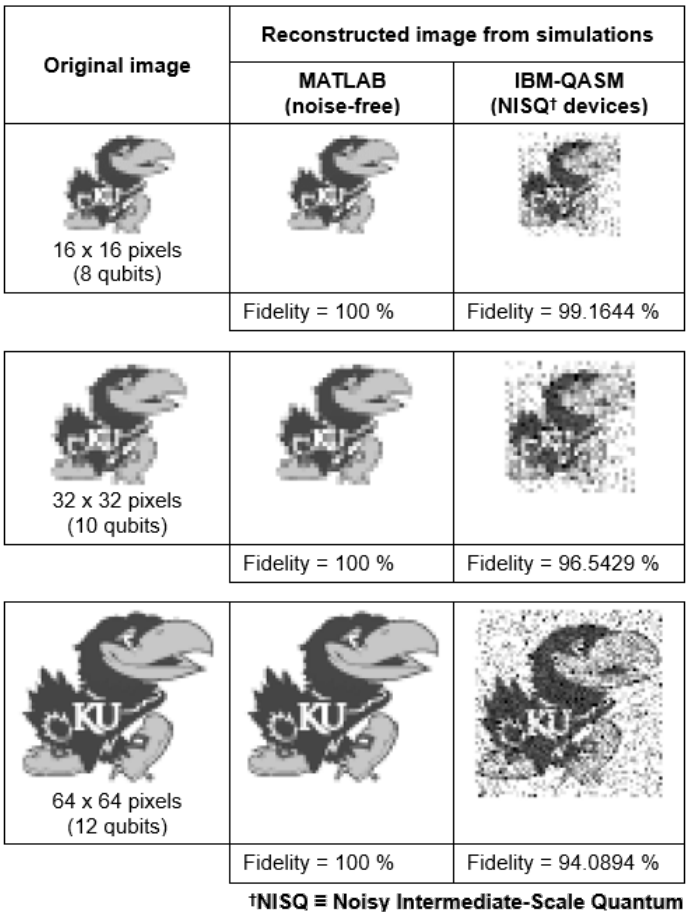


Fig. 43: Original and reconstructed images from synthesized quantum states.

Fig. 43 shows 16×16, 32×32, 64×64-pixel grayscale images encoded using 8-qubit, 10-qubit, and 12-qubit synthesis circuits respectively in both MATLAB and IBM Q. The reconstructed images from the synthesized state are also shown along with the corresponding state fidelity between the original data and the reconstructed data. When the images were encoded as pure states using noise-free qubits in MATLAB, the reconstructed images were identical to the original images, i.e., F = 100%, see Fig. 43. For simulation on realistic Noisy Intermediate Scale Quantum (NISQ) devices, such as the *ibmq_16_melbourne*, the reconstructed images were partially corrupted by device noise. The state fidelity between the original data and the reconstructed data was 99.1644%, 96.5429%, and 94.0894% for the 16×16, 32×32, and 64×64-pixel images respectively, see Fig. 43.

Hardware Emulation Results: The hardware platform used for the evaluating the proposed C2Q architectures was the Xilinx Alveo U250 Data Center Accelerator, see Fig. 41. The Vitis Unified Software from Xilinx [62] was used for design and hardware deployment. MATLAB R2020a was used for data pre-processing, post-processing, and visualizations. For the purposes of comparison and verification, a software-based emulator was also created for the proposed architectures using C++. The Qiskit framework from IBMQ [66] was also used for implementing the proposed quantum circuits and the QASM simulator [66] was used for simulating the circuits on an IBM Quantum cloud-based server.

The hardware architectures for C2Q, see Fig. 12, were implemented as reconfigurable hardware kernels, *kernel_c2q* and *kernel_qht* on the FPGA. The extraction of the 4-tuple (θ, φ, r, t) of parameters from input dataset is performed on the host machine. The parameters and input/output state vectors $|\psi_{in}\rangle$, $|\psi_{out}\rangle$ are stored on the on-board memory and transferred to the kernel reconfigurable regions during computation. The host machine controls memory transfers and kernel execution commands via a high-speed PCIe bus. The *kernel_c2q* is executed first, which operates on the input parameters and synthesizes the input quantum state $|\psi_{in}\rangle$, which is stored on the on-board memory. The input quantum state vector is then transferred to the *kernel_qht*, which executes the parallel *l*-level d-dimensional QHT algorithm and produces output state vector $|\psi_{out}\rangle$, that is transferred back to on-board memory.

The OpenCL framework [69] was used for development of the kernels and host program. The kernel architectures were fully pipelined and computation operations were implemented with 32-bit floating-point arithmetic. RGB images with sizes ranging from $16 \times 16 \times 4$ pixels to $32K \times 32K \times 4$ pixels were used as input data. For these images, C2Q circuits requiring 10 to 32 qubits were emulated using the implemented emulation architectures.

Hardware (HW) run-time results from the conducted experiments are shown in Table 6 for the C2Q kernel. Measurements of T_{in}, T_{FPGA}, and T_{out} were taken from host-controlled executions on the FPGA, see Table 6. Data packing techniques were employed to fully utilize the host-to-FPGA bandwidth and achieve optimal data transfer and compute times. The setup time T_{setup} and configure time T_{config}, see Fig. 41(c), were not included in the analysis to be consistent with CPU-based experiments. The total HW run-time reported is the sum of the time taken to transfer data from the host to the Alveo board, the time taken for emulation computations on the FPGA, and the time taken to transfer data back to the host, i.e., $T_{total}(\text{HW}) = T_{in} + T_{FPGA} + T_{out}$.

Experiments using the same RGB images were repeated on the SW emulator. C2Q circuits requiring 10 to 32 qubits were run on the software emulator. The 4-tuple (θ, φ, r, t) of input parameters as well as the $|\psi_{in}\rangle$, and $|\psi_{out}\rangle$ state vectors were stored in heap-allocated memory after reading input data files and performing computations, respectively. Measurements of software (SW) run-time shown in Table 6 were taken from kernel executions on a single core of the CPU on the host machine. The total CPU run-time, denoted as $T_{total}(\text{SW})$, is the time taken to perform the mathematical operations on the inputs and includes data transfer time between host memory and CPU. The time taken to read the input data files is not included in the reported timings.

Table 6: Run-time results for emulation of C2Q using Xilinx Alveo.

Classical-to-Quantum (C2Q)												
Data size (no. of pixels)	No. of states (N)	No. of qubits (n)	HW run-time (ms)				SW run-time (ms)	QASM run-time (ms)	Speedup			
			T_in	T_FPGA	T_out	T_HW	T_SW	T_QASM (shots=1)	HW/QASM	SW/QASM	HW/SW	FPGA/SW
16x16x4	1024	10	0.12	0.04	0.10	0.26	0.03	8.95	35	289	0	1
32x32x4	4096	12	0.13	0.04	0.11	0.28	0.15	79.59	282	536	1	4
64x64x4	16384	14	0.13	0.09	0.09	0.31	0.31	738.54	2,360	2,367	1	3
128x128x4	65536	16	0.19	0.15	0.15	0.49	1.32	2,096.94	4,306	1,594	3	9
256x256x4	262144	18	0.30	0.34	0.26	0.90	5.31	19,614.12	21,815	3,695	6	16
512x512x4	1048576	20	0.55	1.10	0.52	2.17	21.69	240,249.24	110,510	11,076	10	20
1024x1024x4	4194304	22	1.83	4.23	1.55	7.61	88.85	Data not collected due to IBMQ limitations			12	21
2048x2048x4	16777216	24	8.81	16.65	7.96	33.42	358.24				11	22
4096x4096x4	67108864	26	37.94	66.47	30.47	134.88	1,460.45				11	22
8192x8192x4	268435456	28	153.30	266.07	105.66	525.03	5,716.63				11	21
16384x16384x4	1073741824	30	480.63	1,064.21	370.23	1,915.07	22,891.06				12	22
32768x32768x4	4294967296	32	1,594.50	4,674.30	1,237.96	7,506.76	91,577.17				12	20

For reference, the proposed quantum circuits for C2Q (method 1) were also implemented on Qiskit and simulated using the IBM Quantum QASM simulator. Simulation results for C2Q are shown in Table 6. Circuit execution times for up to 20-qubit circuits were measured for C2Q and simulations of larger circuits were not possible due to memory constraints on the IBM Quantum server machine.

The HW implementation was benchmarked using the SW implementation as a baseline. The HW and SW run-times are presented graphically in Fig 44 for the C2Q kernels. The SW implementation performs better than the HW up to 14-qubit circuits as the CPU and host memory subsystem are able to take advantage of data caching. However, for larger image sizes and larger circuit emulations, the data caching is throttled, and the HW performance improves as it is able to take advantage of the FPGA's high bandwidth and fine-grain parallelism.

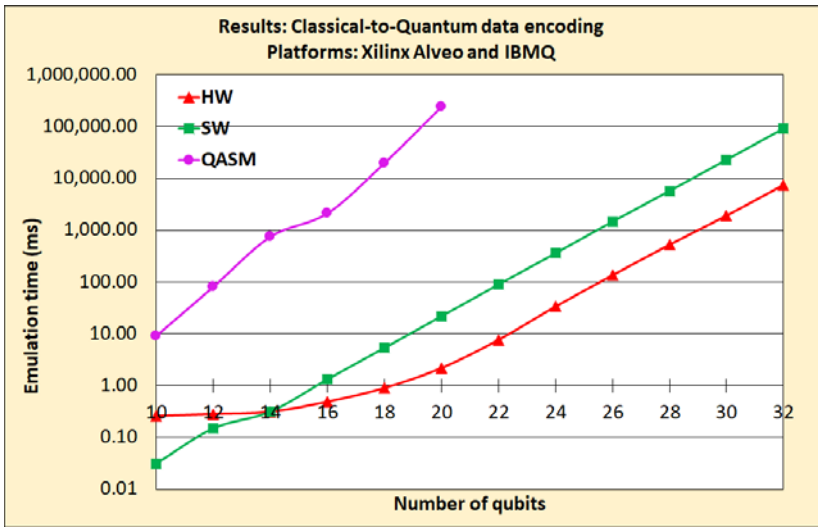


Fig. 44: C2Q emulation run-times on different platforms.

We determined two types of speedup of the HW relative to the reference SW implementation, see Table 6 and (54). We calculated the speedup of the HW implementation relative to the SW implementation and observed up to $\times 12$ improvement in favor of HW for the C2Q kernel, see Table 6. To compare the performances of the FPGA and CPU, we calculated speedup of the total

FPGA execution time relative to the total CPU execution time. For large input data, the FPGA speedup relative to CPU was $\times 21$ for the C2Q kernel. The FPGA was able to fully exploit its parallelism by instantiating concurrent processing elements for each data point.

$$\text{Speedup}_{(HW/SW)} = \frac{T_{total}(SW)}{T_{total}(HW)}$$
$$\text{Speedup}_{(FPGA/SW)} = \frac{T_{total}(SW)}{T_{FPGA}} \quad (54)$$

The HW implementation was also compared with circuit simulation on the QASM simulator. The HW emulation run-times and QASM simulation times are shown in Fig 44. For C2Q, due to the complexity of the circuits, the increase in simulation time is relatively much steeper compared to the HW emulation time, see Fig. 44. The QASM simulator was also able to simulate up to 20-qubit C2Q and 26-qubit QHT circuits, while HW emulation was more scalable up to 32-qubit circuits. For 20-qubit C2Q circuits, the HW emulation achieved 5 orders of magnitude speedup compared to the QASM simulation, see Table 6.

7.2.2 C2Q Method 2 Experiments

The following related methods were considered for implementation and comparison with our proposed method 2 for C2Q.

IBM Quantum: There are two state initialization/preparation functions in Qiskit, i.e., *initialize()* and *StatePreparation()*. The default function for qubit state initialization in Qiskit is *initialize()* . It takes a list of state vectors and the number of qubits as inputs, and returns a state initialization circuit. In addition to further optimizations, the circuit construction follows the methodology proposed in [19]. There are also initial reset gates on all the qubits in the circuit. The *StatePreparation()* function also follows the recursive initialization algorithm proposed in [19],

and includes additional optimizations such as removing zero rotations and pairs of consecutive CNOT gates. It is similar to *initialize()*, but does not contain any reset gates.

Novel Enhanced Quantum Representation (NEQR): The basis encoding technique described in [41] uses basis states to encode the position and color of pixels (8 qubits per grayscale pixel + N qubits for position), where N is the total number of pixels. The benefit of basis encoding is zero fidelity [67] loss when each pixel is measured and observed. However, measuring all the pixels makes this process slower and expensive in terms of qubit requirement compared to other encoding methods. A potential improvement in our implementation is dynamically setting the number of shots by re-running the circuit until all pixels are observed. Due to the higher qubit requirement, we were not able to run this method on hardware.

Flexible Representation of Quantum Images (FRQI): The angle encoding technique described in [42] is similar to the NEQR method, where it uses basis states to encode the position and color of pixels (1 qubit per grayscale pixel + $\log N$ qubits for position). We implemented this method with some modifications for encoding colored images. However, this technique is very inefficient for colored image encoding (each color value per pixel requires 1 qubit) and demonstrated low fidelity.

Analysis of Results: The QASM simulation results are compiled in Table 7 and presented graphically in Fig. 45, displaying the total execution time (circuit setup + circuit execution) plotted against the number of qubits used per method. Results from implementations on the publicly available quantum processor *ibmq_manila* are presented in Table 8 and shown in Fig. 46. The results on *ibmq_manila* were relatively inconclusive. The 5-qubit threshold of the quantum

processor was insufficient to highlight any meaningful difference between the various encoding methods. Moreover, when implementing on quantum processors, the overhead of the control hardware, i.e., the time taken by control hardware to generate and maintain gate pulses, comprises a large portion of the measured execution times. Therefore, it is difficult to compare and analyze the actual circuit execution times for the hardware implementations.

However, meaningful information can be drawn from the QASM simulations. Firstly, the results of our proposed method are consistent with the *StatePreparation()* method in terms of state fidelity and execution time, see Table 7 and Fig. 45. The IBM Quantum methods are slightly faster in execution time (T_{exec}) due to various additional optimizations that are not reported in literature. However, our investigations revealed that these optimizations also add a significant overhead to constructing the circuit, leading our proposed C2Q method 2 to be significantly faster in total execution time, T_{total}, see Table 7 and Fig. 45. The IBM *initialize()* function also includes reset operation. It appears that IBM also includes similar undisclosed software optimizations for *initialize()*, which lower simulation execution time, as shown in Table 7 and Fig. 45.

Table 7: Implementations of C2Q encoding methods on IBM QASM Simulator.

QASM run time (sec) (shots = 16,384)		Proposed					IBMQ Initialize					IBMQ State Preparation					NEQR					FRQI				
Data size (no. of pixels)	No. of states (N)	No. of qubits (n)	T_setup	T_exec	T_total	Fidelity	No. of qubits (n)	T_setup	T_exec	T_total	Fidelity	No. of qubits (n)	T_setup	T_exec	T_total	Fidelity	No. of qubits (n)	T_setup	T_exec	T_total	Fidelity	No. of qubits (n)	T_setup	T_exec	T_total	Fidelity
8x8x3	192	8	2.16	0.12	2.27	99.720%	8	1.61	0.06	1.67	99.718%	8	5.94	0.08	6.01	99.754%	16	5.27	7.32	12.59	100%	9	1.74	0.08	1.82	96.369%
16x16x3	768	10	2.73	0.10	2.83	98.836%	10	2.04	0.10	2.14	98.780%	10	25.65	0.11	25.75	98.748%	18	15.53	30.92	46.45	100%	11	3.34	0.13	3.48	92.557%
32x32x3	3072	12	6.59	0.28	6.86	94.736%	12	1.79	0.15	1.94	94.761%	12	150.43	0.30	150.73	95.141%	20	67.39	143.52	210.91	98%	13	6.67	0.39	7.06	80.438%
64x64x3	12288	14	23.06	2.52	25.58	81.947%	14	3.79	0.11	3.90	82.199%	14	935.52	2.40	937.92	81.990%	22	318.34	768.15	1,086.49	63%	15	19.88	12.68	32.56	57.151%

Table 8: Implementations of C2Q encoding methods on a 5-qubit quantum processor.

No. of qubits (n)	ibm_manila run-time (sec) (shots = 10,000)											
	Proposed			IBMQ Initialize			IBMQ State Preparation			FRQI		
	T_setup	T_exec	T_total	T_setup	T_exec	T_total	T_setup	T_exec	T_total	T_setup	T_exec	T_total
1	1.15	7.55	8.69	0.96	8.01	8.97	0.98	8.03	9.02	1.10	7.85	8.95
2	1.80	7.62	9.42	1.08	8.09	9.17	1.03	7.87	8.91	1.29	7.93	9.22
3	1.17	7.80	8.97	1.40	8.41	9.81	1.07	7.73	8.81	0.98	7.73	8.72
4	2.89	8.15	11.04	1.43	8.16	9.59	2.97	7.73	10.70	1.36	8.21	9.57
5	1.26	8.58	9.83	1.27	8.75	10.03	1.31	8.10	9.41	1.74	7.83	9.57

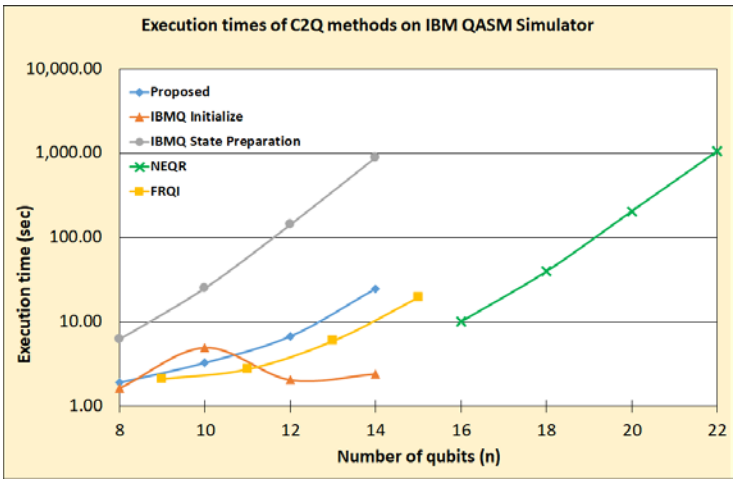


Fig. 45: Simulation times of C2Q encoding methods on IBM QASM Simulator.

