Exploration of Design Alternatives for Reducing Idle Time in Shor's Algorithm: A Study on Monolithic and Distributed Quantum Systems

Moritz Schmidt[1], Abhoy Kole[2], Leon Wichette[3], Rolf Drechsler[2, 4], Frank Kirchner[1,3,*], Elie Mounzer[3,*]

[1] Robotics Research Group, University of Bremen, 28359 Bremen, Germany
[2] German Research Center for Artificial Intelligence Cyber-Phyiscal Systems (CPS), 28359 Bremen, Germany
[3] German Research Center for Artificial Intelligence Robotics Innovation Center (RIC), 28359 Bremen, Germany
[4] Group of Computer Architecture, University of Bremen, 28359 Bremen, Germany
[*] Co-Principal Investigator (Co-PI)
Corresponding author: Moritz Schmidt (email: moritz.schmidt@uni-bremen.de).

Abstract—**Shor's algorithm is one of the most prominent quantum algorithms, yet finding efficient implementations remains an active research challenge. While many approaches focus on low-level modular arithmetic optimizations, a broader perspective can provide additional opportunities for improvement. By adopting a mid-level abstraction, we analyze the algorithm as a sequence of computational tasks, enabling systematic identification of idle time and optimization of execution flow. Building on this perspective, we first introduce an alternating design approach to minimizes idle time while preserving qubit efficiency in Shor's algorithm. By strategically reordering tasks for simultaneous execution, we achieve a substantial reduction in overall execution time. Extending this approach to distributed implementations, we demonstrate how task rearrangement enhances execution efficiency in the presence of multiple distribution channels. Furthermore, to effectively evaluate the impact of design choices, we employ static timing analysis (STA)—a technique from classical circuit design—to analyze circuit delays while accounting for hardware-specific execution characteristics, such as measurement and reset delays in monolithic architectures and ebit generation time in distributed settings. Finally, we validate our approach by integrating modular exponentiation circuits from QRISP and constructing circuits for factoring numbers up to 64 bits. Through an extensive study across neutral atom, superconducting, and ion trap quantum computing platforms, we analyze circuit delays, highlighting trade-offs between qubit efficiency and execution time. Our findings provide a structured framework for optimizing compiled quantum circuits for Shor's algorithm, tailored to specific hardware constraints.**

I. INTRODUCTION

Shor's algorithm [1] offers an exponential speedup for integer factorization, presenting a substantial threat to RSA-based cryptographic security [2]–[5]. Although large-scale quantum computers capable of breaking RSA are still developed, improving the efficiency of Shor's algorithm remains vital for advancing practical quantum computing and shaping post-quantum cryptography research.

Most previous research on implementing Shor's algorithm has focused on optimizing low-level modular arithmetic circuits, particularly modular addition and multiplication [5]–[15]. However, these efforts primarily refine individual operations without fully exploiting the algorithm's overall structure to reduce execution bottlenecks. Adopting a mid-level abstraction allows us to break the algorithm into structured computational tasks rather than solely optimizing gate-level arithmetic. This task-based perspective facilitates a parallelization strategy, allowing for a systematic analysis of idle time and the optimization of critical paths of computation. Moreover, as algorithm-level descriptions are translated into compiled circuits, hardware-specific execution characteristics become crucial in determining performance. Integrating these factors at the compilation stage can lead to more efficient implementations tailored to specific quantum architectures.

In this work, we enable the concurrent execution of operations that were previously sequential by strategically reordering mid-level computational tasks, leading to enhanced performance. As part of this approach, we introduce an alternating design that further reduces idle time while maintaining qubit efficiency. We extend this method to distributed quantum computing (DQC) [16]–[20], demonstrating how task rearrangements optimize execution when multiple ebit channels are available. To systematically assess the impact of different design choices, we utilize static timing analysis (STA) [21], [22], a technique widely applied in classical circuit design to analyze execution timing under hardware constraints. This allow us to assess circuit delays across various quantum hardware platforms and gain insights into how execution bottlenecks arise at different levels of abstraction.

Our results indicate that the proposed task parallelization methods remain effective regardless of the target hardware architecture. In monolithic systems, we demonstrate that idle time can be significantly reduced and quantify the trade-offs between qubit efficiency and execution time, particularly in architectures with slow reset and measurement operations. In distributed setups, our analysis reveals how execution time is influenced by the interplay between ebit generation time, the number of ebit channels, and the size of the factored number. To evaluate our approach, we integrate modular exponentiation circuits from the QRISP library [23] into our designs, enabling us to construct circuits capable of factoring large numbers. We conduct a comprehensive study across neutral atom, superconducting, and ion-trap quantum computing platforms, analyzing how our designs impact circuit delays.

The insights gained from this analysis provide a starting point for optimizing quantum circuit compilation based on specific hardware constraints.

The remainder of this paper is structured as follows: Section II introduces Shor's algorithm, key concepts in dynamic circuits, distributed quantum computing, and the STA techniques used for execution time analysis. Section III presents our mid-level abstraction perspective, examines existing circuit designs in terms of qubit count and idle time, introduces our alternating design and extends this perspective to distributed execution. Section IV details our evaluation methodology, including circuit construction using QRISP, hardware modeling, and delay estimation. Section V presents our experimental results and their implications for quantum circuit design. Finally, section VI concludes with a discussion of future research directions.

II. BACKGROUND

A. Shor's Algorithm

Shor's algorithm [1] is designed to factor a large composite integer N into its prime components p and q where $N = p \cdot q$[1].The factorization problem is reformulated as an order-finding problem. The goal is to find the period r of the modular exponentiation function $f(x) = a^x \mod N$, for a randomly chosen positive integer $a < N$ coprime to N. Once r is obtained, the unknown factors p and q can be computed classically by solving $\gcd(a^{\frac{r}{2}} \pm 1, N)$ using Euclid's algorithm. By proposing an efficient quantum subroutine for finding r, Shor made the factoring problem solvable in polynomial time on a Quantum Processing Unit (QPU).

The quantum order finding subroutine relies on the unitary implementation of the modular exponentiation function $f(x)$:

$$U_f : |x\rangle |y\rangle \mapsto |x\rangle |y \cdot a^x \mod N\rangle \tag{1}$$

Here, the register $|y\rangle$, used for computing the modular exponentiation, is called the *work register* (denoted as $|r_w\rangle$) and requires n qubits, where $n = \lceil \log_2 N \rceil$. The data register (denoted as $|r_d\rangle$), which stores the exponent x, has a size of $m = 2n$.

The algorithm consists of three steps: (i) Initialize the work register $|r_w\rangle$ to 1 and the data register $|r_d\rangle$ to an equal superposition using Hadamard gates; (ii) Apply the unitary operation U_f, and (iii) Apply the inverse Quantum Fourier Transform ($QFT^\dagger$) to the data register $|r_d\rangle$. Upon measuring the final state, an m-bit estimate of j/r is obtained, where $j \in \{0, \dots, r-1\}$. The order r can then be determined through classical post-processing using continued fractions. Figure 1 shows the complete high-level circuit of Shor's algorithm.

The efficiency of Shor's algorithm primarily depends on the implementations of the QFT and modular exponentiation operator U_f. For the QFT, a well-established implementation exists using a gateset composed of controlled phase rotations and Hadamard gates. However, the primary bottleneck lies in the efficient implementation of the modular exponentiation U_f, which poses a substantial challenge. Since U_f must be unitary,

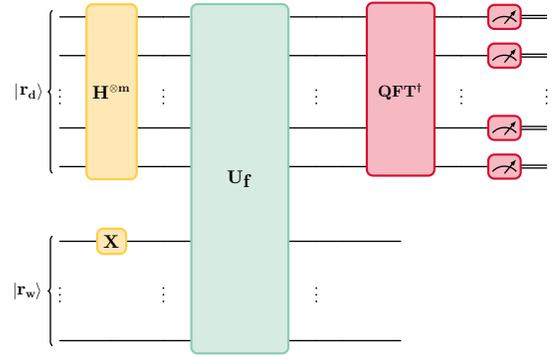[1]For a more detailed introduction we refer to [24]

Fig. 1: A high-level circuit diagram of Shor's algorithm for prime factorization.

reversible arithmetic techniques, such as uncomputation [6], must be employed. Additionally, extra ancilla qubits are often required for the work register.

At the lowest level of abstraction, the core building block for modular exponentiation is a modular adder. Various types of adders have been proposed [7]–[13] and can be classified based on their underlying approaches, including ripple carry [7], [8], carry look-ahead [9] and QFT-based methods [10], [12], [13]. These adders are used within hierarchical structures that progress from (modular) addition to multiplication and finally exponentiation. Implementations are typically evaluated based on trade-offs between circuit width and depth. For instance, some approaches minimize the required qubit count by increasing circuit depth [13]–[15], achieving a current minimum of $2n + 2$ qubits through the reuse of dirty ancillas [14], [15].

Specialized circuits have also been developed for demonstrating Shor's algorithm on Noisy Intermediate-Scale Quantum (NISQ) hardware [25]–[27]. These circuits are often highly optimized for specific values of N and a, rather than being general-purpose solutions. For detailed discussions on optimization techniques, such as window arithmetic, corset representations, and comparisons of adder implementations, we refer readers to [5], [10].

Our proposed designs operate at a mid-level abstraction layer, decomposing Shor's components into tasks, based on its interpretation as an application case of Quantum Phase Estimation (QPE), and without relying on specific arithmetic implementations. This approach ensures compatibility with a wide range of low-level optimizations, including various modular addition realizations.

B. Dynamic Circuits

Circuits that utilize mid-circuit measurements, qubit resets, and feed-forward control based on classical information are referred to as *dynamic circuits* [28]–[30]. These circuit elements are not only essential for implementing quantum error detection and correction [31], [32], but have also recently been found to aid in other tasks, such as enabling efficient long-range entanglement [33], [34] and state preparation [35], [36].

Dynamic circuits play a key role in optimizing Shor's algorithm. This is especially relevant with the introduction of the semiclassical implementation of the QFT [37], [38]. If the

QFT is followed by measurement rather than additional gates, the principle of deferred measurement [24] can be applied. This enables earlier measurements and substitutes each two-qubit controlled rotation with a single-qubit rotation, governed by a classical measurement result, in the QFT realization (see Figure 2). In Iterative Phase Estimation (IPE) [39], [40],
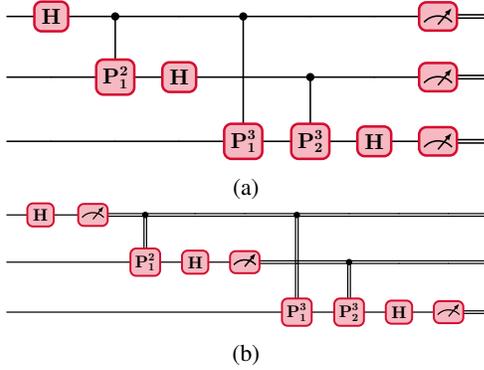

(a)

(b)

Fig. 2: (a) The standard QFT implementation utilizing two-qubit controlled rotations. (b) The semiclassical QFT using early measurements to replace the two-qubit gates with classically controlled single-qubit phase rotations.

the semiclassical QFT is used within QPE, allowing each qubit of the QFT to be executed sequentially. Since no multi-qubit gates are involved, a single qubit is sufficient if qubit resets are available. Shor's order-finding routine, which can be viewed as a form of QPE, benefits from IPE to reduce the required qubit count [41], [42]. This *iterative* approach has been experimentally demonstrated for two digit values of N [43].

While dynamic circuits are now supported by several hardware architectures, including superconducting [30], ion-trap [32], and neutral atom QPUs [44], [45], and corresponding compilation frameworks have been proposed [46], [47], finding practical use cases where these tools can be effectively utilized remains a challenge and often requires manual effort.