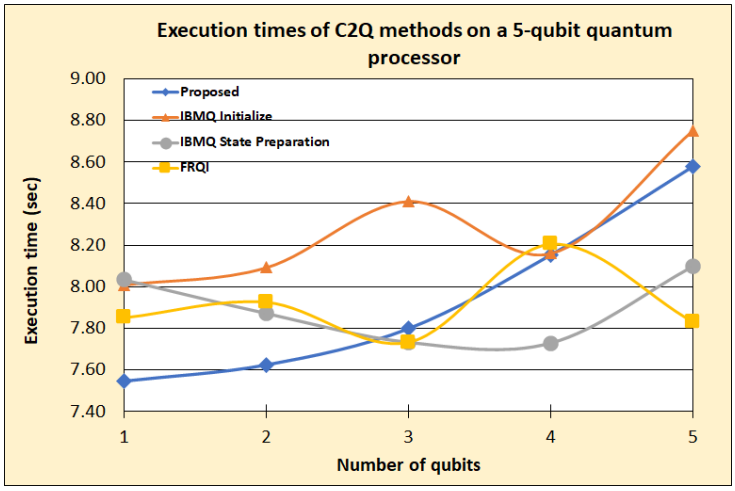


Fig. 46: Hardware execution times of C2Q encoding methods on ibmq manila.

When comparing our proposed method (amplitude encoding) to NEQR (basis encoding) and FRQI (angle encoding), our method illustrates a balance between qubit cost, execution time, and image fidelity. While FRQI can be executed faster compared to our method, it sacrifices qubit cost and image fidelity. Conversely, NEQR theoretically offers perfect fidelity on all measured pixels. In practice, however, NEQR costs a significantly higher number of qubits and incurs substantially longer execution time for the same data size (image pixels), see Table 7 and Fig. 45. Moreover, the number of shots was insufficient for NEQR to measure all required pixels in the larger images, leading to fidelity loss. Resolving this issue would have required increasing the number of shots

for this method, which would have further increased its execution time. The original and reconstructed 64x64x3 images for the implemented C2Q methods are shown in Fig. 47.

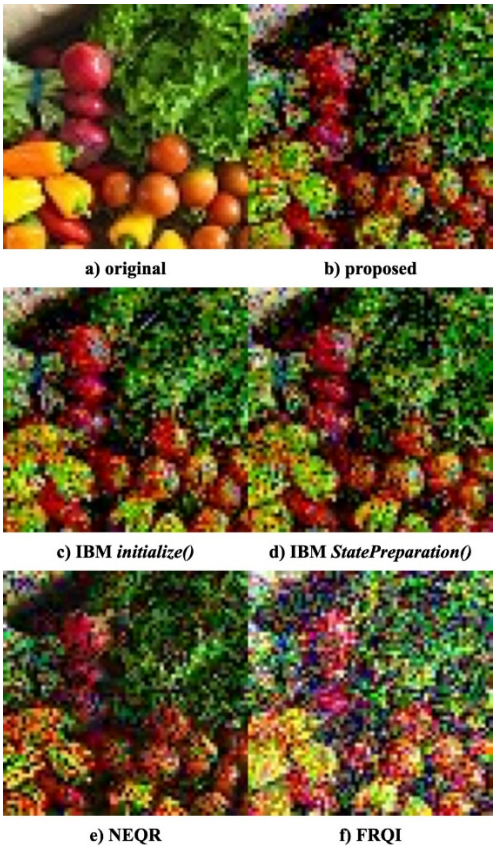


Fig. 47: Original and reconstructed 64x64x3 pixel images for different C2Q methods: (a) original image, (b) proposed, fidelity 81.95% (c) IBM State Initialization, 82.19% (d) IBM State Preparation, fidelity 81.99% (e) NEQR, fidelity 63% (f) FRQI, fidelity 57.15%

7.3 Evaluation of Quantum Algorithms

7.3.1 Implementation of QFT and Grover's search using Gate-based Emulation

We performed experiments to test the proposed gate-based multi-node emulation architectures, see Fig. 19, using QFT and single-pattern Grover's search as test cases. All design components were developed using C++ on the DS8 platform. Hardware synthesis and builds were performed using Quartus Prime Version 17.02. The results were verified with reference models developed using *Qiskit* and MATLAB. For 5-qubit QFT, the design was partitioned, and hardware builds were performed on a 4-node DS8 unit containing three compute (C2) nodes, and a high speed 80

Gigabit Ethernet connection node, see Fig. 40(a) and Fig. 40(b). Each C2 node is populated with an Arria 10 FPGA, i.e., 10AX115N4F45E3SG, see Fig. 40(c). For Grover's search one C2 node was sufficient for the implementation of up to 5 qubits.

Table 9: 5-Qubit QFT Resource Utilization for Multi-Node

FPGA resource	Node 1	Node 2	Node 3
Logic utilization (ALMs)	377,290 (88%)	369,731 (87%)	361,443 (85%)
RAM blocks	800 (29%)	792 (29%)	764 (28%)
DSP Blocks	29 (2%)	26 (2%)	40 (3%)

Table 10: Grover's Search (Hybrid Model) Resource Utilization for Single Node

FPGA resource	3-qubit	4-qubit	5-qubit
Logic utilization (ALMs)	99,436 (23.28%)	100,404 (23.50%)	101,172 (23.68%)
RAM blocks	278 (10.24%)	280 (10.32%)	284 (10.47%)
DSP Blocks	30 (2%)	30 (2%)	30 (2%)

Table 11: Grover's Search (Full Gate Model) Resource Utilization for Single Node

FPGA resource	3-qubit	4-qubit
Logic utilization (ALMs)	197,524 (47%)	374,021 (88%)
RAM blocks	654 (25%)	1,604 (60%)
DSP Blocks	524 (35%)	1,364 (90%)

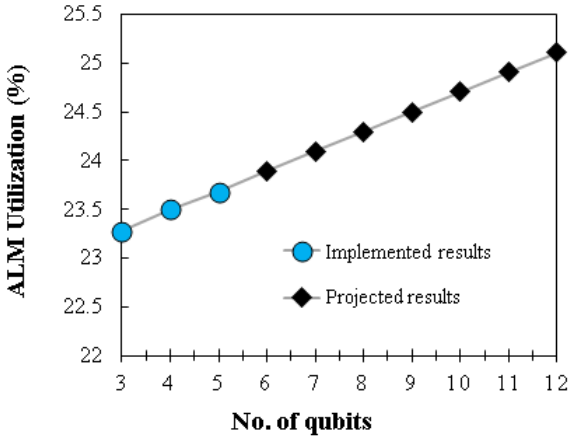
Table 12: Operating Frequencies (MHz)

Grover's search implementation	3-qubit	4-qubit	5-qubit
Lee et. Al. (2016) [29]	160	170	110
Proposed work	233	233	233

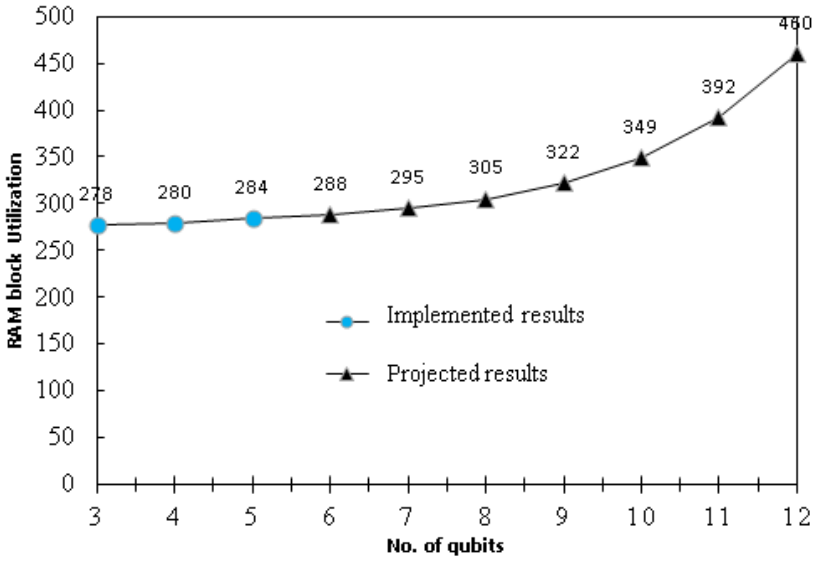
The resource utilization for the experiments related to QFT and Grover's search are presented in Tables 9 and 10 respectively. Table 9 shows the resource utilization on C2 compute nodes 1, 2, and 3 consumed by each partition of the 5-qubit QFT model, see Fig. 19. The resources were shared evenly between the three design partitions. The results in Table 9 show that the ALM resources were a limiting factor for 5-qubit QFT implementation, as approximately one third of the circuit fully consumes a single FPGA node. Low DSP utilization is achieved due to DSP functions being constructed from ALMs, which is handled by the Quartus Prime Compiler [70]. The proposed hybrid design of Grover's circuit had a significant reduction in resources, shown in

Table 10, compared to our preliminary full-gate alternative implementations, shown in Table 11. The proposed architecture, see Fig. 21, allowed for a larger circuit such as the 5-qubit system to be accommodated on a single node. Also, for the case of Grover's algorithm, the benefit of using space-time scheduling techniques in comparison to the full-gate implementation can be observed in Tables 10, 11, and Fig. 48. Resource scheduling results in a fixed amount of DSP resources being used as the quantum circuit grows in size. The increase in Adaptive Logic Module (ALM) utilization is linear, see Table 10 and Fig. 48(a). This is because of the adaptive feature of ALMs for which we directed the compiler to combine multiple functions in a single ALM for efficient usage of resources [70]. Also, the Intel Quartus Prime Compiler automatically searches for functions using common inputs or completely independent functions to be placed in one ALM to make efficient use of device resources [70]. There is an exponential increase in the RAM utilization as the number of memory blocks increases exponentially with the number of qubits *n*, see Fig. 48(b). For 5 qubits, the maximum resource utilization was 24%, see Table 10. Mathematically, we can project the resource utilization for a higher number of qubits as shown in Fig. 48. It is predicted that a system of 12 qubits would consume only 25% of the logic resources (ALMs), see Fig. 48(a). However, due to the exponential increase in RAM resource, see Fig. 48(b), based on our mathematical projections, the current platform can emulate Grover's algorithm up to 17 qubits. The proposed architecture for Grover's search is suitable for integrating into larger quantum circuits and implementing on hardware. Moreover, with a pipelined architecture, the emulator achieved a consistently higher operating frequency, meaning much higher deliverable throughput and lower emulation time. Table 12 demonstrates this improvement in comparison to previous work [29] which was based on Altera Stratix IV EP4SGX530KF43C4 FPGA. Fig. 48(c) shows simulation results for the 5-qubit Grover's search implementation. The binary search pattern was

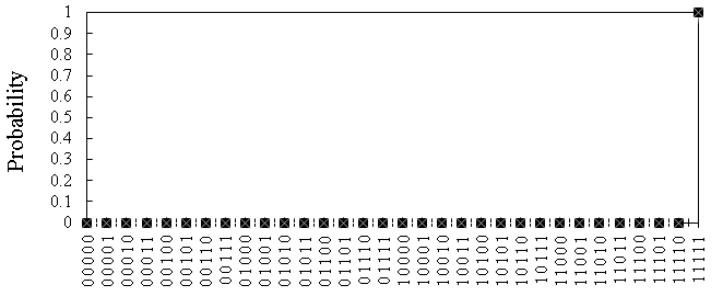
accurately detected by the circuit with a probability of 0.999444077. From the experimental results, we can conclude that for quantum circuits like Grover's search, the scalability of the design can be significantly improved by using our proposed framework and resource scheduling techniques. For circuits such as the QFT, we adopt multi-node, multi-chassis architectures to implement larger scale circuits. More accurate emulation of Grover's search can be achieved by implementing the full quantum gate model instead of a hybrid or abstract model. To make that feasible, a combination of the space and space-time scheduling techniques discussed here are required.



(a) Logic (ALM) utilization for Grover's search implementation on a single Arria 10 FPGA.



(b) RAM block utilization for Grover's search implementation on a single Arria 10 FPGA.



(c) Simulation results for 5-qubit Grover's search for detecting binary string "11111".

Fig. 48: Experimental results for Grover's search.

7.3.2 Implementation of QFT and Grover's search using CMAC-based Emulation

We performed implementations of the CMAC-based emulation model, see Fig. 22, on the DS8 system using QFT and Grover's search as the use cases. Implementations include the various CMAC architectures, (e.g., single, *N*-concurrent, dual-sequential) and different CMAC computation techniques (e.g., *lookup*, *dynamic generation*, *streaming*) discussed previously. All results are collected from hardware deployments on FPGAs with complete system and memory interface implementations on the DS platform. The hardware architectures were implemented in High-Level Synthesis (HLS) using C++. The high-level C++ codes were built for hardware using Quartus Prime version 17.0.2 on an Arria 10 10AX115N4F45E3SG FPGA, and the resource utilizations and latencies were obtained from compiler reports. As a result of fully pipelining the designs, a high operating frequency of 233 MHz was reported, resulting in high system throughput.

We first implemented the single, *N*-concurrent, and dual-sequential-CMAC architectures using the *lookup* technique and both on-chip resources (OCR) and on-board memory (OBM) configurations. Emulation of the QFT algorithm was performed using these implementations. Table 13 reports the on-chip implementation results for the single-CMAC architecture. Fig. 49 presents the resource utilizations as a function of the number of qubits. From this experiment, the ALM and DSP resource utilizations reported were constant, which was a result of using one

CMAC hardware unit. The BRAM units are used for on-chip storage and lookup of the algorithm matrix/vector elements, and the BRAM resource utilization increases exponentially with the number of qubits. An increase in emulation time with circuit size is also observed, as expected, due to the increasing number of temporal iterations of the single CMAC unit.

Table 13: QFT Implementation Results using Single-CMAC architecture, On-chip Resources, and Lookup

Number of qubits	OCR* utilization (%)			Emulation time (sec)
	ALMs	BRAMs	DSPs	
2	10.3	8.04	1.05	1.4E-6
3	10.24	8.12	1.05	1.15E-6
4	10.24	8.11	1.05	2.01E-6
5	10.27	8.18	1.05	5.37E-6
6	10.26	8.55	1.05	1.87E-5
7	10.26	10.25	1.05	7.17E-5
8	10.29	16.73	1.05	3.19E-4
9	10.31	41.28	1.05	0.0013

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518$.

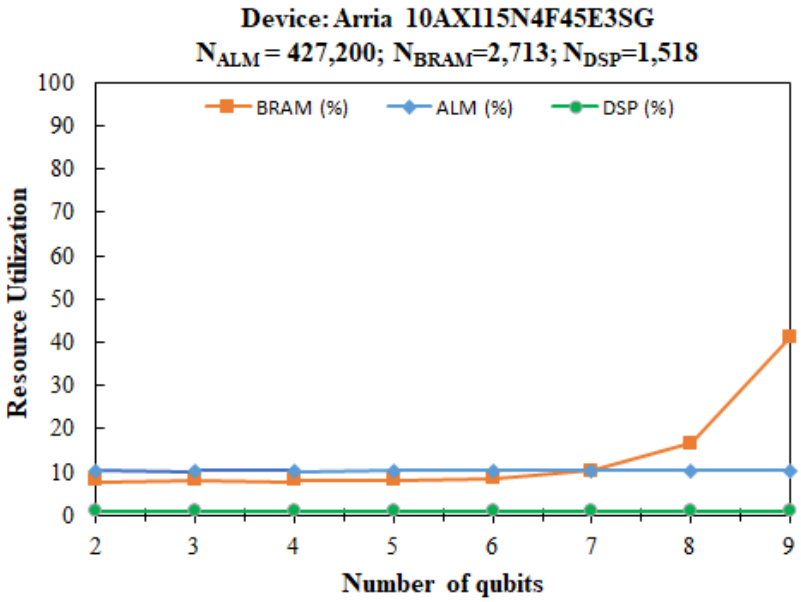


Fig. 49: QFT on-chip resource utilizations using single-CMAC architecture and lookup.

The implementation results of the *N*-concurrent-CMAC architecture are reported in Table 14 and Fig. 50. There is a consistent increase in ALMs as the number of CMAC hardware units in this architecture increases with the number of qubits. The Intel Quartus Prime hardware compiler applies optimizations to maintain the constant utilizations for scarce DSP units. For example, the

compiler automatically searches for functions using common inputs or completely independent functions to be placed in one ALM to make efficient use of device resources [70].