C. Distributed Quantum Computing

Current quantum hardware faces limitations in qubit scaling due to technology-specific engineering challenges. Superconducting quantum computers rely on dilution refrigerators to keep qubits at low temperatures. As the number of qubits increases, the complexity and cost of managing wiring, control systems, and refrigeration also rise significantly [48]. In neutral atom systems, efficiently scaling qubit arrays requires high-intensity lasers to maintain strong optical tweezers while optimizing spatial constraints [49]. Trapped-ion systems face challenges with frequency crowding, where closely spaced motional modes complicate individual qubit addressing without crosstalk, necessitating precise control lasers [50]. To address these challenges and enable interaction between distant quantum computers, Distributed Quantum Computing has been proposed. DQC involves operating multiple QPUs interconnected in a network, allowing for scalable quantum processing [16]. This section offers an overview of the key terms and concepts from [17] that are relevant to this work.

In a DQC environment, each QPU contains a set of qubits, which are classified into two types: *compute qubits*, used for general computations, and *communication qubits*. The communication qubits are responsible for generating shared entanglement between distant QPUs. DQC protocols depend on communication *ebits* sharing the Bell state:

$$|\Phi^+\rangle_{AB} = \frac{1}{\sqrt{2}}(|0\rangle_A |0\rangle_B + |1\rangle_A |1\rangle_B) \qquad (2)$$

where the first qubit from the pair facilitates communication with the compute qubits on QPU A, while the second qubit enables communication with the compute qubits on QPU B.
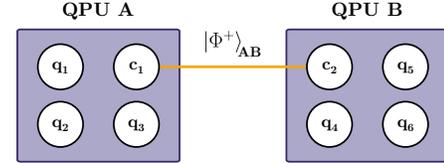

Fig. 3: A distributed setup with two QPUs, A and B, linked by an ebit channel that generates ebits $|\Phi^+\rangle_{AB}$ on communication qubits c_1 and c_2, facilitates computation involving computing qubits qi and qj, located on separate QPUs.

A communication channel where such ebits sharing the Bell state $|\Phi^+\rangle_{AB}$ are repeatedly generated is called an *ebit channel* (see Figure 3).

Analogous to classical communication, when computing nodes lack direct ebit channels, *Quantum repeaters* and *quantum routers* leveraging the entanglement swapping (ES) protocol have also been proposed [18], [51], [52] to facilitate communication.

There are two primary protocols for utilizing ebits in distributed computation. The first is the teleportation or *teledata* protocol [53], which consumes one ebit to transfer the state of a qubit from one QPU to another.

The second is the EJPP protocol, also known as *telegate* [54]. This protocol enables the remote execution of certain gates, with controlled gates being the most notable example.

The EJPP protocol is executed in three phases, as shown in Figure 4: Initially, during the *starting process* (also known as *cat-entangler* phase), the state of the compute qubit from one QPU, serving as the control, is temporarily mirrored onto the communication qubit of the ebit associated with the compute qubit on the other QPU. Then, a controlled gate is executed locally between the mirrored communication qubit and the target compute qubit. Finally, in the *ending process* (also called the *cat-disentangler* phase), the state of the control compute qubit is decoupled from the communication qubit.

An advantage of the EJPP protocol over teledata is that, with the proposed embedding extension [55], multiple controlled gates can be executed using only a single ebit, provided no interrupting gates are present—such as an intermediate Hadamard gate on the control qubit of two CNOT gates or a pair of CNOT gates acting on the same qubits with flipped control and target. As a result, EJPP enables remote execution of a CU operation involving a complex multi-qubit U with just one ebit, even if U is decomposed into multiple simpler controlled gates.
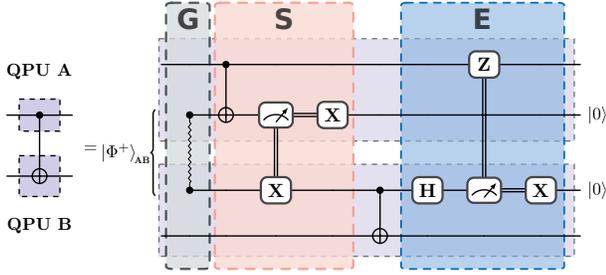
Fig. 4: The steps involved in conducting a remote CNOT operation between QPUs A and B using the EJPP protocol: ebit generation (G), the EJPP starting phase (S), local execution of the CNOT gate, and the EJPP ending phase (E).

Term/Symbol	Definition
N	Number to be factored
n	Number of bits of N i.e. $\lceil \log_2 N \rceil$
Gate delay t_U	Time for an input state to transition a gate U
Circuit graph	A directed acyclic graph (DAG) modelling gate dependencies
Weighted circuit graph	Weighted directed acyclic graph (WDAG) extending the DAG with a weight function representing gate delays
Critical path	Path with longest delay, determining the delay of the overall circuit
Circuit depth	Length of critical path in the circuit DAG
Path delay t_P	Combined signal delay along a path P of gates in the WDAG
Circuit delay t_C	Path delay of critical path

TABLE I: Abbreviations and adopted notation from static timing analysis [21], [22].

D. Static timing analysis

Currently, the time analysis of quantum circuits is primarily approached from the standpoint of computational complexity, i.e., the scaling of algorithms in terms of O-notation [56]. However, as quantum hardware advances, a more practical approach to time analysis is becoming essential. The supported gate set and corresponding *gate times* vary depending on the targeted hardware. Additionally, factors like the execution time of measurement and reset operations, as well as qubit *coherence times*, differ depending on the chosen QPU. To assess practical feasibility and optimize circuits effectively, it's important to account for these specific execution times. Some works already consider such hardware timing aspects in the context of circuit cutting [57], circuit compilation [58], and qubit mapping [59]–[61]. With the progress in quantum error correction, timing will become an even more critical consideration.

Further, in the context of distributed quantum computing, the time required for ebit distribution and routing through quantum networks must also be considered [62]–[66]. As quantum systems become larger and more distributed, resources for distribution and the scheduling and synchronization of computing tasks (e.g. with an execution manager [67]) will be necessary. Therefore, simply analyzing the runtime of a circuit from a complexity standpoint is no longer sufficient. This challenge is not new: classical circuits are also evaluated based on their theoretical complexity, but must be examined with specific timing constraints for practical circuit design [21], [22]. In this work, we adopt definitions of classical circuit design automation, similar to how definitions of classical circuit complexity are used in quantum circuit complexity [56]. A summary of definitions and notation is given in Table I.

Definition 1 (Gate Delay). *The gate delay is defined as the time required to transform the state of n qubits (where $n \geq 1$) when applying an n-qubit gate U and denoted as t_U.*

In classical circuits, the delay of a logical gate refers to the time it takes for a signal to pass through it. In quantum circuits, we define such delay as the gate time of the native gate set supported by the targeted QPU.

Definition 2 (Circuit Graph). *A circuit graph is a directed acyclic graph (DAG), where each vertex $v \in V$ represents*

a gate, and an edge $(u, v) \in E$ indicates that gate v is a direct successor of gate u in a given quantum circuit C, and denoted as $G_C = (V, E)$. The set V also includes two additional vertices, Sc and Sk, which have no predecessor and successor nodes, respectively, to mark the start and end of the circuit execution.
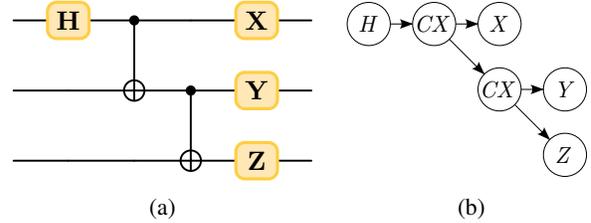

Fig. 5: An example quantum circuit (a) and corresponding DAG representation (b). The DAG has two longest paths: H⤳CU⤳CU⤳Y and H⤳CU⤳CU⤳Z.

Figure 5 illustrates an example circuit along with its corresponding DAG. The *depth* of a circuit is typically represented as the number of distinct time steps required to execute all circuit instructions. In the DAG representation, this can be defined more precisely.

Definition 3 (Critical Path, Circuit Depth). *The critical path is the longest sequence of vertices of the form $Sc \rightsquigarrow Sk$ that are connected by edges for a given circuit graph G_C and the length of the critical path is referred to as the circuit depth. The critical path is not necessarily unique.*

This aligns with the intuitive definition, as each vertex represents an instruction and each edge indicates a direct dependency. Consequently, all instructions on critical paths must be executed sequentially in separate time steps. Since the instructions on critical paths determine the overall execution time of the circuit, they act as a bottleneck in the computation. Understanding how the structure of a circuit defines these critical paths can provide valuable insights for optimization.

Definition 4 (Weighted Circuit Graph). *The weighted circuit graph is a circuit graph $G_C(V, E)$, with edges like (u, v) weighted according to the gate delay t_v of vertex v and*

denoted by $G_W(V, E, w)$. For all edges like (u, Sk) where Sk indicates the end of execution, the weight w is set to 0.

The WDAG not only represents the dependencies between gates but also incorporates their associated delays. Figure 6 illustrates an abstract Shor circuit along with its corresponding DAG and WDAG.

Definition 5 (Path Delay). *The path delay is the sum of weights of the edges along a given path P in a weighted circuit graph $G_W(V, E, w)$ and denoted as t_P.*

Definition 6 (Circuit Delay). *The circuit delay is the critical path delay in a weighted circuit graph $G_W(V, E, w)$ and denoted as t_C. For a sequence of operations $U_1, \ldots, U_k$ forming a circuit C we also denote the circuit delay $t_C = t(U_1 U_2 \ldots U_k)$. The circuit delay is bound by the sum of the delay of all its operations i.e. $t_C \leq \sum_{i=1}^{k} t_{U_i}$, where the inequality becomes an equality only if the U_i operations form a path.*

Circuit delay, like circuit depth, measures a circuit's execution time but does so with greater accuracy for specific hardware by considering gate execution times. This is important because gate delays influence path delays and can alter the longest path within the circuit. Consequently, the critical path identified in the DAG representation may not align with that of the WDAG.
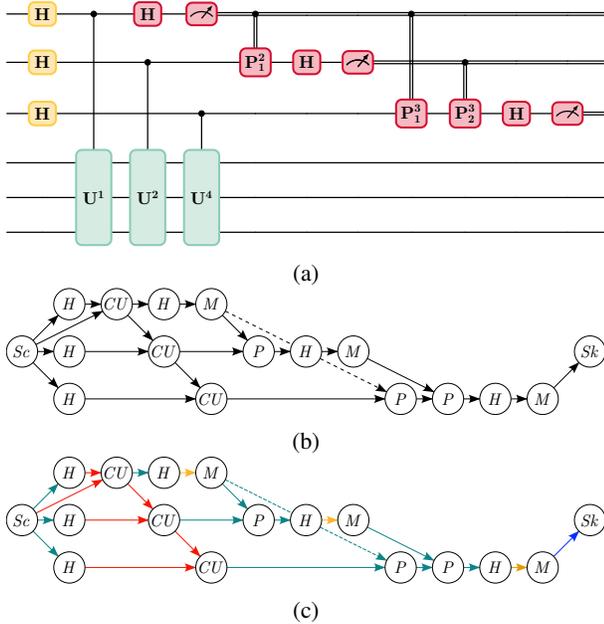

(a)

(b)

(c)

Fig. 6: Abstract Shor circuit (a) and corresponding DAG (b) and WDAG (c) representation. The colors in the WDAG represent the magnitude of gate delays.