Table 14: QFT Implementation Results using N-concurrent CMAC architecture, On-chip Resources, and Lookup

Number of qubits	OCR* utilization (%)			Emulation time (sec)
	ALMs	BRAMs	DSPs	
2	10.70	7.08	1.05	6.78E-7
3	10.74	7.08	1.05	7.64E-7
4	11.53	7.08	1.05	9.36E-7
5	17.10	7.08	1.05	1.28E-6
6	24.50	7.08	1.05	1.97E-6
7	39.50	7.08	1.05	3.34E-6
8	74.88	7.08	1.05	6.09E-6

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518.$

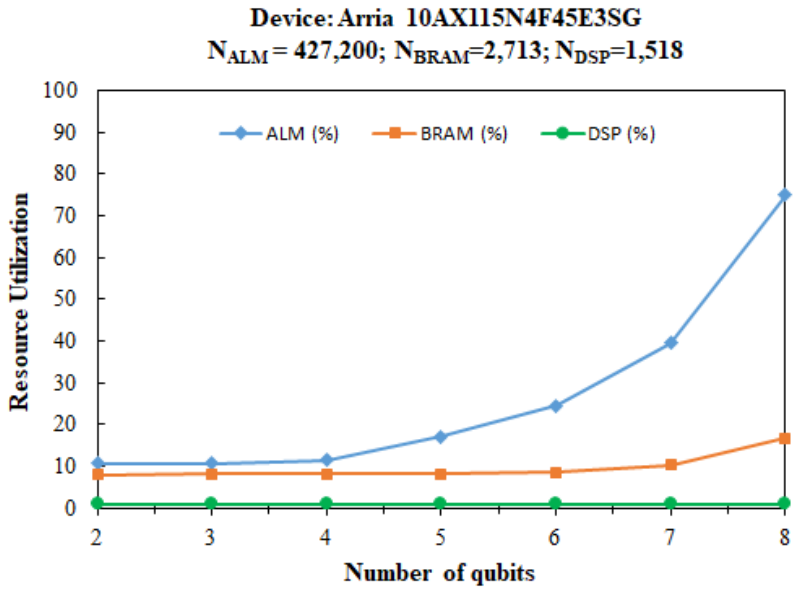


Fig. 50: QFT on-chip resource utilization using N-concurrent-CMAC architecture and lookup.

Table 15: QFT Implementation Results using Dual-sequential CMAC Architecture, On-chip Resources, and Lookup

Number of qubits	OCR* utilization (%)			Emulation time (sec)
	ALMs	BRAMs	DSPs	
2	12.39	8.55	2.11	7.55E-7
3	12.34	8.55	2.11	9.61E-7
4	12.36	8.63	2.11	1.79E-6
5	12.43	8.70	2.11	5.08E-6
6	12.38	8.99	2.11	1.83E-5
7	12.39	10.69	2.11	7.1E-5
8	12.37	17.18	2.11	0.0003
9	12.37	43.54	2.11	0.0011

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518.$

We implemented the third proposed architecture, i.e., dual-sequential-CMAC, in which two sequentially operating CMAC computations are overlapped with data write operations. Table 15 and Fig. 51 show the obtained results. The results are similar to the first architecture implementation in which the ALM utilization increases exponentially while the remaining resource utilization is fixed. In Fig. 52, we compare the emulation time of all three implementations, and we observe that the *N*-concurrent implementation has the highest performance. This is due to the parallel operation of the CMAC units. The trade-off for the *N*-concurrent implementation is area since we were only able to emulate up to 8 qubits, while using the single-CMAC and dual-sequential-CMAC architectures we were able to emulate up to 9 qubits. Any larger circuit exceeds the FPGA on-chip resources allocated for storing the computation vectors and algorithm matrix.

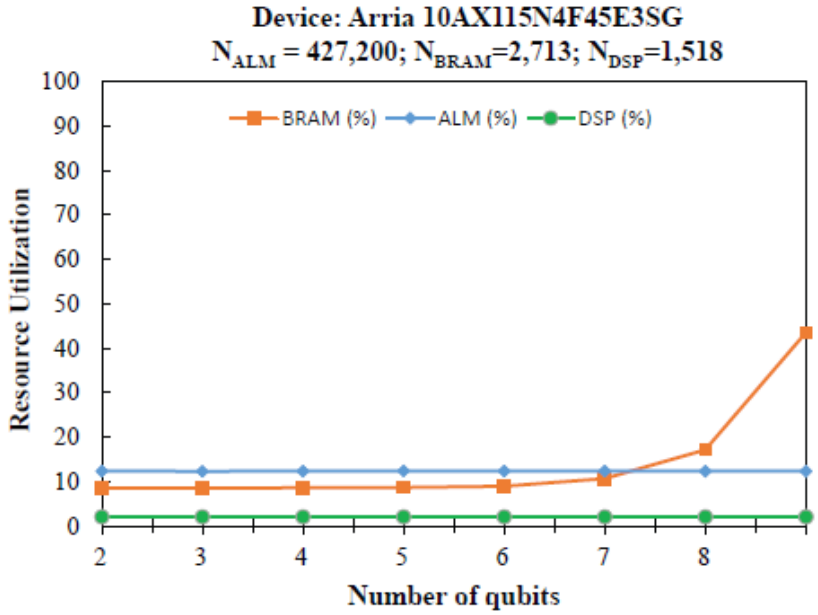


Fig. 51: QFT on-chip resource utilization using dual-sequential-CMAC architecture and lookup.

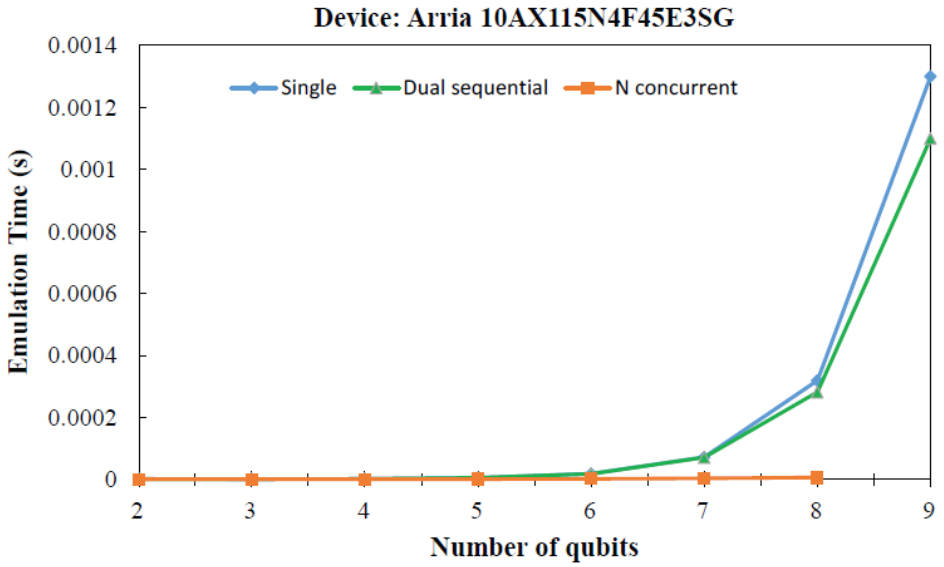


Fig. 52: Comparison of QFT emulation times using CMAC architectures with on-chip memory.

On-board memory (OBM) configurations of the proposed architectures were also implemented to scale the emulation to a higher number of qubits. The storage of state vectors and algorithm matrix is performed using on-board SRAM and on-board SDRAM memories respectively. We implemented this for the single-CMAC and dual-sequential-CMAC architectures running QFT. For the *N*-concurrent-CMAC architecture, an OBM configuration leads to SDRAM read/write contention issues, which significantly degrades the performance, and it was not considered for implementation. Table 16 shows the results from implementation of the single-CMAC architecture with an OBM configuration. The obtained results demonstrate that the on-chip resources are constant with increasing qubits because they are only used for the fixed number of adders/multipliers of the single CMAC unit. Therefore, the scalability limit is determined by the size of the on-board memory, which is being used to store the state vectors and algorithm matrix. Using 1×32 GB SDRAM bank of a single C2 compute node, it was possible to emulate up to 16-qubit QFT, compared to 9-qubit QFT using on-chip resources.

Table 16: QFT Implementation Results using Single-CMAC Architecture, On-board Memory and Lookup.

Number of qubits	On-chip resource* utilization (%)			OBM** Utilization (bytes)		Emulation time (sec)***
	ALMs	BRAM	DSPs	SRAM	SDRAM	
2	10.71	8.44	1.05	32	128	1.7E-6
3	10.71	8.44	1.05	64	512	2.0E-6
4	10.71	8.44	1.05	128	2K	3.9E-6
5	10.71	8.44	1.05	256	8K	1.1E-5
6	10.71	8.44	1.05	512	32K	3.9E-5
7	10.71	8.44	1.05	1K	128K	0.00015
8	10.71	8.44	1.05	2K	512K	0.00061
9	10.71	8.44	1.05	4K	2M	0.00241
10	10.71	8.44	1.05	8K	8M	0.00963
11	10.71	8.44	1.05	16K	32M	0.03851
12	10.71	8.44	1.05	32K	128M	0.15399
13	10.71	8.44	1.05	64K	512M	0.61586
14	10.71	8.44	1.05	128K	2G	2.36324
15	10.71	8.44	1.05	256K	8G	9.853
16	10.71	8.44	1.05	512K	32G	39.4209

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518$.
**Total on-board memory: 4 parallel SRAM banks of 8MB each and 2 parallel SDRAM banks of 32GB each.

We also implemented the dual-sequential-CMAC architecture with OBM configuration, and the results are shown in Table 17. For both OBM configurations, we observe, as expected, that the on-chip resources (OCR) on the FPGA are fixed for emulation of a particular algorithm due to the fixed architecture of the CMAC. Fig. 53 shows the comparison of the emulation times between the two configurations. It can be observed that the dual-sequential-CMAC architecture performs better in terms of emulation time. The time complexity of $O(N^2)$ for single-CMAC and dual-sequential-CMAC, see Table 4, is also reflected in these results. From our experiments, we conclude that the proposed dual-sequential CMAC architecture provides the highest performance in terms of emulation time when compared to other configurations. Integrating on-board memory with that architecture enables us to emulate QFT using 16 fully entangled qubits on a single Arria 10 FPGA node with 32 GB memory, with an emulation time of 18 seconds.

Table 17: QFT Implementation Results using dual-sequential-CMAC Architecture, On-board Memory, and Lookup.

Number of qubits	On-chip resource* utilization (%)			OBM** Utilization (bytes)		Emulation time (sec)***
	ALMs	BRAM	DSPs	SRAM	SDRAM	
2	12	8.63	2.11	32	128	7.55E-7
3	12	8.63	2.11	64	512	9.61E-7
4	12	8.63	2.11	128	2K	1.79E-6
5	12	8.63	2.11	256	8K	5.08E-6
6	12	8.63	2.11	512	32K	1.83E-5
7	12	8.63	2.11	1K	128K	7.10E-5
8	12	8.63	2.11	2K	512K	0.00028
9	12	8.63	2.11	4K	2M	0.00113
10	12	8.63	2.11	8K	8M	0.00451
11	12	8.63	2.11	16K	32M	0.018002
12	12	8.63	2.11	32K	128M	0.072006
13	12	8.63	2.11	64K	512M	0.2888021
14	12	8.63	2.11	128K	2G	1.152083
15	12	8.63	2.11	256K	8G	4.608329
16	12	8.63	2.11	512K	32G	18.4331

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518$.
**Total on-board memory: 4 parallel SRAM banks of 8MB each and 2 parallel SDRAM banks of 32GB each.

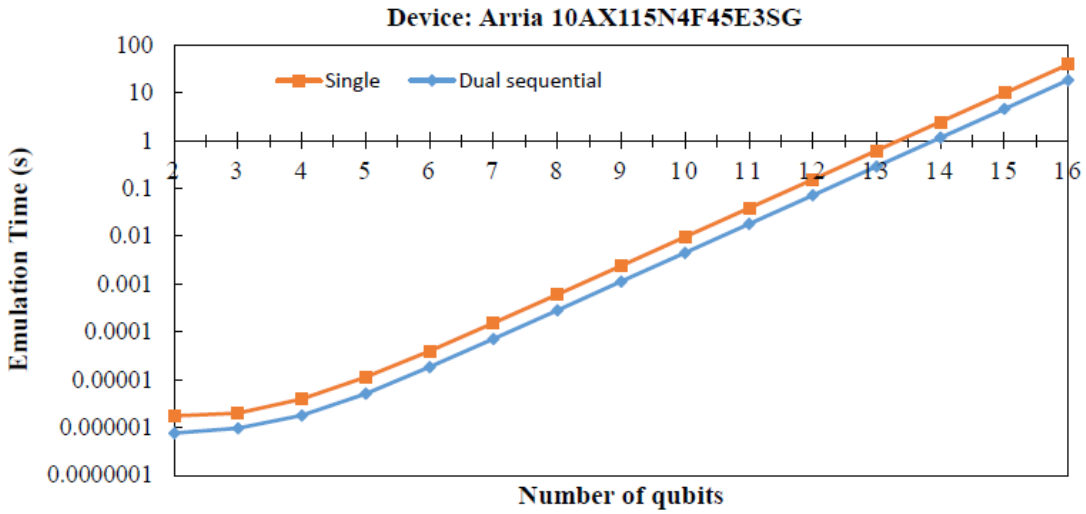


Fig. 53: Comparison of QFT emulation times using CMAC architectures with on-board memory.

Table 18: QFT Implementation Results using Dual-sequential-CMAC Architecture, On-board Memory, and Dynamic Generation.

Number of qubits	**On-chip resource* utilization (%)**			**OBM** Utilization (bytes)**	**Emulation time (sec)*****
	ALMs	**BRAMs**	**DSPs**	**SDRAM**	
2	13.16	9.58	3.23	32	1.99E-6
4	13.16	9.58	3.23	128	3.02E-6
6	13.16	9.58	3.23	512	1.95E-5
8	13.16	9.58	3.23	2K	0.0003
10	13.16	9.58	3.23	8K	0.0045
12	13.16	9.58	3.23	32K	0.0720
14	13.16	9.58	3.23	128K	1.1521
16	13.16	9.58	3.23	512K	18.433
18	13.16	9.58	3.23	2M	294.93
20	13.16	9.58	3.23	8M	4718.934
22	13.16	9.58	3.23	32M	18876†
24	13.16	9.58	3.23	128M	302012†
26	13.16	9.58	3.23	512M	4832188†
28	13.16	9.58	3.23	2G	7.73E+7 †
30	13.16	9.58	3.23	8G	1.23E+9 †
32	13.16	9.58	3.23	32G	1.979E+10 †

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518.$

**Total on-board memory: 4 parallel SRAM banks of 8MB each and 2 parallel SDRAM banks of 32GB each.

†Results are projected using a performance estimation model.

To emulate larger QFT circuits, we perform implementation of the dual-sequential-CMAC architecture with OBM and using the *dynamic generation* technique. QFT results are shown in Table 18. Using the *dynamic generation* technique, the algorithm matrix elements are generated in hardware dynamically, and the SDRAM stores only the input/output state vectors. Therefore, up to 32-qubit emulation of QFT was possible on a single FPGA with 32 GB on-board memory, compared to the maximum of 16 qubits using *lookup*. On-chip resources are slightly higher because of the additional generation hardware units. Although QFT circuits for up to 32 qubits were successfully built on hardware, emulation times were unrealistically large for circuits larger than 20 qubits, and the runtimes for these circuits were estimated using an accurate model derived from (18), (20), and (22) for the proposed pipelined architectures.

Table 19: Grover's Algorithm Implementation Results using Dual-sequential-CMAC Architecture, On-board Memory, and Streaming.

Number of qubits	On-chip resource* utilization (%)			OBM** Utilization (bytes)		Emulation time (sec)***
	ALMs	BRAMs	DSPs	SDRAM		
2	11	8	1	32	2.3E-6	
4	11	8	1	128	3.4E-6	
6	11	8	1	512	2.0E-5	
8	11	8	1	2K	2.8E-4	
10	11	8	1	8K	4.5E-3	
12	11	8	1	32K	7.2E-2	
14	11	8	1	128K	1.15E0	
16	11	8	1	512K	1.84E+1	
18	11	8	1	2M	2.95E+2	
20	11	8	1	8M	4.72E+3	
22	11	8	1	32M	7.5E+4†	
24	11	8	1	128M	1.2E+6†	
26	11	8	1	512M	1.93E+7†	
28	11	8	1	2G	3.09E+8†	
30	11	8	1	8G	4.95E+9†	
32	11	8	1	32G	7.92E+10†	

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518.$
**Total on-board memory: 4 parallel SRAM banks of 8MB each and 2 parallel SDRAM banks of 32GB each.
†Results are projected using a performance estimation model.

Finally, we implement the dual-sequential-CMAC architecture with OBM and use the *data streaming* technique. The algorithm matrix elements are streamed in during computation and only the state vectors require storage. As a case study for this technique, we emulated our proposed multi-pattern Grover's search algorithm, see Fig. 34. The target patterns were set to {1 11 2 13 4 15 6 7}, where each number corresponds to the index of a target state we are searching for. Output results demonstrated high probability amplitudes identifying the target states, and these were verified against results obtained from software simulations in MATLAB. The hardware implementation results are shown in Table 19. For our experimental setup, we utilized 2×32 GB SDRAM banks to store the input and output quantum state vectors respectively, while the input algorithm matrix elements were streamed in. This allowed emulation of a higher number of qubits, i.e., 32. The space complexity of this architecture is $O(1)$, as there are only two operating CMACs. The time complexity is $O(N^2)$ due to the computation of N^2 elements of the algorithm matrix, see

Table 4. Hardware builds of up to 32-qubit circuits for Grover's algorithm were performed on a single FPGA with 32 GB SDRAM memory. Emulation times for circuits larger than 22 qubits were estimated using the performance model derived from (18), (20), and (22).