III. PARALLELIZATION OF SHOR'S ALGORITHM

A. Task Abstraction

The quantum order-finding subroutine of Shor's algorithm can be viewed from the perspective of quantum phase estimation. Specifically, consider an n-qubit unitary operator U with eigenvalues $\lambda_j = e^{2\pi i \theta_j}$ and corresponding eigenstates $|\psi_j\rangle$. The objective of QPE is to estimate θ_j given an eigenstate $|\psi_j\rangle$. Since $|\psi_j\rangle$ is an eigenstate, it satisfies the relation $U |\psi_j\rangle = \lambda_j |\psi_j\rangle$. By employing a controlled-U (CU) gate and preparing the control register in the state $|+\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)$, the eigenvalue can be moved to the control register using phase kickback:

$$CU(|+\rangle |\psi_j\rangle) = \frac{1}{\sqrt{2}}(|0\rangle |\psi_j\rangle + \lambda_j |1\rangle |\psi_j\rangle) \qquad (3)$$

By applying a Hadamard gate to the control register and then measuring it, we can extract λ_j and consequently determine θ_j, provided $\lambda_j \in \{\pm 1\}$. For higher resolution of θ_j, i.e., $\theta_j = \frac{k}{2^m}$, the approach can be extended by using an m-qubit control register $|l\rangle$ and replacing the CU gate with an m-controlled-U ($C_m U$) gate such that:

$$C_m U : |l\rangle |\psi_j\rangle \mapsto |l\rangle U^l |\psi_j\rangle \qquad (4)$$

The desired value of θ_j is obtained by performing $QFT^\dagger$ on the control register, followed by measurement. The general structure of the algorithm is depicted in Figure 7.
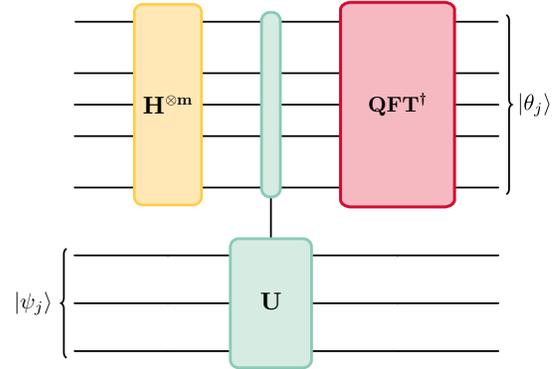

Fig. 7: A high-level circuit diagram of quantum phase estimation (QPE). Applying QPE on the eigenstate $|\psi_j\rangle$ yields the phase θ_j of eigenvalue $\lambda_j = e^{2\pi i \theta_j}$.

In Shor's algorithm for solving the order-finding problem to factor a number N, the circuit can be adapted by choosing the U operation as:

$$U_a : |x\rangle \mapsto |x \cdot a \mod N\rangle \qquad (5)$$

Then $C_m U_a$ is equivalent to U_f from Equation 1. In this setup, the m-qubit control register $|l\rangle$ corresponds to the data register $|r_d\rangle$, while the bottom register holding the eigenstate $|\psi_j\rangle$ corresponds to the work register $|r_w\rangle$.

Further, the state preparation of $|\psi_j\rangle$ can be avoided in this circuit formation. This is because the state $|1\rangle$ is an equal superposition over all eigenstates of U_a. To extract the order r, any θ_j where j is coprime to r can be used. Consequently, initializing the work register in the state $|1\rangle$ and running the algorithm effectively results in a random selection of one of the eigenstates.

For our proposed circuit optimization schemes an unraveling of U_a is not necessary. The designs make use of the inherent structure of QPE and can, in principle, be applied to any QPE-based application. Specifically, in the context of Shor's

algorithm, these optimizations can be integrated with any modular adder circuit or other enhancements, as long as they do not interfere with the circuit's higher-level abstraction.

The C_mU operation from Equation 4 can be decomposed as follows: Since $l = \sum_{i=0}^{n-1} l_i 2^i$ is represented in binary on the data register, each bit l_i accounts for controlling 2^i repetitions of U, i.e., a CU^{2^i} operation. Therefore, the C_mU operation can be expressed as a sequence of m gates acting on the work register, which is initialized in the state $|1\rangle$. Each CU^{2^i} gate is controlled by a separate qubit from the date register $|r_d\rangle$, as shown below:

$$C_mU(|l\rangle\,|\psi\rangle) = \prod_{i=0}^{n-1} CU^{2^i}(|l\rangle\,|\psi\rangle) \tag{6}$$

The $QFT^\dagger$ has a sequential structure. For each qubit q_j, starting from the first qubit q_0 to the last qubit q_m, a phase correction operation P_j is applied, followed by a Hadamard gate. The phase correction operation P_j on qubit q_j involves a set of controlled phase rotations $R_k^{-1} = P(-2\pi i/2^k)$ determined by the states of the preceding qubits, specifically:

$$P_j = \prod_{k=1}^{j} R_{k+1}^{j-k} \tag{7}$$

where R_k^l is the phase rotation R_k controlled by qubit q_l. Applying $QFT^\dagger$ to the data register $|r_d\rangle$ can be expressed as:

$$QFT^\dagger\,|r_d\rangle = \prod_{i=0}^{m-1} P_i H_i\,|r_d\rangle \tag{8}$$

where H_i denotes the Hadamard gate applied to qubit q_i. This standard approach is referred to as the *regular* design and Figure 8 shows an exemplary circuit for $m = 3$.
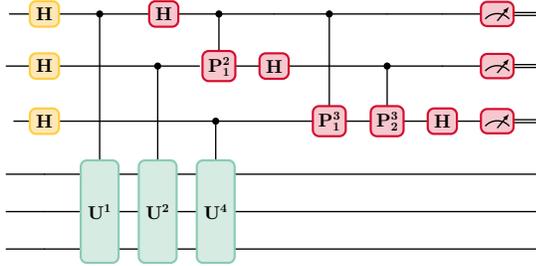

Fig. 8: The high-level circuit for the regular design of Shor's algorithm, implemented as QPE with a 3-qubit data register (i.e., $m = 3$). Each data qubit controls the execution of a specific modular exponentiation operation of the form U^{2^i}, followed by phase processing as part of the $QFT^\dagger$.

To summarize, the main operations for the work register involve the m CU^{2^i} gates, which are executed sequentially. Each qubit in the data register controls exactly one of these gates and then undergoes a phase correction step influenced by all preceding qubits.

B. Monolithic Parallelization

An effective optimization technique to reduce the qubit count in Shor's algorithm is to use the semi-classical implementation of the QFT [37]. This approach leverages the principle of deferred measurement [24]. Specifically, after applying the final Hadamard gate to a qubit, the qubit can be measured immediately, as all subsequent operations are controlled gates that can be implemented using classical controls. This process is illustrated in Figure 9.
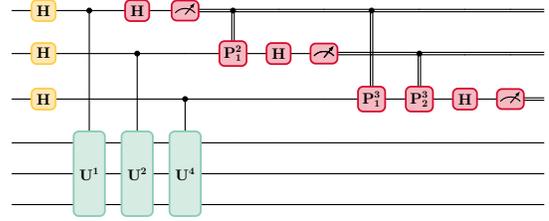

Fig. 9: The regular design of Shor's algorithm utilizing the semi-classical $QFT^\dagger$. All two-qubit gates in $QFT^\dagger$ acting on 3-qubit data register (i.e., $m = 3$) are replaced with classically controlled single-qubit gates.