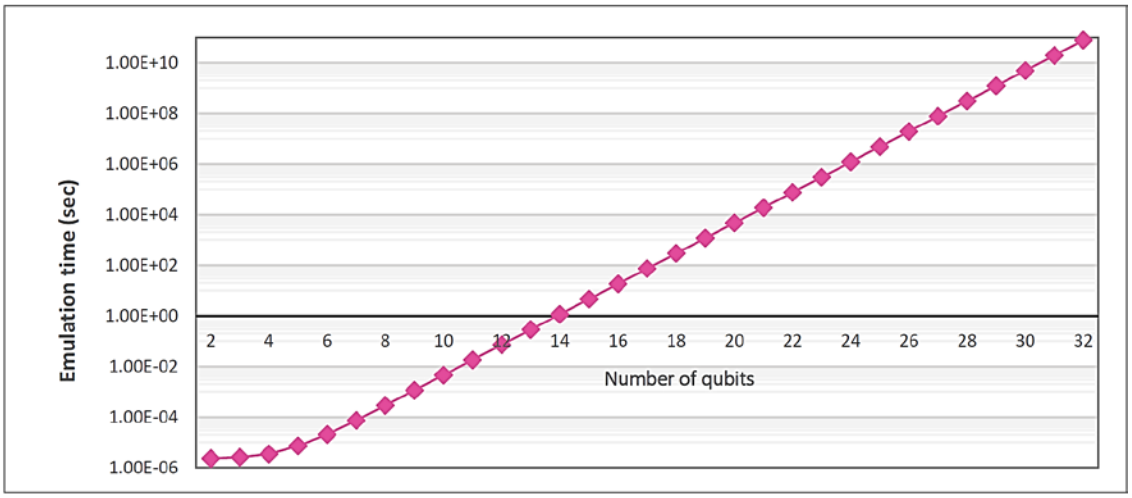


Fig. 54: Grover's search algorithm emulation using dual-sequential-CMAC Architecture, on-board memory, and streaming.

The emulation time as a function of the number of qubits is shown in Fig. 54. A sustained operating frequency of 233 MHz was reported by the hardware compiler for these implementations, indicating very high throughput as a result of a fully pipelined dataflow design. The *streaming* technique does not require any generation hardware and therefore can be used to emulate any quantum algorithm that is reducible to a single unitary transformation. The complexity of the algorithm does not affect the performance of emulation. Therefore, emulation of other algorithms would yield the same results in terms of hardware utilization and emulation time. The reconfigurable architecture of our emulator allows improvement of the time complexity to $O(N)$ by instantiating N parallel instances of CMAC units for vector matrix multiplications. To emulate a larger number of qubits using the single CMAC approach, the amount of on-chip resources, and/or on-board memory would need to be increased. Other approaches include adopting multi-CMAC architectures, and/or using a multi-node architecture where the design is partitioned among

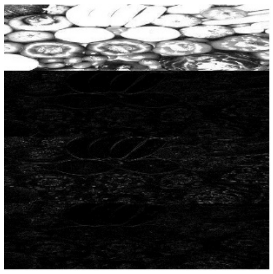
the nodes. In other words, scaling to higher quantum circuit sizes would require using more hardware resources such as on-chip resources (OCR), on-board memory (OBM), number of CMACs, or the number of FPGA nodes.

7.3.3 Implementation of QHT using Kernel-based Emulation

We performed experiments to evaluate our proposed kernel-based emulation, see Fig. 26, using QHT algorithm as a use case. Hardware architectures for emulation of 1D-QHT and 2D-QHT, see Fig. 33, were implemented using C++ on the DS8 programming environment. Input images with resolution of up to 1024×1024, and 256 shades of grayscale pixels, were used to test the designs. MATLAB was used to convert the images into greyscale, generate the input vectors for DS8, and reconstruct images from the output vectors. Synthesis and hardware builds were performed using Quartus Prime Version 17.02 on the DS8 environment. Fig. 55(a) shows one of the input images converted to greyscale, Fig. 55(b) is the output after a 1D-QHT operation with 1 level of decomposition. Fig. 55(c) is the output after a 1D-QHT operation with 2 levels of decomposition and Fig. 55(d) shows the reconstructed images after a 1D-IQHT operation was applied. Figs. 56(a) to 56(d) show the results from repeating the experiment using the 2D-QHT and 2D-IQHT architectures.

(a) Original image

(b) 1-level 1D-QHT

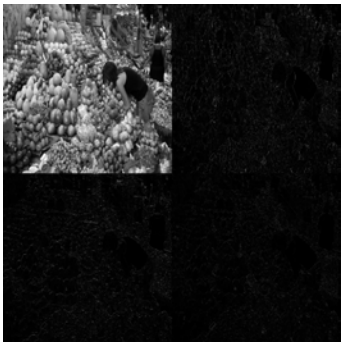


(c) 2-level 1D-QHT

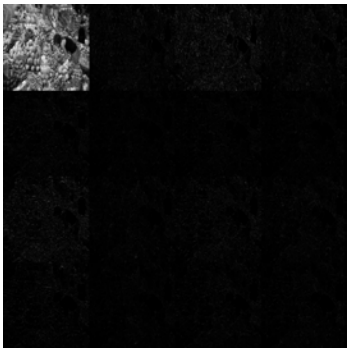
(d) Reconstructed image using 1D-IQHT

Fig. 55: Experimental results of 1D-QHT emulation using kernel-based architectures.

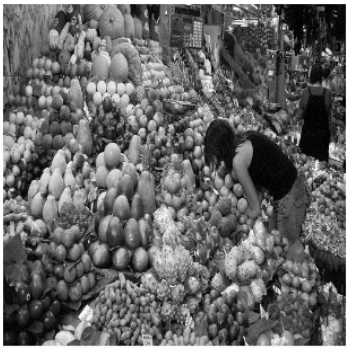
(a) Original image



(b) 1-level 2D-QHT



(c) 2-level 2D-QHT



(d) Reconstructed image using 2D-IQHT

Fig. 56: Experimental results of 2D-QHT emulation using kernel-based architectures

Table 20: 1D-QHT Implementation Results on Arria 10 FPGA

Number of pixels	Number of qubits	Resource Utilization* (%)			SDRAM** (bytes)	Emulation time (sec)
		ALMs	BRAMs	DSPs		
16x16	8	11	8	1	4K	0.00018
32x32	10	11	8	1	16K	0.00071
64x64	12	11	8	1	64K	0.00285
128x128	14	11	8	1	256K	0.01139
256x256	16	11	8	1	1M	0.04557
512x512	18	11	8	1	4M	0.18226
1024x1024	20	11	8	1	16M	0.72905

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518$.
**Total on-board memory: 4 parallel SRAM banks of 8MB each and 2 parallel SDRAM banks of 32GB each.

Table 21: 2D-QHT Implementation Results on Arria 10 FPGA

Number of pixels	Number of qubits	Resource Utilization* (%)			SDRAM** (bytes)	Emulation time (sec)
		ALMs	BRAMs	DSPs		
16x16	8	14	9	2	4K	0.00012
32x32	10	14	9	2	16K	0.00047
64x64	12	14	9	2	64K	0.00187
128x128	14	14	9	2	256K	0.00746
256x256	16	14	9	2	1M	0.02982
512x512	18	14	9	2	4M	0.11926
1024x1024	20	14	9	2	16M	0.47704

*Total on-chip resources: $N_{ALM} = 427{,}2000, N_{BRAM} = 2{,}713, N_{DSP} = 1{,}518$.
**Total on-board memory: 4 parallel SRAM banks of 8MB each and 2 parallel SDRAM banks of 32GB each.

Resource utilizations from the hardware implementations are summarized in Tables 20 and 21 for 1D and 2D respectively. The on-chip resources (ALMs, BRAMs, DSPs) are used up in implementing the static components of the design such as counters, adders, shift operators, etc. and hence are constant as the emulated circuit size (number of qubits) increases. The low on-chip resource utilizations indicate that our proposed approach and emulation architecture designs are highly space-efficient. The 1D-QHT architecture consumes lower on-chip resources than 2D-QHT, due to its less complex kernel operations. The low resource utilizations also indicate the flexibility of the QHT and IQHT designs for integrating with larger algorithms.

The SDRAM memory requirements for storage of the input and output images as quantum state vectors are also reported in Tables 20 and 21. For the highest resolution image of size 1024×1024, the pixels occupy 25% of the total on-board SDRAM memory (64 GB) available on a single DS node. The pixels of the input images are encoded as basis coefficients of a quantum

state. For example, to store 16×16 or 256 pixels, we need 256 complex coefficients each of which have a real and imaginary component occupying total 2×4=8 bytes in 32-bit floating point representation. Therefore, for storing both input and output images, 2×256×8 = 4096 bytes of memory was required. The obtained memory usages for larger QHT circuits are consistent with expected values.

The hardware designs on the FPGA were pipelined to ensure a constant and high operating frequency of 233 MHz. The obtained emulation times for high resolution images are also feasible. For a 1024×1024 image, 20 qubits were sufficient for achieving dimension reduction using 1D and 2D QHT. From our experimental results, we observe that the emulation time increases linearly with increase in the number of image pixels (states), as illustrated by Fig. 57. This is because a large portion of the emulation time is dedicated to writing in and reading out the input/output state vectors of size N (number of pixels) hence the emulation time complexity is $O(N)$. This indicates the benefit of using quantum encoding of data, i.e., encoding each image pixel as a basis state coefficient in the quantum state space. Finally, the emulation times for 1D-QHT are higher than 2D-QHT because of the higher number of iterations $N/2$ in the 1D algorithm, compared to $N/4$ iterations in the 2D algorithm, see Algorithms A1 and A2 in the appendix.

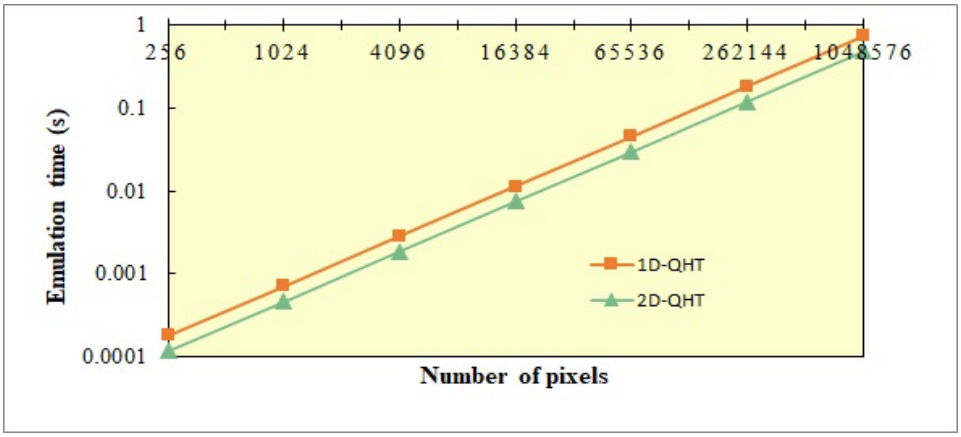


Fig. 57: Emulation time as a function of data size (number of pixels).

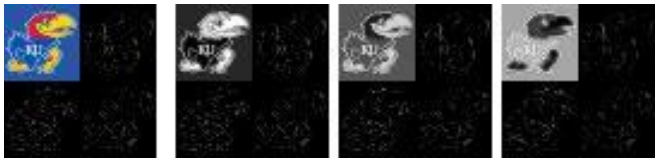
In general, on a classical emulation platform, the emulation execution time increases with both the spatial and temporal complexities of the quantum circuit. In other words, the emulation time of a quantum circuit on a classical platform is generally a function of both the circuit width (number of qubits) and depth (number of gate levels). Due to optimizations and encoding techniques we used, the emulation time of our proposed emulation architectures is a function of only the quantum circuit width (number of qubits), as shown by our experimental results. On state-of-the-art superconducting NISQ devices [14] [71], the execution time is a function of only the depth (number of gate levels) of the circuit [72]. For our proposed 1D-QHT and 2D-QHT circuits, which are simple quantum circuits of depth 1, we estimate an execution time of 0.01ms on a typical NISQ device processing a 7×7 qubit array with sampling frequency of 100 KHz [72]. The estimated execution time is constant for a fixed circuit depth and variable number of qubits in the quantum processing unit (QPU) array, i.e., the time complexity is theoretically $O(1)$. In comparison, the time complexity of our emulation is $O(N)$.

7.3.4 Implementation of QHT using MATLAB and IBM Quantum

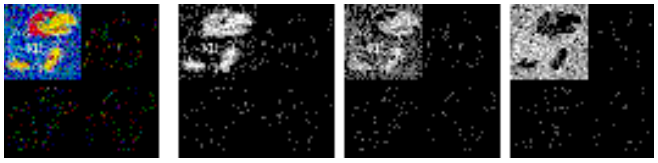
The quantum circuits for sequential and parallel QHT, see Fig. 31, were evaluated on the IBM Quantum system. Simulations were performed using the IBM *qasm* simulator, while real implementations were performed on the 15-qubit real quantum processor, *ibmq_16_melbourne*. For reference, noise-free simulation models of the QHT circuits were also developed and implemented in MATLAB. The test data used were $64 \times 64 \times 3$ RGB-images and high-resolution $1024 \times 1344 \times 33$ multi-spectral images. Zero-padding was used to extend the number of datapoints to powers-of-2 in each dimension for the proper operation of the QHT kernel. Fig. 58(a) shows a $64 \times 64 \times 3$ input image and Figs. 58(b) and (c) show the corresponding output images after 1 level of parallel (1-stage) 3D-QHT packet decomposition performed in MATLAB and IBM

Q simulations, respectively. After 1-level ($l = 1$) 3D-QHT, the dimensions were reduced by a factor of $\frac{1}{2^1} = \frac{1}{2}$, where l is the number of decomposition levels. There was distortion in output images from the IBM Q simulation, due to the statistical noise that was generated during measurement of the output.

(a) Original 64x64 pixel RGB Images

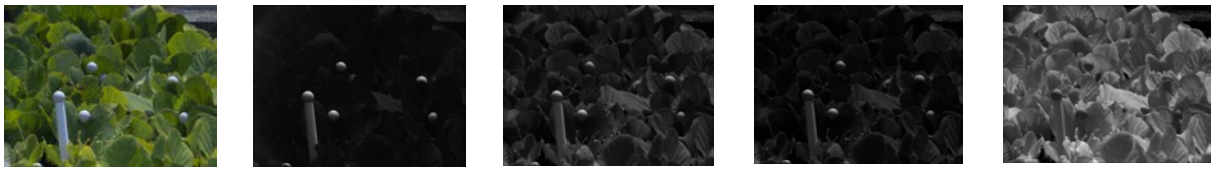


(b) Output images from MATLAB simulations after 1-level 3D-QHT



(c) Output images from IBM Q simulations after 1-level 3D-QHT.

Fig. 58: Test RGB image data and output image results from MATLAB and IBM Q simulations.



Original multi-spectral image in RGB representation

Image band # 01

Image band # 15

Image band # 22

Image band # 33

Image band #01 after 2-level 3D-QHT

Image band #15 after 2-level 3D-QHT

Image band #22 after 2-level 3D-QHT

Image band #33 after 2-level 3D-QHT

Fig. 59: Test multi-spectral images and output images from MATLAB simulations.

Fig. 59 shows a $1024 \times 1344 \times 33$ multi-spectral image in RGB representation, four of its spectral bands, and the corresponding decomposed image bands after 2-level ($l = 2$) 3D-QHT

packet decomposition, where each dimension is reduced by a factor of $\frac{1}{2^2} = \frac{1}{4}$. The experiments were repeated using MATLAB models for sequential (3-stage) 3D-QHT, producing consistent results. It was not possible to implement the circuits for multi-spectral images on IBM Q due to simulator and hardware limitations.

Table 22: Theoretical expectations and experimental results for 14-qubit 3D-QHT using IBM-Q.

Decoherence time = 62.1 usec		Theoretical Expectations			Simulation Results on the 32-qubit IBM QASM simulator (8,000 shots)			Hardware Results on the 15-qubit IBM quantum processor ibmq_16_melbourne (8,000 shots)	
		Run-time (sec)	Fidelity	Circuit Coherence Ratio	Run-time (sec)	Fidelity		Run-time (sec)	Fidelity
		QHT	QHT	QHT	State initialization + QHT	QHT	State initialization + QHT	QHT	QHT
Sequential QHT	Unopt.	1.0494E-04	100.00%	0.59	5.1529E-02	99.96%	72.21%	4.8823E+01	35.50%
	Opt.	2.4683E-05	100.00%	2.52	4.8907E-02	99.98%	72.35%	4.6082E+01	42.65%
	Fidelity improvement	NA	1.0000	NA	NA	1.0002	1.0020	NA	1.2013
	Speedup	4.2515	NA	NA	1.0536	NA	NA	1.0595	NA
Parallel QHT	Unopt.	4.6869E-05	100.00%	1.32	4.9562E-02	99.96%	72.58%	4.8298E+01	36.48%
	Opt.	1.1200E-05	100.00%	5.54	4.5523E-02	99.98%	72.64%	4.5726E+01	55.38%
	Fidelity improvement	NA	1.0000	NA	NA	1.0002	1.0009	NA	1.5182
	Speedup	4.1848	NA	NA	1.0887	NA	NA	1.0563	NA
Fidelity improvement (Parallel Opt. vs. Sequential Unopt.)		NA	1.0000	NA	NA	1.0002	1.0060	NA	1.5600
Speedup (Parallel Opt. vs. Sequential Unopt.)		9.3695	NA	NA	1.1319	NA	NA	1.0677	NA

Theoretical Expectations and Metrics:

Theoretical run-times were estimated, see Table 22, for the proposed QHT circuit variants using real gate times of the *ibmq_16_melbourne* machine. The theoretical run-times in Table 22 refer to expected run-times of the 14-qubit QHT circuits used for the test RGB images. The relative improvement between unoptimized and optimized circuits serves as a reference point to which we can compare the measured experimental run-times. We measured the gate times for SWAP and H gates on the IBM-Q systems to be τ_{SWAP} = 2229.33 ns and τ_H= 53.333 ns respectively, and calculated realistic run-times for each circuit using the time-delay expressions from (36)-(39). The proposed optimizations provide theoretical speedups of 4.2515 and 4.1848 fold for sequential and parallel QHT, respectively, see Table 22. Comparing the optimized parallel with the unoptimized sequential circuit shows a 9.3695 speedup.

$$CCR = \frac{\mathrm{T2}}{t_{total}^{run-time}} \tag{54}$$

We define circuit coherence ratio (CCR) in (54), as a metric to evaluate how coherent a given circuit is by comparing its execution time to the system decoherence time T2. A CCR greater-than-unity corresponds to a coherent circuit, while a CCR less-than-unity corresponds to a decoherent circuit. The CCR is calculated for each of the proposed circuit variants, see Table 22. CCR for the unoptimized sequential circuit is less-than-unity which indicates that the circuit violates the decoherence time constraint. CCR for the optimized sequential circuit is greater-than-unity which indicates that the circuit execution time is within the decoherence time constraint. Thus, the proposed optimizations are favorable for improving coherence of the sequential QHT circuits. The optimizations for parallel QHT also significantly improve the respective CCRs from 1.32 to 5.54, see Table 22. We also verified the correctness and evaluated the accuracy of each circuit by

measuring the state fidelity as defined in (53). By comparing fidelities among the circuit variants, see Table 22, we determined how effective the optimizations were in reducing circuit depth and improving coherence and state-fidelity.

Table 22 also shows the simulation run-times and fidelities obtained for each of the implemented circuit variants. Run-times were measured for QHT circuits with qubit state-initialization (using image data) which resulted in very deep circuits. The additional time-delay (overhead) of state-initialization circuit is much larger than the actual QHT circuit execution time. This results in lower speedups for simulation compared to theoretical speedups that only take into account the QHT circuit execution time.

The circuit output measurements were obtained on a 14-bit classical register using multiple shots (samples) to minimize the statistical noise of measurements. The state fidelities were measured from 8000-shot simulations. For QHT circuits without state-initialization, the fidelities were above 99%. However, the circuit fidelities decreased because of the additional circuit required for state-initialization with the image data which introduced more noise to the measured results. Comparing the sequential (unoptimized) with parallel (optimized), the fidelity improved from 72.21% to 72.35% for sequential QHT, and from 72.58% to 72.64% for parallel QHT.

Hardware implementations were also performed on the *ibmq_16_melbourne* quantum processor and the obtained run-times and fidelities are shown in Table 22. Qubit state-initialization with image data could not be implemented, as the resulting circuits were too large, and run-times exceeded the device repetition and readout rate. The hardware run-times are in the range of seconds, compared to the simulation run-times which were in milliseconds. This is due to the unavoidable configuration overhead of the quantum processor, i.e., the time taken to generate control pulses of the quantum gates, which is much larger than the actual circuit execution time.

The fidelities measured from hardware executions are also shown in Table 22. Due to high sampling noise of the actual quantum hardware, the fidelities are lower than 55%. However, the fidelities improve as the circuits become optimized, see Table 22 For further improving the fidelities, quantum error correction is required before sampling the quantum circuit and forming the probability distribution data. Given the current status of the technology/tools, it's not possible to isolate the different types of run-time overhead, i.e., state-initialization overhead and hardware setup/configuration overhead, in experimental studies. The simulation and hardware run-times could consequently be incomparable. However, both experiments are useful to evaluate the effect of optimizations on relative run-times for each experiment. Therefore, in our results we have included the analysis of theoretical, simulation, and hardware experiments.

7.4 Evaluation of Quantum Pattern Recognition

We implemented the emulation architectures for the proposed quantum system for pattern recognition based on dimension reduction, see Fig. 39. The kernel-based emulation was used for multi-dimensional QHT emulation and CMAC-based emulation was employed for performing multi-pattern QGS. 32-bit floating-point precision was used to represent the real and imaginary components of the complex state coefficients for both emulators, and the architectures were fully pipelined for highest throughput. High resolution single-band and multi-spectral images were used as test data sets for the experiments. We have obtained implementation results emulating up to 32 qubits on a single FPGA node, with an operating frequency of 233 MHz.

The experimental results for the single-band images are presented in Table 23. In this experiment, multi-level 2D-QHT dimension reduction and pattern search using QGS was performed on single-band grayscale images of up to 64K×64K pixel size and using up to 32 emulated qubits, see Table 23. Fig. 60(a) shows an example single-band grayscale image, Fig.

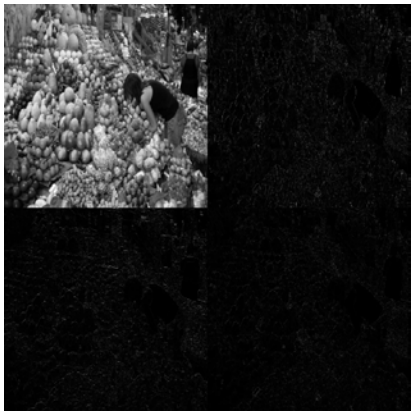
60(b) shows the reduced image after 1 level of 2D-QHT decomposition, and Fig. 60(c) shows the reduced image with pattern indices identifying a person in it with the help of QGS. A 10-qubit QGS circuit was emulated to perform pattern search on the reduced image data and output the pattern indices.

Table 23: Quantum Pattern Recognition Implementation Results using Single-spectral Images on Arria 10 FPGA.

No. of pixels	No. of qubits	No. of levels	OCR* utilization (%)			OBM** (bytes)	Emulation time (sec)***
			ALMs	BRAMs	DSPs		
128x128	14	3	22	16	2	128K	1.15E0
256x256	16	4	22	16	2	512K	1.84E01
512x512	18	5	22	16	2	2M	2.95E02
1024x1024	20	6	22	16	2	8M	4.72E03
2048x2048	22	7	22	16	2	32M	7.5E04
4096x4096	24	8	22	16	2	128M	1.2E06
8192x8192	26	9	22	16	2	512M	1.93E07
16Kx16K	28	10	22	16	2	2G	3.09E08
32Kx32K	30	11	22	16	2	8G	4.95E09
64Kx64K	32	12	22	16	2	32G	7.92E10

*Total on-chip resources: N_{ALM} = 427,000, N_{BRAM} = 2,713, N_{DSP} = 1,518
**Total on-board memory: 2x32 GB SDRAM banks
***Operating frequency: 233 MHz

(a) Original image



(b) 1-level 2D-QHT

(c) Pattern identified in reduced image using QGS

Fig. 60: Experimental results of 2D-QHT decomposition and QGS pattern recognition.

The FPGA resource utilization data shown in Table 23 refers to both the QHT and QGS circuit utilizations. The number of decomposition levels for 2D-QHT is increased as the input image size increases. This is done to keep the size of the reduced image to a fixed resolution, and therefore the QGS circuit can perform pattern search using a fixed number of qubits. The on-chip resources (ALMs, BRAMs, DSPs) are used for implementing the static components of the design such as counters, adders, shift operators, etc. and hence are constant as the emulated circuit size (number of qubits) increases. The low on-chip resource utilizations indicate that our proposed emulation architecture designs are highly space-efficient and highly scalable. The on-board memory is used to store the coefficients of the input and output quantum states and therefore the memory utilization increases exponentially with the number of qubits, making the emulation highly memory bound. The highest resolution image of size 64K×64K occupies a full 32 GB SDRAM bank. The image pixels are encoded as the quantum state coefficients which have 32-bit real and imaginary components and occupy 8 bytes each. A 64K×64K image contains 2^{32} pixels and so the total memory required to encode it is $2^{32} \times 8$ bytes, or 32 GB. The two 32 GB SDRAM banks on the FPGA node are utilized to store the input and output images respectively.

The system emulation time obtained in Table 23 is a function of the emulation times for QHT and QGS. In previous experiments kernel-based emulation of QHT, our findings show that execution time of the kernel-based emulator increases linearly with the data size, i.e., number of states. As emulation times for QGS is the same due to a fixed circuit size, the overall system emulation time increases linearly with the number of states, *N*, as illustrated in Fig. 61.

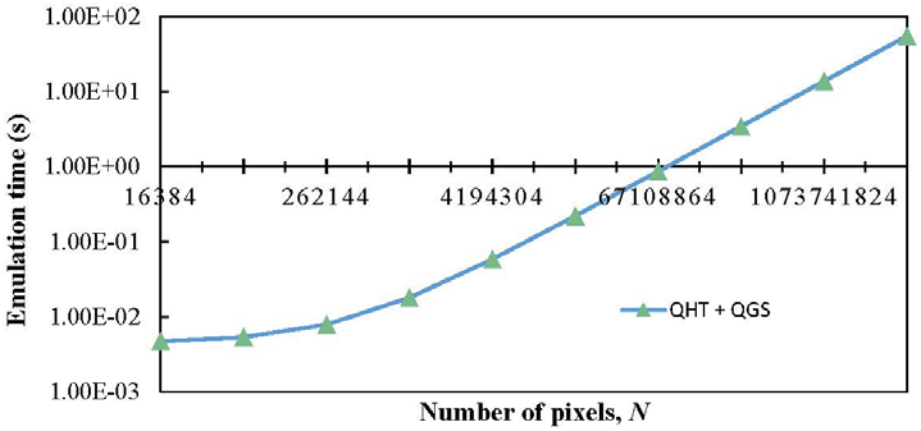


Fig. 61: System emulation time as a function of data size.

7.5 Evaluation of Quantum-to-Classical Data Decoding

To evaluate the proposed and related Q2C data decoding methods, we conducted experiments on IBM Quantum using Qiskit for implementing the proposed quantum circuits. Simulations of the developed circuits were performed using QASM simulator. The number of circuit samples or shots for the experiments ranged from 1,024 to 16,384. Experimental evaluations of our proposed QHT-based method and the QFT-based approach reported in [16] were performed. The methods were evaluated in terms of overhead incurred and time efficiency.

7.5.1 Characterizing measurement (circuit sampling) time on IBM QASM

We characterized the circuit sampling time on the IBM QASM simulator as a function of number of qubits and number of shots. Measurement gates were applied across qubits that are initialized in their ground state and the number of qubits and shots were varied. The obtained execution times of the measurement gates (circuit sampling times) from the simulator are shown in Table 24 and Fig. 62. The measurement time increases linearly with the number of qubits for varying number of shots, as observed in Fig. 62. Based on the linear behavior, the measurement times for odd numbers of qubits were linearly interpolated from the datasets shown in Table 24 and Fig. 62 and used in the overhead analysis of the proposed Q2C method.

Table 24: Measurement timing data on IBM QASM simulator.

Execution Time (ms)					
Number of qubits	Number of shots				
	1024	2048	4096	8192	16384
2	2.72	4.69	9.11	17.74	35.14
4	3.35	6.64	12.86	27.32	51.76
6	4.62	8.79	18.04	33.92	69.37
8	6.11	10.86	20.71	40.83	84.29
10	6.42	13.03	25.34	49.22	101.50
12	7.61	14.75	29.55	58.37	117.58
14	8.73	19.19	33.64	69.03	132.93
16	9.78	19.05	38.17	77.40	155.60
18	11.85	22.25	43.35	87.40	180.94
20	12.42	24.16	49.62	96.05	191.52
22	13.94	27.01	53.84	106.88	214.16
24	15.06	29.65	61.77	119.33	239.15
26	17.79	33.46	66.14	130.73	259.08
28	19.05	35.30	71.14	141.06	281.05
30	24.89	42.36	84.88	157.09	283.11
32	28.40	46.53	90.30	167.50	327.12

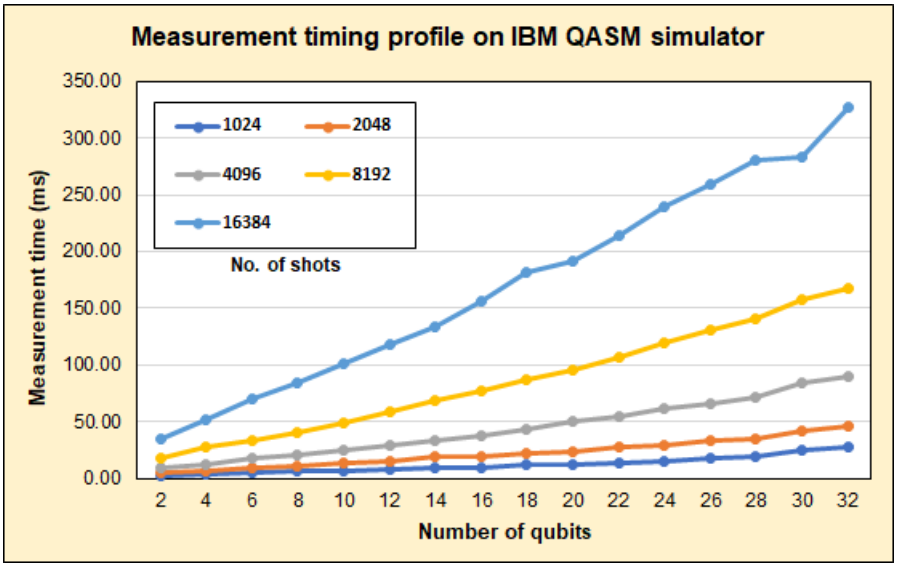


Fig. 62: Measurement time as a function of number of qubits and number of shots on IBM QASM Simulator.

7.5.2 Simulation of QFT-based Q2C

The QFT-based method for Q2C was evaluated by simulating *n*-qubit QFT circuits. The number of qubits, *n*, was varied from 2 to 28 and the number of shots was varied from 1,024 to 16,384, see Table 25. Larger circuit simulations could not be performed due to simulator memory

limitations. The obtained results were consistent with our theoretical expectations. The execution time increases exponentially with the number of qubits and this behavior is consistent for higher number of shots. The experimental data for QFT will be used for quantitative comparison with our proposed QHT-based Q2C method.

Table 25: Quantum Fourier Transform execution times on IBM QASM simulator

Execution Time (ms)						
Number of qubits	Circuit depth	Number of shots				
		1024	2048	4096	8192	16384
2	5	6.75	9.51	17.42	24.56	39.15
4	12	7.93	12.91	16.87	29.45	49.66
6	23	11.15	15.27	25.79	32.85	64.02
8	38	12.03	21.33	27.28	42.04	70.25
10	57	15.95	19.64	25.97	47.02	93.53
12	80	8.04	20.73	27.91	58.09	106.68
14	107	20.71	35.04	41.41	61.37	126.11
16	138	372.02	392.40	410.51	458.01	558.54
18	173	446.00	521.30	533.46	479.67	577.21
20	212	710.94	819.74	827.47	764.68	839.81
22	255	1434.82	1495.12	1550.84	1482.87	1694.57
24	302	4069.67	4234.29	4263.60	4218.60	4265.49
26	353	15794.70	15683.52	15647.70	14368.76	14675.60
28	408	49883.76	64556.78	65938.46	65019.31	70521.61
30	No data collected due to simulator limitations					
32						

7.5.3 Simulation of QHT-based Q2C

We evaluated our proposed QHT-based method of Q2C by simulating multi-level packet and pyramidal decomposable, 2D and 3D-QHT circuits, varying the number of qubits, *n* from 4 to 32. The number of packet/pyramidal decomposition levels was varied from l to $l_{max}^{pkt}/l_{max}^{pyr}$ respectively, see (34), and we obtained circuit depth measurements and circuit execution times. All data was collected for 16,384 shot simulations. The multi-level QHT circuits were highly optimized, resulting in significantly lower circuit depths compared to QFT which is consistent with our theoretical expectations, see Table 26. Moreover, the simplistic nature of quantum gates in the QHT circuit such as SWAP gates, as compared to controlled phase shift gates in the QFT [11], should theoretically incur lower execution time.

Table 26: Multi-level pyramidal decomposable 3D Quantum Haar Transform circuit depths compared to QFT circuit depths.