The IPE [39], [40] incorporates the semi-classical QFT within QPE. In this approach, once a data qubit has been used as a control for its corresponding CU^{2^i} operation, completed its phase processing, and its phase has been measured, it is no longer needed. This allows the qubit to be reset and reused for the next CU^{2^i} operation. As a result, the entire data register can be implemented using just a single qubit. The iterative process can be represented as follows:

$$\prod_{i=0}^{m-1} HCU^{2^i} P_i HMR \tag{9}$$

Here, the index of the Hadamard gate H is omitted since only one qubit is used for the data register. M denotes measurement in the computational basis, R represents the qubit reset, and P_i corresponds to the phase correction as described in Equation 7. This method is known as the *iterative* realization of Shor's algorithm. An example circuit for the case $m = 3$ is shown in Figure 10.
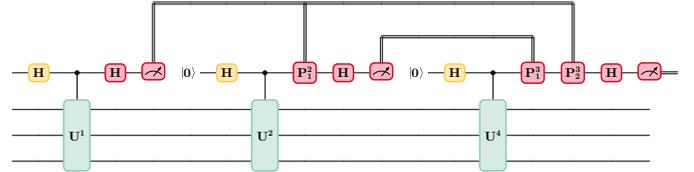

Fig. 10: The iterative design of Shor's algorithm with a 3-qubit data register (i.e., $m = 3$) implemented as IPE. The data register is reduced to a single qubit at the expense of longer circuit delay.

Although the iterative realization uses only one qubit instead of m qubits for the data register, it results in a longer execution time. As outlined in Equation 9, each operation must be performed sequentially: starting with the Hadamard initialization, followed by the controlled modular arithmetic CU^{2^i}, phase correction P_i, a second Hadamard gate, measurement, and finally a qubit reset. This sequential execution creates a critical path that traverses all CU^{2^i} operations for $i = 0, 1, \ldots, m-1$, along with the phase correction steps. Throughout this process, the work register remains active

only during the CU^{2^i} operations, leading to idle periods, as illustrated in Figure 11.
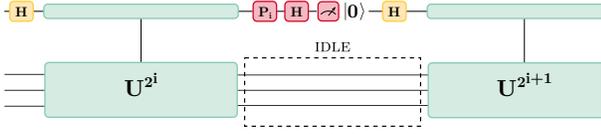

Fig. 11: The idle time experiencing in the work register of the iterative design of Shor's algorithm due to phase processing on the data qubit as part of the semi-classical $QFT^\dagger$ operation.

In contrast, in the regular realization, once a CU^{2^i} operation is completed on the work register, the next $CU^{2^{i+1}}$ operation can proceed without waiting for phase processing, measurements, resets, and reinitialization. Consequently, the regular approach enables parallel execution of computational tasks, reducing the overall processing time, at the cost of more qubits.

To address this issue, the *alternating* design combines qubit resets and the semi-classical $QFT^\dagger$, similar to the iterative approach, but uses two data qubits instead of one to enable parallelization. The key idea is to alternate the iterative process between the two data qubits. After the work register is initialized and the first data qubit undergoes the initial Hadamard gate, i.e., H_0, the first CU^1 operation is executed using this qubit and the work register. While CU^1 is being executed, the second qubit is initialized with H_1. Once CU^1 is complete, the next operation CU^2 is executed using the second qubit and the work register. During CU^2, the phase information is extracted from the first qubit, which is then reset and is reinitialized with H_0 and becomes ready for the next operation CU^4 as soon as CU^2 finishes. This alternating pattern continues until all CU^{2^i} operations are completed, effectively interleaving the usage of the two data qubits to achieve parallelism and minimize idle time in the work register observed in the corresponding iterative design (see Figure 11), thereby enhancing overall execution time. Figure 12 illustrates the design of Shor's algorithm using the proposed alternating approach.
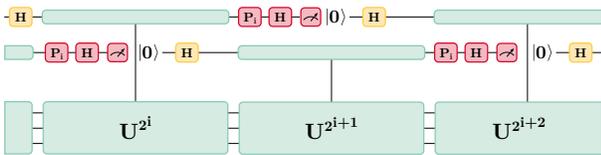

Fig. 12: The alternating design of Shor's algorithm reducing the idle time experienced in the work register of the corresponding iterative design by introducing an additional data qubit to alternate the phase processing tasks.

The potential timing advantage of the alternating approach depends on the duration of the CU^{2^i} operations and the phase processing gates. If the CU^{2^i} operations are short, there may not be enough time to simultaneously extract the phase and reset the other data qubit, resulting in some idle time — although shorter than the idle time observed in the iterative

design. Conversely, if the CU^{2^i} gates are sufficiently long and satisfies the condition:

$$t_{CU^{2^{i+1}}} \geq t(P_i HMRH) \quad \forall i \tag{10}$$

then there is no delay in the work register because, during each CU^{2^i} operation, there is sufficient time to extract the phase, reset, and reinitialize the other qubit. Consequently, the critical path consistently runs through the work register without being affected by the data register.