Number of qubits	Circuit depth									
	QFT	3D-QHT								
		1-level	2-level	3-level	4-level	5-level	6-level	7-level	8-level	9-level	10-level
2	5										
4	12	3									
6	23	7	12								
8	38	11	20								
10	57	15	28	37							
12	80	19	36	49	59						
14	107	22	42	58	71						
16	138	24	46	64	79	91					
18	173	26	50	70	87	101	112				
20	212	28	54	76	95	111	124				
22	255	30	58	82	103	121	136	148			
24	302	31	59	85	107	126	142	155	165		
26	353	31	59	84	106	126	142	155	165		
28	408	31	59	84	106	125	141	155	165	172	
30	No data collected due to simulator limitations	31	59	84	106	125	141	154	164	172	176
32		31	59	84	106	125	141	154	164	171	175

Table 27 presents the execution timing data obtained from multi-level packet decomposable 2D-QHT simulations on IBM Quantum. For comparison, also presented in Table 27 are the *n*-qubit measurement timing data, which is the execution time of only measurement gates (without QFT or QHT) obtained from Table 24, and *n*-qubit QFT circuit execution timing data from Table 25. For every l^{th}-level 2D-QHT decomposition, $l = 1,2, \dots, l_{max}^{pkt}$, the QHT circuit execution times, the reduced number of qubits *k*, and the corresponding *k*-qubit measurement times are also shown in Table 27. From this data we calculate the total time for l^{th}-level QHT as the sum of the 2D-QHT circuit execution time and the corresponding *k*-qubit measurement time. In Table 27 we also present the speedup of QHT-based total time relative to general *n*-qubit measurement time (without QFT or QHT), calculated as shown in (55). Table 28 contains the same data collected from simulation of multi-level packet decomposable 3D-QHT.

$$\text{Speedup} = \frac{t_{measure}(n)}{t_{measure}(k) + t_{exec}^{QHT}} = \frac{t_{measure}(n)}{t_{total}^{QHT}(n, l)} \tag{55}$$

Table 27: Multi-level packet decomposable 2D-QHT execution times compared to QFT simulation times on IBM QASM simulator.

shots = 16,384		QFT	2D-QHT																												
			1-level					2-level					3-level					4-level					5-level								
Number of qubits, n	n-qubit $t_{measure}$ (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup
4	51.76	49.66	2	0.33	35.14	35.47	1.46																						
6	69.37	64.02	4	0.27	51.76	52.03	1.33	2	0.53	35.14	35.67	1.94																	
8	84.29	70.25	6	0.32	69.37	69.69	1.21	4	0.35	51.76	52.11	1.62	2	0.42	35.14	35.56	2.37												
10	101.50	93.53	8	0.30	84.29	84.59	1.20	6	0.38	69.37	69.75	1.46	4	0.47	51.76	52.23	1.94	2	0.57	35.14	35.71	2.84							
12	117.58	106.68	10	0.32	101.50	101.82	1.15	8	0.41	84.29	84.70	1.39	6	0.50	69.37	69.87	1.68	4	0.64	51.76	52.40	2.24	2	0.76	35.14	35.90	3.28
14	132.93	126.11	12	0.38	117.58	117.96	1.13	10	0.42	101.50	101.92	1.30	8	0.53	84.29	84.82	1.57	6	0.69	69.37	70.06	1.90	4	0.78	51.76	52.54	2.53
16	155.60	558.54	14	0.35	132.93	133.28	1.17	12	0.44	117.58	118.02	1.32	10	0.55	101.50	102.05	1.52	8	0.90	84.29	85.19	1.83	6	0.82	69.37	70.19	2.22
18	180.94	577.21	16	0.53	155.60	156.13	1.16	14	0.46	132.93	133.39	1.36	12	0.61	117.58	118.19	1.53	10	0.88	101.50	102.38	1.77	8	0.99	84.29	85.28	2.12
20	191.52	839.81	18	0.39	180.94	181.33	1.06	16	0.50	155.60	156.10	1.23	14	0.65	132.93	133.58	1.43	12	0.84	117.58	118.42	1.62	10	0.94	101.50	102.44	1.87
22	214.16	1694.57	20	0.41	191.52	191.93	1.12	18	0.57	180.94	181.51	1.18	16	0.80	155.60	156.40	1.37	14	0.99	132.93	133.92	1.60	12	0.99	117.58	118.57	1.81
24	239.15	4265.49	22	0.46	214.16	214.62	1.11	20	0.67	191.52	192.19	1.24	18	0.93	180.94	181.87	1.31	16	0.89	155.60	156.49	1.53	14	1.12	132.93	134.05	1.78
26	259.08	14675.60	24	0.52	239.15	239.67	1.08	22	0.69	214.16	214.85	1.21	20	0.94	191.52	192.46	1.35	18	0.97	180.94	181.91	1.42	16	1.25	155.60	156.85	1.65
28	281.05	70521.61	26	0.46	259.08	259.54	1.08	24	0.72	239.15	239.87	1.17	22	1.03	214.16	215.19	1.31	20	1.02	191.52	192.54	1.46	18	1.27	180.94	182.21	1.54
30	283.11	No data collected due to simulator limitations	28	0.49	281.05	281.54	1.01	26	0.79	259.08	259.87	1.09	24	1.07	239.15	240.22	1.18	22	1.20	214.16	215.36	1.31	20	1.38	191.52	192.90	1.47
32	327.12		30	0.63	283.11	283.74	1.15	28	0.85	281.05	281.90	1.16	26	1.11	259.08	260.19	1.26	24	1.12	239.15	240.27	1.36	22	1.34	214.16	215.50	1.52

shots = 16,384		QFT	2D-QHT																												
			1-level					2-level					3-level					4-level					5-level								
Number of qubits, n	n-qubit $t_{measure}$ (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup
14	132.93	126.11	2	0.85	35.14	35.99	3.69																						
16	155.60	558.54	4	1.03	51.76	52.79	2.95	2	0.98	35.14	36.12	4.31																	
18	180.94	577.21	6	0.98	69.37	70.35	2.57	4	1.16	51.76	52.92	3.42	2	1.34	35.14	36.48	4.96												
20	191.52	839.81	8	1.12	84.29	85.41	2.24	6	1.27	69.37	70.64	2.71	4	1.51	51.76	53.27	3.60	2	1.54	35.14	36.68	5.22							
22	214.16	1694.57	10	1.37	101.50	102.87	2.08	8	1.43	84.29	85.72	2.50	6	1.52	69.37	70.89	3.02	4	1.65	51.76	53.41	4.01	2	1.73	35.14	36.87	5.81
24	239.15	4265.49	12	1.43	117.58	119.01	2.01	10	1.31	101.50	102.81	2.33	8	1.62	84.29	85.91	2.78	6	1.74	69.37	71.11	3.36	4	1.63	51.76	53.39	4.48
26	259.08	14675.60	14	1.38	132.93	134.31	1.93	12	1.37	117.58	118.95	2.18	10	1.61	101.50	103.11	2.51	8	1.75	84.29	86.04	3.01	6	2.02	69.37	71.39	3.63
28	281.05	70521.61	16	1.50	155.60	157.10	1.79	14	1.59	132.93	134.52	2.09	12	1.86	117.58	119.44	2.35	10	1.74	101.50	103.24	2.72	8	2.06	84.29	86.35	3.25
30	283.11	No data collected due to simulator limitations	18	1.55	180.94	182.49	1.55	16	1.80	155.60	157.40	1.80	14	1.99	132.93	134.92	2.10	12	2.11	117.58	119.69	2.37	10	2.27	101.50	103.77	2.73
32	327.12		20	1.43	191.52	192.95	1.70	18	1.56	180.94	182.50	1.79	16	1.71	155.60	157.31	2.08	14	2.23	132.93	135.16	2.42	12	2.29	117.58	119.87	2.73

Table 28: Multi-level packet decomposable 3D-QHT execution times compared to QFT simulation times on IBM QASM simulator.

shots = 16,384		QFT	3D-QHT																												
			1-level					2-level					3-level					4-level					5-level								
Number of qubits, n	n-qubit measurement time (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup
4	51.76	49.66	1	0.27	17.57	17.84	2.90																						
6	69.37	64.02	3	0.32	43.45	43.77	1.58																						
8	84.29	70.25	5	0.30	60.57	60.87	1.38	2	0.28	35.14	35.42	2.38																	
10	101.50	93.53	7	0.25	76.83	77.08	1.32	4	0.30	51.76	52.06	1.95	1	0.35	17.57	17.92	5.66												
12	117.58	106.68	9	0.29	92.90	93.19	1.26	6	0.37	69.37	69.74	1.69	3	0.41	43.45	43.86	2.68												
14	132.93	126.11	11	0.28	109.54	109.82	1.21	8	0.40	84.29	84.69	1.57	5	0.36	60.57	60.92	2.18	2	0.50	35.14	35.64	3.73							
16	155.60	558.54	13	0.28	125.26	125.54	1.24	10	0.41	101.50	101.91	1.53	7	0.49	76.83	77.32	2.01	4	0.57	51.76	52.33	2.97	1	0.63	17.57	18.20	8.55
18	180.94	577.21	15	0.30	144.27	144.57	1.25	12	0.38	117.58	117.96	1.53	9	0.45	92.90	93.34	1.94	6	0.64	69.37	70.01	2.58	3	0.67	43.45	44.12	4.10
20	191.52	839.81	17	0.32	168.27	168.59	1.14	14	0.37	132.93	133.30	1.44	11	0.52	109.54	110.06	1.74	8	0.64	84.29	84.93	2.26	5	0.68	60.57	61.25	3.13
22	214.16	1694.57	19	0.35	186.23	186.58	1.15	16	0.46	155.60	156.06	1.37	13	0.58	125.26	125.83	1.70	10	0.67	101.50	102.17	2.10	7	0.75	76.83	77.58	2.76
24	239.15	4265.49	21	0.45	202.84	203.29	1.18	18	0.44	180.94	181.38	1.32	15	0.58	144.27	144.84	1.65	12	0.69	117.58	118.27	2.02	9	0.73	92.90	93.63	2.55
26	259.08	14675.60	23	0.34	226.66	226.99	1.14	20	0.56	191.52	192.08	1.35	17	0.61	168.27	168.88	1.53	14	0.71	132.93	133.64	1.94	11	0.81	109.54	110.35	2.35
28	281.05	70521.61	25	0.33	249.12	249.45	1.13	22	0.52	214.16	214.68	1.31	19	0.71	186.23	186.94	1.50	16	0.71	155.60	156.31	1.80	13	0.84	125.26	126.09	2.23
30	283.11	No data collected due to simulator limitations	27	0.44	270.07	270.51	1.05	24	0.55	239.15	239.70	1.18	21	0.68	202.84	203.52	1.39	18	0.73	180.94	181.67	1.56	15	0.94	144.27	145.21	1.95
32	327.12		29	0.41	282.08	282.49	1.16	26	0.56	259.08	259.64	1.26	23	0.59	226.66	227.24	1.44	20	0.81	191.52	192.33	1.70	17	0.96	168.27	169.23	1.93

shots = 16,384		QFT	3D-QHT																												
			6-level					7-level					8-level					9-level					10-level								
Number of qubits, n	n-qubit measurement time (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup	k	t_{exec} (ms)	k-qubit measurement time (ms)	Total time (ms)	Speedup
20	191.52	839.81	2	0.73	35.14	35.87	5.34																						
22	214.16	1694.57	4	0.86	51.76	52.62	4.07	1	1.01	17.57	18.58	11.52																	
24	239.15	4265.49	6	0.86	69.37	70.23	3.41	3	1.07	43.45	44.52	5.37																	
26	259.08	14675.60	8	1.02	84.29	85.31	3.04	5	1.10	60.57	61.66	4.20	2	1.12	35.14	36.26	7.15												
28	281.05	70521.61	10	1.31	101.50	102.81	2.73	7	1.19	76.83	78.02	3.60	4	1.09	51.76	52.85	5.32	1	1.33	17.57	18.90	14.87							
30	283.11	No data collected due to simulator limitations	12	0.91	117.58	118.49	2.39	9	1.17	92.90	94.06	3.01	6	1.21	69.37	70.58	4.01	3	1.34	43.45	44.79	6.32							
32	327.12		14	1.22	132.93	134.15	2.44	11	1.14	109.54	110.68	2.96	8	1.34	84.29	85.63	3.82	5	1.50	60.57	62.07	5.27	2	1.54	35.14	36.68	8.92

Table 29: Multi-level pyramidal decomposable 2D-QHT execution times compared to QFT simulation times on IBM QASM simulator.

shots = 16,384		QFT	2D-QHT																								
			1-level					2-level					3-level					4-level					5-level				
Number of qubits, n	n-qubit $t_{measure}$ (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup
4	51.76	49.66	2	0.84	35.14	35.98	1.44																				
6	69.37	64.02	4	0.65	51.76	52.41	1.32	2	0.85	35.14	35.99	1.93															
8	84.29	70.25	6	0.71	69.37	70.08	1.20	4	0.93	51.76	52.69	1.60	2	0.91	35.14	36.05	2.34										
10	101.50	93.53	8	0.84	84.29	85.13	1.19	6	1.17	69.37	70.54	1.44	4	1.10	51.76	52.86	1.92	2	1.37	35.14	36.51	2.78					
12	117.58	106.68	10	1.13	101.50	102.63	1.15	8	0.99	84.29	85.28	1.38	6	1.26	69.37	70.63	1.66	4	1.70	51.76	53.46	2.20	2	1.55	35.14	36.69	3.20
14	132.93	126.11	12	0.36	117.58	117.94	1.13	10	0.96	101.50	102.46	1.30	8	1.10	84.29	85.39	1.56	6	1.61	69.37	70.98	1.87	4	2.34	51.76	54.10	2.46
16	155.60	558.54	14	0.31	132.93	133.24	1.17	12	1.62	117.58	119.20	1.31	10	1.25	101.50	102.75	1.51	8	1.78	84.29	86.07	1.81	6	1.85	69.37	71.22	2.18
18	180.94	577.21	16	0.31	155.60	155.91	1.16	14	1.41	132.93	134.34	1.35	12	1.72	117.58	119.30	1.52	10	1.73	101.50	103.23	1.75	8	0.69	84.29	84.98	2.13
20	191.52	839.81	18	1.15	180.94	182.09	1.05	16	1.36	155.60	156.96	1.22	14	1.83	132.93	134.76	1.42	12	1.83	117.58	119.41	1.60	10	1.80	101.50	103.30	1.85
22	214.16	1694.57	20	0.74	191.52	192.26	1.11	18	1.54	180.94	182.48	1.17	16	1.89	155.60	157.49	1.36	14	2.09	132.93	135.02	1.59	12	2.36	117.58	119.94	1.79
24	239.15	4265.49	22	1.05	214.16	215.21	1.11	20	1.73	191.52	193.25	1.24	18	1.96	180.94	182.90	1.31	16	2.20	155.60	157.80	1.52	14	2.21	132.93	135.14	1.77
26	259.08	14675.60	24	1.35	239.15	240.50	1.08	22	1.05	214.16	215.21	1.20	20	1.95	191.52	193.47	1.34	18	3.12	180.94	184.06	1.41	16	2.46	155.60	158.06	1.64
28	281.05	70521.61	26	0.56	259.08	259.64	1.08	24	1.62	239.15	240.77	1.17	22	2.01	214.16	216.17	1.30	20	2.73	191.52	194.25	1.45	18	2.88	180.94	183.82	1.53
30	283.11	No data collected due to simulator limitations	28	0.52	281.05	281.57	1.01	26	1.66	259.08	260.74	1.09	24	2.42	239.15	241.57	1.17	22	2.89	214.16	217.05	1.30	20	2.12	191.52	193.64	1.46
32	327.12		30	0.98	283.11	284.09	1.15	28	1.90	281.05	282.95	1.16	26	2.46	259.08	261.54	1.25	24	3.00	239.15	242.15	1.35	22	3.17	214.16	217.33	1.51

shots = 16,384		QFT	2D-QHT																								
			1-level					2-level					3-level					4-level					5-level				
Number of qubits, n	n-qubit $t_{measure}$ (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup
14	132.93	126.11	2	1.78	35.14	36.92	3.60																				
16	155.60	558.54	4	1.92	51.76	53.68	2.90	2	2.35	35.14	37.49	4.15															
18	180.94	577.21	6	2.33	69.37	71.70	2.52	4	1.35	51.76	53.11	3.41	2	2.68	35.14	37.82	4.78										
20	191.52	839.81	8	2.56	84.29	86.85	2.21	6	0.85	69.37	70.22	2.73	4	3.21	51.76	54.97	3.48	2	3.11	35.14	38.25	5.01					
22	214.16	1694.57	10	2.24	101.50	103.74	2.06	8	0.89	84.29	85.18	2.51	6	3.26	69.37	72.63	2.95	4	3.66	51.76	55.42	3.86	2	3.71	35.14	38.85	5.51
24	239.15	4265.49	12	2.67	117.58	120.25	1.99	10	0.94	101.50	102.44	2.33	8	3.59	84.29	87.88	2.72	6	3.73	69.37	73.10	3.27	4	2.61	51.76	54.37	4.40
26	259.08	14675.60	14	2.90	132.93	135.83	1.91	12	3.28	117.58	120.86	2.14	10	3.67	101.50	105.17	2.46	8	3.94	84.29	88.23	2.94	6	1.84	69.37	71.21	3.64
28	281.05	70521.61	16	3.66	155.60	159.26	1.76	14	3.63	132.93	136.56	2.06	12	3.55	117.58	121.13	2.32	10	4.37	101.50	105.87	2.65	8	4.95	84.29	89.24	3.15
30	283.11	No data collected due to simulator limitations	18	3.29	180.94	184.23	1.54	16	4.24	155.60	159.84	1.77	14	4.39	132.93	137.32	2.06	12	4.47	117.58	122.05	2.32	10	4.87	101.50	106.37	2.66
32	327.12		20	3.79	191.52	195.31	1.67	18	4.44	180.94	185.38	1.76	16	4.43	155.60	160.03	2.04	14	4.67	132.93	137.60	2.38	12	5.14	117.58	122.72	2.67

Table 30: Multi-level pyramidal decomposable 3D-QHT execution times compared to QFT simulation times on IBM QASM simulator.

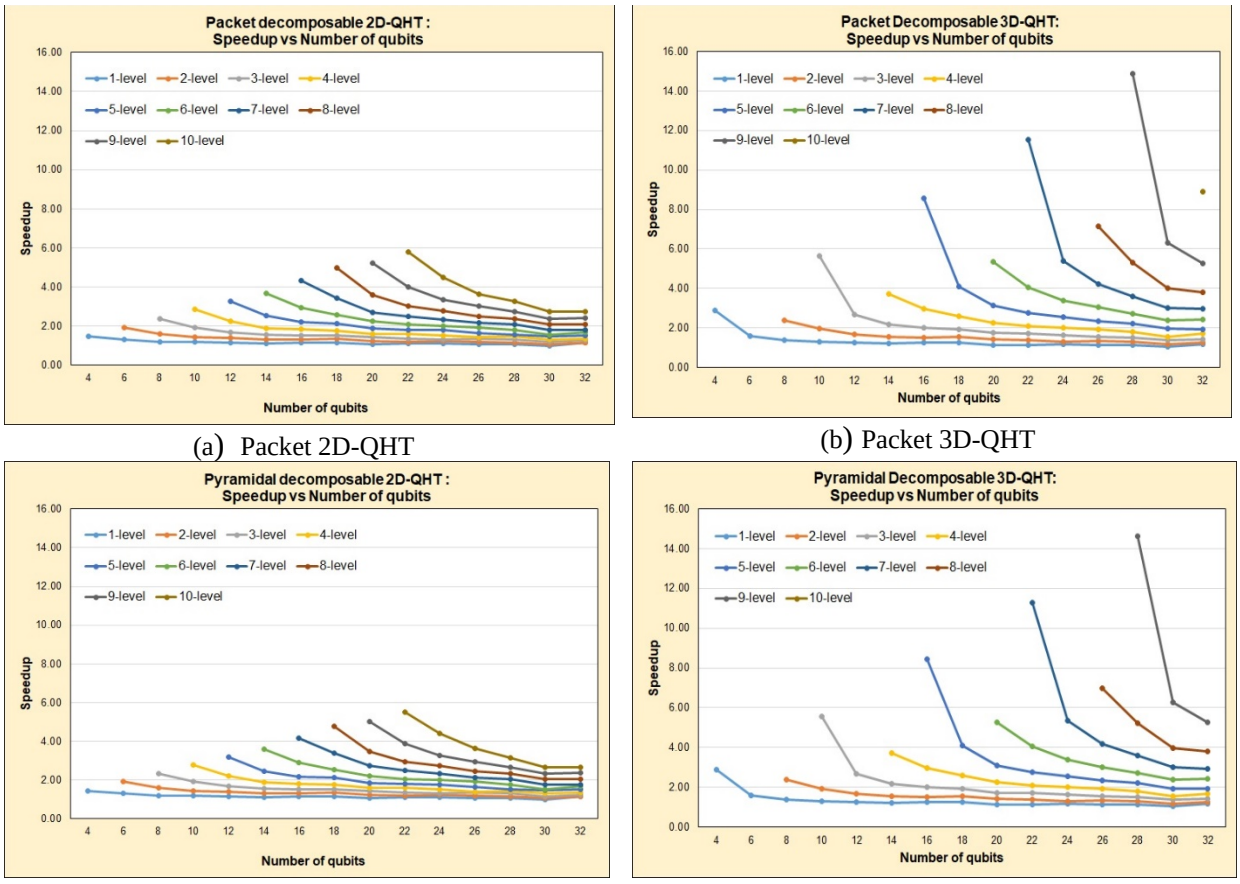
shots = 16,384		QFT	3D-QHT																								
			1-level					2-level					3-level					4-level					5-level				
Number of qubits, n	n-qubit $t_{measure}$ (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup
2	35.14	39.15																									
4	51.76	49.66	1	0.34	17.57	17.91	2.89																				
6	69.37	64.02	3	0.29	43.45	43.74	1.59																				
8	84.29	70.25	5	0.28	60.57	60.85	1.39	2	0.41	35.14	35.55	2.37															
10	101.50	93.53	7	0.30	76.83	77.13	1.32	4	0.46	51.76	52.22	1.94	1	0.63	17.57	18.20	5.58										
12	117.58	106.68	9	0.32	92.90	93.22	1.26	6	0.47	69.37	69.84	1.68	3	0.74	43.45	44.19	2.66										
14	132.93	126.11	11	0.33	109.54	109.87	1.21	8	0.57	84.29	84.86	1.57	5	0.58	60.57	61.14	2.17	2	0.65	35.14	35.79	3.71					
16	155.60	558.54	13	0.43	125.26	125.68	1.24	10	0.73	101.50	102.23	1.52	7	0.78	76.83	77.61	2.00	4	0.68	51.76	52.44	2.97	1	0.86	17.57	18.43	8.44
18	180.94	577.21	15	0.49	144.27	144.75	1.25	12	0.68	117.58	118.26	1.53	9	0.84	92.90	93.73	1.93	6	0.73	69.37	70.10	2.58	3	0.92	43.45	44.37	4.08
20	191.52	839.81	17	0.48	168.27	168.75	1.13	14	0.83	132.93	133.76	1.43	11	0.91	109.54	110.45	1.73	8	0.81	84.29	85.10	2.25	5	1.09	60.57	61.65	3.11
22	214.16	1694.57	19	0.59	186.23	186.82	1.15	16	0.93	155.60	156.53	1.37	13	0.91	125.26	126.17	1.70	10	0.92	101.50	102.42	2.09	7	1.19	76.83	78.02	2.74
24	239.15	4265.49	21	0.83	202.84	203.67	1.17	18	0.97	180.94	181.91	1.31	15	1.10	144.27	145.36	1.65	12	0.95	117.58	118.53	2.02	9	1.19	92.90	94.09	2.54
26	259.08	14675.60	23	0.79	226.66	227.44	1.14	20	0.86	191.52	192.38	1.35	17	1.12	168.27	169.39	1.53	14	1.26	132.93	134.19	1.93	11	1.29	109.54	110.83	2.34
28	281.05	70521.61	25	0.74	249.12	249.85	1.12	22	0.89	214.16	215.05	1.31	19	1.13	186.23	187.36	1.50	16	1.07	155.60	156.67	1.79	13	1.30	125.26	126.56	2.22
30	283.11	No data collected due to simulator limitations	27	0.89	270.07	270.95	1.04	24	0.96	239.15	240.11	1.18	21	1.05	202.84	203.89	1.39	18	1.03	180.94	181.97	1.56	15	1.44	144.27	145.70	1.94
32	327.12		29	0.94	282.08	283.02	1.16	26	1.18	259.08	260.26	1.26	23	1.04	226.66	227.70	1.44	20	1.48	191.52	193.00	1.69	17	1.45	168.27	169.72	1.93

shots = 16,384		QFT	3D-QHT																								
			6-level					7-level					8-level					9-level					10-level				
Number of qubits, n	n-qubit $t_{measure}$ (ms)	t_{exec} (ms)	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup	k	t_{exec} (ms)	k-qubit $t_{measure}$ (ms)	t_{total} (ms)	Speedup
20	191.52	839.81	2	1.10	35.14	36.24	5.28																				
22	214.16	1694.57	4	1.27	51.76	53.03	4.04	1	1.38	17.57	18.95	11.30															
24	239.15	4265.49	6	1.42	69.37	70.79	3.38	3	1.35	43.45	44.80	5.34															
26	259.08	14675.60	8	1.50	84.29	85.79	3.02	5	1.35	60.57	61.92	4.18	2	2.09	35.14	37.23	6.96										
28	281.05	70521.61	10	1.41	101.50	102.91	2.73	7	1.71	76.83	78.54	3.58	4	2.01	51.76	53.77	5.23	1	1.67	17.57	19.24	14.61					
30	283.11	No data collected due to simulator limitations	12	1.34	117.58	118.92	2.38	9	1.53	92.90	94.43	3.00	6	1.67	69.37	71.04	3.99	3	1.70	43.45	45.15	6.27					
32	327.12		14	1.49	132.93	134.42	2.43	11	1.76	109.54	111.30	2.94	8	1.96	84.29	86.25	3.79	5	1.66	60.57	62.22	5.26	2	2.03	35.14	37.17	8.80

Table 29 and 30 presents the execution timing data obtained from multi-level pyramidal 2D-QHT and 3D-QHT simulations, respectively. Similar to previous experiments, the n-qubit measurement timing data is obtained from Table 24, and n-qubit QFT circuit execution timing data obtained from Table 25 is also shown. For every l^{th}-level decomposition, $l = 1,2, \dots, l_{max}^{pyr}$, the 2D/3D-QHT circuit execution times, the reduced number of qubits k, and the corresponding k-qubit measurement times are also shown in Tables 29 and 30. The total time for l^{th}-level 2D/3D-QHT is calculated as the sum of the QHT circuit execution time and the corresponding k-qubit measurement time. The speedup of QHT-based total time relative to general n-qubit measurement time (without QFT or QHT) is calculated according to (55).

7.5.4 Analysis of Results

The QFT or the multi-level QHT-based methods incur overhead in the overall measurement time due to the additional QFT or QHT circuits, respectively. Using the data obtained from our experiments, we characterized the timing overheads of both methods. We also determined the speedups gained by use of the proposed packet and pyramidal QHT circuits relative to the general measurement method without QFT or QHT. For example, considering the data in Table 28, the measurement time for a 28-qubit circuit sampled for 16,384 shots is 281.05ms. If QFT-based sampling is applied, the equivalent 28-qubit QFT circuit adds a large overhead of 70s. Assuming that the number of shots required is now 1,024 as a result of QFT sampling, the reduction in measurement time from 16,384 shots to 1,024 shots, see Table 24, is much less compared to the increased overhead due to the added 28-qubit QFT circuit, see Table 25. Therefore, the overall effect is an increase in total execution time.



(a) Packet 2D-QHT (b) Packet 3D-QHT

(c) Pyramidal 2D-QHT (d) Pyramidal 3D-QHT

Fig. 63: Speedups of the proposed multi-level QHT based Q2C methods as a function of number of qubits.

Using the proposed pyramidal decomposable QHT-based sampling for the case when $n = 28$ and $l = 4$, the number of qubits is reduced from $n = 28$ to $k = 16$, see Table 28. The reduced time taken for measurement is now 155.60ms, while the additional overhead of 4-level 3D-QHT is 1.07ms. Therefore the total time is 156.67ms, which is a 44.4% reduction relative to the time taken (281.05ms) for measuring all 28 qubits, and equivalent to a speedup of $\times$ 1.79. For $n = 32$, the maximum number of decomposition levels l_{max}^{pyr}, is 10, and applying 10-level 3D-QHT results in a $\times$ 8.8 speedup in measurement time. The speedups gained by the proposed QHT-based Q2C methods relative to the general measurement is presented as a function of number of qubits in Fig. 63 for packet 2D-QHT, packet 3D-QHT, pyramidal 2D-QHT, and pyramidal 3D-QHT

respectively. It is worth mentioning that for a fixed level of decomposition of both packet and pyramidal decompositions, the speedup decreases with increase in the number of qubits, see Fig. 63. This is because for large number of qubits n, the measurement times of k qubits become very close to the n-qubit measurement times, and the overhead due to QHT becomes relatively negligible such that the speedup asymptotically approaches unity, see (55) and Fig. 63. However, for a fixed number of qubits the speedup increases, as expected, with increase in the number of decomposition levels, see Fig. 63. It is also worth noting that the maximum speedup at a particular decomposition level is always higher for 3D-QHT compared to 2D-QHT for either packet or pyramidal which shows the efficiency of our proposed techniques for larger datasets of higher dimensions.

Conclusions

Quantum computing is at its nascent stage, and it is the right time to explore all its possibilities and potential. There are numerous challenges to quantum computing technology, and in this work, we investigated and proposed solutions to several important quantum computing problems. We also demonstrated how classical reconfigurable hardware such as FPGAs can be efficiently utilized in emulating the behavior of quantum systems and algorithms. We proposed methodologies for performing classical-to-quantum (C2Q) data encoding and quantum-to-classical (Q2C) data decoding, and presented the corresponding optimized quantum circuits. We investigated several quantum algorithms such as Quantum Haar Transform, Quantum Grover's Search and proposed circuit optimizations and extensions. A hardware-based emulation framework was developed for investigating quantum algorithms. An OpenCL-based methodology was used to develop and deploy emulation hardware architectures for quantum algorithms on HPRC systems. The flexibility of the proposed emulation framework allowed us to extend algorithms with newer capabilities, optimize algorithms, and combine algorithms to develop new applications. For example, a novel quantum application was proposed for pattern matching using quantum dimension reduction, that can be used in domains such as High-Energy Physics and Hyperspectral Remote-Sensing. We also explored architectural optimizations for the proposed emulation framework to achieve higher scalability, accuracy, and throughput, compared to existing emulators. Future directions of this work are integrating run-time full/partial reconfiguration with the proposed emulation framework, and developing methodologies for a multi-node (multi-FPGA) architectures for emulation of extreme-scale quantum circuits.

References

[1] P. Benioff, “The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines,” *J. Stat. Phys.*, vol. 22, no. 5, pp. 563–591, May 1980, doi: 10.1007/BF01011339.

[2] R. P. Feynman, “Simulating physics with computers,” *Int J Theor Phys*, vol. 21, no. 6/7, 1982.

[3] D. Deutsch, “Quantum theory, the Church–Turing principle and the universal quantum computer,” *Proc. R. Soc. Lond. Math. Phys. Sci.*, vol. 400, no. 1818, pp. 97–117, 1985.

[4] P. W. Shor, “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” p. 30.

[5] L. K. Grover, “A fast quantum mechanical algorithm for database search,” 1996, pp. 212–219.

[6] S. Boixo *et al.*, “Characterizing quantum supremacy in near-term devices,” *Nat. Phys.*, vol. 14, no. 6, pp. 595–600, 2018.

[7] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, 2018.

[8] F. Arute *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[9] S. Jordan, “Quantum algorithm zoo,” *Retrieved June*, vol. 27, p. 2013, 2011.

[10] Y. Cao *et al.*, “Quantum chemistry in the age of quantum computing,” *Chem. Rev.*, vol. 119, no. 19, pp. 10856–10915, 2019.

[11] M. A. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2002.

[12] L. Gomes, “Quantum computing: Both here and not here,” *IEEE Spectr.*, vol. 55, no. 4, pp. 42–47, 2018.

[13] M. Schlosshauer, “Quantum decoherence,” *Phys. Rep.*, vol. 831, pp. 1–57, 2019.

[14] IBMQ, “User guides.” Accessed: June. 24, 2022. [Online]. Available: https://quantum-computing.ibm.com/docs/

[15] S. S. Tannu and M. K. Qureshi, “Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.

[16] C. P. Williams and S. H. Clearwater, *Explorations in quantum computing*. Springer, 1998.

[17] C. Reedy, *When will quantum computers be consumer products? Futurism*. 2017.

[18] D. P. DiVincenzo, “The physical implementation of quantum computation,” *Fortschritte Phys. Prog. Phys.*, vol. 48, no. 9-11, pp. 771–783, 2000.

[19] V. V. Shende, S. S. Bullock, and I. L. Markov, “Synthesis of quantum logic circuits,” in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, New York, NY, USA, Jan. 2005, pp. 272–275. doi: 10.1145/1120725.1120847.

[20] A. Amariutei and S. Caraiman, “Parallel quantum computer simulation on the GPU,” in *15th International Conference on System Theory, Control and Computing*, Oct. 2011, pp. 1–6.

[21] J. Chen, F. Zhang, C. Huang, M. Newman, and Y. Shi, “Classical simulation of intermediate-size quantum circuits,” *ArXiv Prepr. ArXiv180501450*, 2018.

[22] H. De Raedt *et al.*, “Massively parallel quantum computer simulator, eleven years later,” *Comput. Phys. Commun.*, vol. 237, pp. 47–61, Apr. 2019, doi: 10.1016/j.cpc.2018.11.005.

[23] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, “QuEST and high performance simulation of quantum computers,” *Sci. Rep.*, vol. 9, no. 1, pp. 1–11, 2019.

[24] D. Willsch, M. Willsch, F. Jin, K. Michielsen, and H. De Raedt, “GPU-accelerated simulations of quantum annealing and the quantum approximate optimization algorithm,” *ArXiv210403293 Phys. Physicsquant-Ph*, Apr. 2021, Accessed: June. 24, 2022. [Online]. Available: http://arxiv.org/abs/2104.03293

[25] M. Fujishima, “FPGA-based high-speed emulator of quantum computing,” in *Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT)(IEEE Cat. No. 03EX798)*, 2003, pp. 21–26.

[26] T. Häner, D. S. Steiger, M. Smelyanskiy, and M. Troyer, “High performance emulation of quantum circuits,” in *SC’16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 866–874.

[27] A. U. Khalid, Z. Zilic, and K. Radecka, “FPGA emulation of quantum circuits,” in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, 2004, pp. 310–315.

[28] M. Khalil-Hani, Y. H. Lee, M. N. Marsono, B. Javadi, and S. K. Garg, “An accurate FPGA-based hardware emulation on quantum fourier transform,” in *Proceedings of the 13th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2015)*, 2015, pp. 23–30.

[29] Y. H. Lee, M. Khalil-Hani, and M. N. Marsono, “An FPGA-based quantum computing emulation framework based on serial-parallel architecture,” *Int. J. Reconfigurable Comput.*, vol. 2016, 2016.

[30] A. Silva and O. G. Zabaleta, “FPGA quantum computing emulator using high level design tools,” in *2017 Eight Argentine Symposium and Conference on Embedded Systems (CASE)*, 2017, pp. 1–6.

[31] P. A. M. Dirac, “A new notation for quantum mechanics,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, 1939, vol. 35, no. 3, pp. 416–418.