To understand the amount of idle time that can be reduced, both by the alternating and regular design, we first split the overall circuit delay t_C into three parts:

$$t_C = t_H + \sum_{i=0}^{m-1} t_{CU^{2^i}} + \delta_P \tag{11}$$

The initial Hadamard as well as all CU^{2^i} operations have to be executed sequentially, so their delays will always contribute to the overall circuit delay. The additional delay based on phase processing, denoted as δ_P, varies based on the choice of design. We again split δ_P into two parts:

$$\delta_P = \delta_P^M + \delta_P^{\neg M} \tag{12}$$

where δ_P^M denotes the mitigatable delay and $\delta_P^{\neg M}$ denotes the unavoidable delay due to phase processing. In principle, each phase processing step can be executed in parallel to following $CU^{2^{i+1}}$ gates. The exception is the last phase processing, which incurs an unavoidable delay since no further CU^{2^i} gates remain for overlap:

$$\delta_P^{\neg M} = t(P_{m-1}HMRH) \tag{13}$$

The amount of delay that can be mitigated from all other phase processing steps depends on the choice of design and the length of the CU^{2^i} operations. If the CU^{2^i} operations are long, the critical path goes through the CU^{2^i} gates and no delay of the phase processing contributes to the circuit delay. On the other hand, if the CU^{2^i} operations are instant, all phase processing delay contributes. Therefore, the mitigatable delay due to phase processing is bound by:

$$0 \leq \delta_P^M \leq \sum_{i=0}^{m-2} t(P_i HMRH) \tag{14}$$

As a more relaxed upper limit, the worst-case duration $(m-1)t(P_{m-1}HMRH) > \delta_P^M$ can be considered, which uses the worst-case $t_{P_{m-1}}$ for phase gates, given that each phase correction P_i requires one more adjustment than its predecessor P_{i-1}.

These bounds hold for the alternating as well as the regular design. The concrete difference in circuit delay then depends on the durations of the CU^{2^i} operations. Longer delays in CU^{2^i} operation create more opportunities for alternating parallelization while maintaining an overall circuit delay comparable to the regular design. In the worst-case scenario, if the CU^{2^i} delay significantly exceeds the upper limit $(m-1)t(P_{m-1}HMRH)$, the idle time becomes negligible. Consequently, the overall circuit delays for all three approaches converge, with a slight preference for the iterative design due to its use of one fewer data qubit compared to the alternating approach.

C. Distributed Shor

Several early studies have explored distributed implementations of Shor's algorithm. One of the most comprehensive early works [68] presents a distributed approach for the regular design using VBE [7] adders in the modular exponentiation circuits. This approach assumes a setup with multiple nodes, each containing $n+c$ qubits, where c represents additional ancilla and communication qubits. Due to space constraints, the data register — comprising $m = 2n$ qubits — is divided into two blocks across the nodes. The work register is split into five parts to align with the structure of the VBE implementation.

A comparative study [69] evaluates different adder choices for various distributed settings, including qubit capacities on QPUs, network topology, and I/O capabilities. It also examines the use of EJPP versus teleportation protocols. Another recent work [70] investigates the impact of imperfect ebits in QPE with a distributed QFT using embedding techniques. Additionally, [5] proposes optimizations for monolithic implementations of Shor's algorithm for large numbers and highlights distributed implementations as a potential area for future enhancement. Meanwhile, general approaches to distributed compilation, such as [19], [20], [55], [71], do not consider the specific structure of Shor's algorithm, like the roles of the data and work quantum registers, potentially limiting their optimization effectiveness.
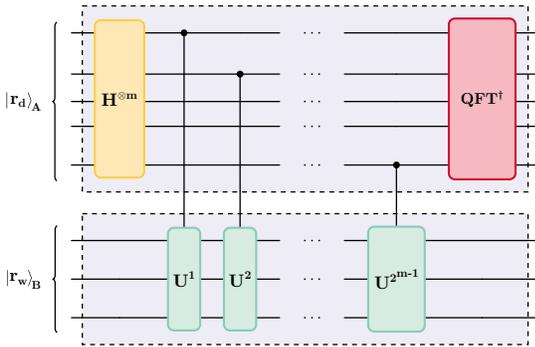

Fig. 13: The distributed design of Shor's algorithm: the data register $|r_d\rangle$ is allocated to QPU A, while the work register $|r_w\rangle$ is allocated to QPU B. The CU^{2^i} operations must be performed remotely between the two QPUs.

In contrast to these previous works, our proposed designs operate at a higher level of abstraction and are agnostic to specific adder choices. As a result, we do not explore partitioning the work register. Instead, our approach places the data register on one QPU and the work register on another (see Figure 13). This configuration maintains flexibility, allowing our proposed distribution designs to be combined with further subdivisions of the work register at lower abstraction levels.

D. Distributed Parallelization

In Shor's algorithm, since the $QFT^\dagger$ operates entirely on the data register, the primary task for distribution involves modular exponentiation, which affects both the data and work registers. According to the $C_m U$ decomposition from Equation 6, the $2n$ CU^{2^i} gates, each controlled by a single qubit on the data

register, must be executed across separate QPUs. Teleporting the control qubits to the work QPU and then back to the data QPU would be costly, requiring two ebits per CU^{2^i} gate, amounting to a total of $2m$ ebits. Instead, we use the EJPP protocol to perform each CU^{2^i} operation remotely. This process consists of three steps: (i) Applying a starting process S, which involves the control data qubit for the CU^{2^i} gate on QPU A and a shared ebit between QPUs A and B, (ii) Executing the CU^{2^i} gate locally on the work register located on QPU B using the shared ebit, and (iii) Completing the remote operation with an ending process E, which uses the control qubit from QPU A and the shared ebit. The entire process can be represented as:

$$CU\,|r_d\rangle\,|r_w\rangle \rightarrow E(CU)S\,|r_d\rangle\,|\phi^+\rangle\,|r_w\rangle \qquad (15)$$

By utilizing embedding, only one ebit is needed for each CU^{2^i} operation, requiring a total of m ebits, even when the CU gates are decomposed into native gates. The S and E operations involve only a few basic gates and introduce minimal delay. However, generating an ebit (G) between two QPUs can be time-consuming depending on the hardware and distance. Since the distribution setup requires m ebits, multiple ebit generation cycles will be necessary during the distributed execution of the algorithm.
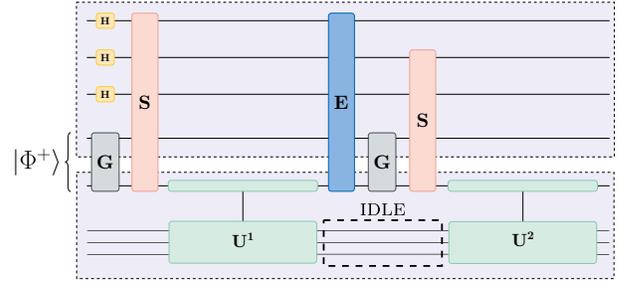

Fig. 14: The regular distributed design of Shor's algorithm results in idle time between two CU^{2^i} operations due to the usage of a single ebit channel. The idle period occurs due to the the end process (E) of the previous CU^{2^i} operation, as well as the ebit generation (G) and start process (S) for the next $CU^{2^{i+1}}$ operation.