[32] F. L. Marquezino, R. Portugal, and F. D. Sasse, “Obtaining the Quantum Fourier Transform from the classical FFT with QR decomposition,” *J. Comput. Appl. Math.*, vol. 235, no. 1, pp. 74–81, Nov. 2010, doi: 10.1016/j.cam.2010.05.012.

[33] Esam El-Araby, Tarek El-Ghazawi, J. L. Moigne, and K. Gaj, “Wavelet spectral dimension reduction of hyperspectral imagery on a reconfigurable computer,” in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, Dec. 2004, pp. 399–402. doi: 10.1109/FPT.2004.1393309.

[34] J. M. G. Wickmann, “A wavelet approach to dimension reduction and classification of hyperspectral data,” Master’s Thesis, 2007.

[35] A. Fijany and C. P. Williams, “Quantum Wavelet Transforms: Fast Algorithms and Complete Circuits,” in *Quantum Computing and Quantum Communications*, Berlin, Heidelberg, 1999, pp. 10–33. doi: 10.1007/3-540-49208-9_2.

[36] N. Mahmud, B. Haase-Divine, A. MacGillivray, and E. El-Araby, “Quantum Dimension Reduction for Pattern Recognition in High-Resolution Spatio-Spectral Data,” *IEEE Trans. Comput.*, 2020.

[37] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, “Tight Bounds on Quantum Searching,” *Fortschritte Phys.*, vol. 46, no. 4–5, pp. 493–505, 1998. Accessed: June 24, 2022. doi: https://doi.org/10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P.

[38] A. Avila, A. Maron, R. Reiser, M. Pilla, and A. Yamin, “GPU-aware distributed quantum simulation,” in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 860–865.

[39] M. Aminian, M. Saeedi, M. S. Zamani, and M. Sedighi, “FPGA-based circuit model emulation of quantum algorithms,” in *2008 IEEE Computer Society Annual Symposium on VLSI*, 2008, pp. 399–404.
[40] M. Weigold, J. Barzen, F. Leymann, and M. Salm, “Data encoding patterns for quantum computing,” in *Proceedings of the 27th Conference on Pattern Languages of Programs*, 2020, pp. 1–11.
[41] Y. Zhang, K. Lu, Y. Gao, and M. Wang, “NEQR: a novel enhanced quantum representation of digital images,” *Quantum Inf. Process.*, vol. 12, no. 8, pp. 2833–2860, 2013.
[42] P. Q. Le, F. Dong, and K. Hirota, “A flexible representation of quantum images for polynomial preparation, image compression, and processing operations,” *Quantum Inf. Process.*, vol. 10, no. 1, pp. 63–84, 2011.
[43] S. Johri *et al.*, “Nearest centroid classification on a trapped ion quantum computer,” *Npj Quantum Inf.*, vol. 7, no. 1, pp. 1–11, 2021.
[44] L. Song and C. P. Williams, “Computational synthesis of any n -qubit pure or mixed state,” Orlando, FL, Aug. 2003, p. 195. doi: 10.1117/12.486347.
[45] M. Mottonen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, “Quantum Circuits for General Multiqubit Gates,” *Phys. Rev. Lett.*, vol. 93, no. 13, p. 130502, Sep. 2004, doi: 10.1103/PhysRevLett.93.130502.
[46] M. Mottonen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, “Transformation of quantum states using uniformly controlled rotations,” *ArXivquant-Ph0407010*, Jul. 2004, Accessed: June 24, 2022. [Online]. Available: http://arxiv.org/abs/quant-ph/0407010
[47] P. Niemann, R. Datta, and R. Wille, “Logic Synthesis for Quantum State Generation,” in *2016 IEEE 46th International Symposium on Multiple-Valued Logic (ISMVL)*, May 2016, pp. 247–252. doi: 10.1109/ISMVL.2016.30.
[48] J. Watrous, “Quantum Computational Complexity,” *ArXiv08043401 Quant-Ph*, Apr. 2008, Accessed: June 24, 2022. [Online]. Available: http://arxiv.org/abs/0804.3401
[49] M. Möttönen and J. J. Vartiainen, “Decompositions of general quantum gates,” *Trends Quantum Comput. Res.*, 2006.
[50] K. Hwang and N. Jotwani, *Advanced computer architecture: parallelism, scalability, programmability*, vol. 199. McGraw-Hill New York, 1993.
[51] G. Nannicini, “An introduction to quantum computing, without the physics,” *SIAM Rev.*, vol. 62, no. 4, pp. 936–981, 2020.
[52] K. Jacobs, *Quantum measurement theory and its applications*. Cambridge University Press, 2014.
[53] S. Kaewpijit, J. Le Moigne, and T. El-Ghazawi, “Automatic reduction of hyperspectral imagery using wavelet spectral analysis,” *IEEE Trans. Geosci. Remote Sens.*, vol. 41, no. 4, pp. 863–871, 2003.
[54] C. Van Loan, *Computational frameworks for the fast Fourier transform*. SIAM, 1992.
[55] P. Hoyer, “Efficient quantum transforms,” *ArXiv Prepr. Quant-Ph9702028*, 1997.
[56] H.-S. Li, P. Fan, H. Xia, S. Song, and X. He, “The multi-level and multi-dimensional quantum wavelet packet transforms,” *Sci. Rep.*, vol. 8, no. 1, pp. 1–23, 2018.
[57] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *ArXiv Prepr. ArXiv14114028*, 2014.
[58] C. Figgatt, D. Maslov, K. A. Landsman, N. M. Linke, S. Debnath, and C. Monroe, “Complete 3-qubit Grover search on a programmable quantum computer,” *Nat. Commun.*, vol. 8, no. 1, pp. 1–9, 2017.

[59] W. Huang *et al.*, “Fidelity benchmarks for two-qubit gates in silicon,” *Nature*, vol. 569, no. 7757, pp. 532–536, 2019.
[60] E. El-Araby, T. El-Ghazawi, J. Le Moigne, and R. Irish, “Reconfigurable processing for satellite on-board automatic cloud cover assessment,” *J. Real-Time Image Process.*, vol. 4, no. 3, pp. 245–259, 2009.
[61] E. El-Araby, S. G. Merchant, and T. El-Ghazawi, “Assessing productivity of high-level design methodologies for high-performance reconfigurable computers,” in *High-Performance Computing using FPGAs*, Springer, 2013, pp. 719–745.
[62] V. Kathail, “Xilinx Vitis Unified Software Platform,” in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, Feb. 2020, pp. 173–174. doi: 10.1145/3373087.3375887.
[63] “IBM Quantum System One,” *IBM Quantum System One*, Mar. 12, 2018. https://research.ibm.com/ibm-q/qed/index.html (Accessed June 24, 2022).
[64] M. I. and Strategy, “IBM Doubles Its Quantum Computing Power Again,” *Forbes*. https://www.forbes.com/sites/moorinsights/2020/01/08/ibm-doubles-its-quantum-computing-power-again/ (Accessed June 24, 2022).
[65] “Driving quantum performance: more qubits, higher Quantum Volume, and now a proper measure of speed,” *IBM Research Blog*, Feb. 09, 2021. https://research.ibm.com/blog/circuit-layer-operations-per-second (accessed Jan. 26, 2022).
[66] “ibmq-device-information/backends at master · Qiskit/ibmq-device-information,” *GitHub*. https://github.com/Qiskit/ibmq-device-information (Accessed June 24, 2022).
[67] R. Jozsa, “Fidelity for mixed quantum states,” *J. Mod. Opt.*, vol. 41, no. 12, pp. 2315–2323, 1994.
[68] A. Uhlmann, “The ‘transition probability’ in the state space of a$*$-algebra,” *Rep. Math. Phys.*, vol. 9, no. 2, pp. 273–279, 1976.
[69] A. Munshi, “The opencl specification,” in *2009 IEEE Hot Chips 21 Symposium (HCS)*, 2009, pp. 1–314.
[70] Intel, “Intel® Stratix® 10 Logic Array Blocks and Adaptive Logic Modules User Guide.” 2020.
[71] “A Preview of Bristlecone, Google’s New Quantum Processor,” *Google AI Blog*. http://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html (Accessed June 24, 2022).
[72] B. Villalonga *et al.*, “Establishing the quantum supremacy frontier with a 281 pflop/s simulation,” *Quantum Sci. Technol.*, vol. 5, no. 3, p. 034003, 2020.

Appendix

A1. Algorithm 1

Algorithm 1 - Multi-level 1D Quantum Haar Transform

Input: $X = [x_0, x_1, ...x_{N-1}]$, n_{rows}, n_{cols}, n_{levels}
Output: $Y = [y_0, y_1, ...y_{N-1}]$
$n_{states} = n_{rows} \times n_{cols} = N$
for $i_{level} = 1; i_{level} \leq n_{levels}; i_{level} + +$ **do**

  for $i_{group} = 0; i_{group} < (n_{states}/2); i_{group} + +$ **do**

    // Initial scheduler setup
    $i_{colGroup} = \lfloor i_{group}/(n_{rows}/2) \rfloor$
    $i_{rowGroup} = i_{group} - i_{colGroup} \times (n_{rows}/2)$
    $i_{col} = i_{colGroup}$
    $i_{row} = 2 \times i_{rowGroup}$

    // Input Permutations/Scheduler
    $i_{X_{00}} = i_{row} + (i_{col} \times n_{rows})$
    $i_{X_{10}} = i_{X_{00}} + 1$

    $X_{00} \leftarrow X[i_{X_{00}}]$
    $X_{10} \leftarrow X[i_{X_{10}}]$

    // 1D-QHT kernel
    $Y_{00} = (X_{00} + X_{10})/\sqrt{2}$
    $Y_{10} = (X_{00} - X_{10})/\sqrt{2}$

    // Output Permutations/Scheduler
    $i_{Y_{00}} = (i_{row} + (i_{col} \times n_{rows}))/2$
    $i_{Y_{10}} = i_{Y_{00}} + (n_{rows}/2)$

    $Y[i_{Y_{00}}] \leftarrow Y_{00}$
    $Y[i_{Y_{10}}] \leftarrow Y_{10}$
  end for
end for

A2. Algorithm 2

Algorithm 2 - Multi-level 2D Quantum Haar Transform

Input: $X = [x_0, x_1, ...x_{N-1}]$, n_{rows}, n_{cols}, n_{levels}
Output: $Y = [y_0, y_1, ...y_{N-1}]$
$n_{states} = n_{rows} \times n_{cols} = N$
for $i_{level} = 1; i_{level} \leq n_{levels}; i_{level}++$ **do**

  for $i_{group} = 0; i_{group} < (n_{states}/4); i_{group}++$ **do**

    // Initial scheduler setup
    $i_{colGroup} = \lfloor i_{group}/(n_{rows}/2) \rfloor$
    $i_{rowGroup} = i_{group} - i_{colGroup} \times (n_{rows}/2)$
    $i_{col} = 2 \times i_{colGroup}$
    $i_{row} = 2 \times i_{rowGroup}$

    // Input Permutations/Scheduler
    $i_{X_{00}} = i_{row} + (i_{col} \times n_{rows})$
    $i_{X_{10}} = i_{X_{00}} + 1$
    $i_{X_{01}} = i_{X_{00}} + n_{rows}$
    $i_{X_{11}} = i_{X_{01}} + 1$

    $X_{00} \leftarrow X[i_{X_{00}}]$
    $X_{10} \leftarrow X[i_{X_{10}}]$
    $X_{01} \leftarrow X[i_{X_{01}}]$
    $X_{11} \leftarrow X[i_{X_{11}}]$

    // 2D-QHT kernel
    $Y_{00} = (X_{00} + X_{10} + X_{01} + X_{11})/2$
    $Y_{10} = (X_{00} - X_{10} + X_{01} - X_{11})/2$
    $Y_{01} = (X_{00} + X_{10} - X_{01} - X_{11})/2$
    $Y_{11} = (X_{00} - X_{10} - X_{01} + X_{11})/2$

    // Output Permutations/Scheduler
    $i_{Y_{00}} = (i_{row} + (i_{col} \times n_{rows}))/2$
    $i_{Y_{10}} = i_{Y_{00}} + (n_{rows}/2)$
    $i_{Y_{01}} = i_{Y_{00}} + (n_{states}/2)$
    $i_{Y_{11}} = i_{Y_{01}} + (n_{rows}/2)$

    $Y[i_{Y_{00}}] \leftarrow Y_{00}$
    $Y[i_{Y_{10}}] \leftarrow Y_{10}$
    $Y[i_{Y_{01}}] \leftarrow Y_{01}$
    $Y[i_{Y_{11}}] \leftarrow Y_{11}$
  end for
end for

A3. Algorithm 3

Algorithm 3 - Multi-level 3D Quantum Haar Transform

Input: $X = [x_0, x_1, ..., x_{N-1}]$, n_{rows}, n_{cols}, n_{bands} n_{levels}
Output: $Y = [y_0, y_1, ..., y_{N-1}]$
$n_{states/band} = n_{rows} \times n_{cols}$
$n_{states} = n_{states/band} \times n_{bands} = N$
$n_{groups/band} = (n_{rows} \times n_{cols})/4$
for $i_{level} = 1$; $i_{level} \leq n_{levels}$; $i_{level}++$ **do**

for $i_{group} = 0$; $i_{group} < (n_{states}/8)$; $i_{group}++$ **do**

// Initial scheduler setup
$i_{bandGroup} = \lfloor i_{group}/n_{groups/band} \rfloor$
$i_{groups/band} = i_{group} - (i_{bandGroup} * n_{groups/band})$
$i_{colGroup} = \lfloor i_{groups/band}/(n_{rows}/2) \rfloor$
$i_{rowGroup} = i_{groups/band} - i_{colGroup} \times (n_{rows}/2)$
$i_{band} = 2 \times i_{bandGroup}$
$i_{col} = 2 \times i_{colGroup}$
$i_{row} = 2 \times i_{rowGroup}$
// Input Permutations/Scheduler
$i_{X_{000}} = i_{row} + i_{col} * n_{rows} + i_{band} * n_{states/band}$
$i_{X_{001}} = i_{X_{000}} + 1$
$i_{X_{010}} = i_{X_{000}} + n_{rows}$
$i_{X_{011}} = i_{X_{010}} + 1$
$i_{X_{100}} = i_{X_{000}} + n_{states/band}$
$i_{X_{101}} = i_{X_{100}} + 1$
$i_{X_{110}} = i_{X_{100}} + n_{rows}$
$i_{X_{111}} = i_{X_{110}} + 1$
$X_{000} \leftarrow X[i_{X_{000}}]$
$X_{001} \leftarrow X[i_{X_{001}}]$
$X_{010} \leftarrow X[i_{X_{010}}]$
$X_{011} \leftarrow X[i_{X_{011}}]$
$X_{100} \leftarrow X[i_{X_{100}}]$
$X_{101} \leftarrow X[i_{X_{101}}]$
$X_{110} \leftarrow X[i_{X_{110}}]$
$X_{111} \leftarrow X[i_{X_{111}}]$
// 3D-QHT kernel
$Y_{000} = (X_{000} + X_{001} + X_{010} + X_{011} + X_{100} + X_{101} + X_{110} + X_{111})/2\sqrt{2}$
$Y_{001} = (X_{000} - X_{001} + X_{010} - X_{011} + X_{100} - X_{101} + X_{110} - X_{111})/2\sqrt{2}$
$Y_{010} = (X_{000} + X_{001} - X_{010} - X_{011} + X_{100} + X_{101} - X_{110} - X_{111})/2\sqrt{2}$
$Y_{011} = (X_{000} - X_{001} - X_{010} + X_{011} + X_{100} - X_{101} - X_{110} + X_{111})/2\sqrt{2}$
$Y_{100} = (X_{000} + X_{001} + X_{010} + X_{011} - X_{100} - X_{101} - X_{110} - X_{111})/2\sqrt{2}$
$Y_{101} = (X_{000} - X_{001} + X_{010} - X_{011} - X_{100} + X_{101} - X_{110} + X_{111})/2\sqrt{2}$
$Y_{110} = (X_{000} + X_{001} - X_{010} - X_{011} - X_{100} - X_{101} + X_{110} + X_{111})/2\sqrt{2}$
$Y_{111} = (X_{000} - X_{001} - X_{010} + X_{011} - X_{100} + X_{101} + X_{110} - X_{111})/2\sqrt{2}$
// Output Permutations/Scheduler
$i_{Y_{000}} = (i_{row} + i_{col} * n_{rows} + i_{band} * n_{states/band})/2$
$i_{Y_{001}} = i_{Y_{000}} + (n_{rows}/2)$
$i_{Y_{010}} = i_{Y_{000}} + (n_{states/band}/2)$
$i_{Y_{011}} = i_{Y_{010}} + (n_{rows}/2)$
$i_{Y_{100}} = i_{Y_{000}} + (n_{states}/2)$
$i_{Y_{101}} = i_{Y_{100}} + (n_{rows}/2)$
$i_{Y_{110}} = i_{Y_{100}} + (n_{states/band}/2)$
$i_{Y_{111}} = i_{Y_{110}} + (n_{rows}/2)$
$Y[i_{Y_{000}}] \leftarrow Y_{000}$
$Y[i_{Y_{001}}] \leftarrow Y_{001}$
$Y[i_{Y_{010}}] \leftarrow Y_{010}$
$Y[i_{Y_{011}}] \leftarrow Y_{011}$
$Y[i_{Y_{100}}] \leftarrow Y_{100}$
$Y[i_{Y_{101}}] \leftarrow Y_{101}$
$Y[i_{Y_{110}}] \leftarrow Y_{110}$
$Y[i_{Y_{111}}] \leftarrow Y_{111}$
end for
end for