Further, if only a single ebit channel is available between the two QPUs before each CU^{2^i} operation, the previous EJPP protocol must be completed with an ending process E, a new ebit is then generated over the ebit channel, and the starting process begins the next EJPP protocol. Similar to the iterative monolithic design, the work register remains idle due to the inclusion of processes G, E, and S in the critical path. Figure 14 illustrates the regular distributed design of Shor's algorithm and highlights the instances of idle time that occur during the scheduling of CU^{2^i} operations on the work register when only a single ebit channel is used.

When multiple ebit channels are available, distribution tasks can be parallelized, as illustrated in Figure 15 for the regular design. While one CU^{2^i} operation is in progress, the ebit for the next $CU^{2^{i+1}}$ can be generated, and the starting process for the next operation can be executed. Since the next $CU^{2^{i+1}}$
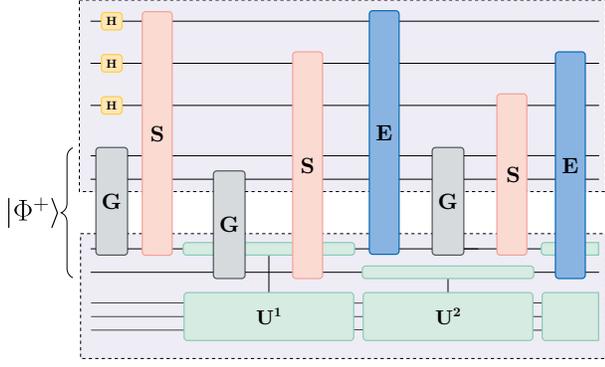
Fig. 15: The regular distributed design of Shor's algorithm utilizing two ebit channels to eliminate idle time in the work register caused by ebit generation (G), start process (S), and end process (E). The idle period reduction is achieved by alternating the two ebit channels, allowing the scheduling of distribution processes to occur simultaneously to CU^{2^i} operations.

works on a different data qubit, it can proceed while the current CU^{2^i} is running. Furthermore, the next $CU^{2^{i+1}}$ does not need to wait for the ending process of the previous one, as it can be processed in parallel. By alternating between two ebit channels, similar to the alternating design for the monolithic setup, the idle time in the work register can be reduced.

The benefit of using more ebit channels depends on both the delay of the CU^{2^i} operations and the time required for ebit generation. If the ebit generation process is too slow, it cannot run fully in parallel with a CU^{2^i} gate and will delay the next $CU^{2^{i+1}}$ operation. With two ebit channels, during the time of one CU^{2^i} gate, the start process (S), ebit generation (G), and end process (E) must all be completed to avoid idle time. With three ebit channels, the next ebit must be ready after execution of two gates, CU^{2^i} and $CU^{2^{i+1}}$. Similarly, with four ebit channels, the next ebit must be ready after three gates, CU^{2^i}, $CU^{2^{i+1}}$, and $CU^{2^{i+2}}$ and so on. This sets the condition for zero idle time with k ebit channels, meaning that a $CU^{2^{i+k}}$ operation will not be delayed if the full distribution block (G, S, E) has been executed during all the previous $k-1$ gates, CU^{2^i}, $CU^{2^{i+1}}$, ..., $CU^{2^{i+k-1}}$:

$$\sum_{j=0}^{k-1} t_{CU^{2^{i+j}}} \geq t(GSE) \quad \forall i \tag{16}$$

For large m, a looser bound can be derived by using the average CU^{2^i} gate delay $\bar{t}_{CU}$, which approximates Equation 16 for large m:

$$(k-1)\bar{t}_{CU} \geq t(SGE) \tag{17}$$

We investigate the amount of idle time from distribution that can be mitigated by analyzing how the delay of distribution processes contributes to the overall circuit delay of Equation (11):

$$t_C = t_H + \sum_{i=0}^{m-1} t_{CU^{2^i}} + \delta_P + \delta_D \tag{18}$$

The initial Hadamard and the CU^{2^i} operations still have to be executed sequentially, so their delays will always contribute to the overall circuit delay and there can be delay from phase processing. The additional delay based on distribution is denoted as δ_D and depends on the amount and speed of EJPP channels used, as well as the duration of CU^{2^i} operations. We split δ_D into mitigatable delay δ_D^M and unavoidable delay $\delta_D^{\neg M}$ due to distribution:

$$\delta_D = \delta_D^M + \delta_D^{\neg M} \tag{19}$$

The ebit generation and start process for the first CU^{2^i} gate will inevitably contribute to the circuit delay, as they can not be parallelized. Similarly, the final termination process cannot be executed concurrently with the last $CU^{2^{m-1}}$ gate. Combined, the unavoidable delay is equal to the delay of one full distribution block (G, S, E):

$$\delta_D^{\neg M} = t(GS) + t_E = t(GSE) \tag{20}$$

For the other distribution operations, if the other operations are long enough and the critical path goes through the CU^{2^i} gates or phase processing, no delay of the distribution operation processing contributes to the circuit delay. On the other hand, if the CU^{2^i} and phase processing operations are instant or not running in parallel, all distribution delay contributes. Therefore, the mitigatable delay from distribution processes is bound by:

$$0 \leq \delta_D^M \leq t_E + \sum_{i=1}^{m-2} t(GSE) + t(SG)$$
$$= (m-1)t(GSE) \tag{21}$$

In principle, these bounds apply to all setups with multiple ebit channels and all three monolithic designs (i.e., regular, iterative, and alternating), though for the iterative design, the start and end processes cannot be parallelized because all operations (S, E, and CU^{2^i}) act on the same single data qubit. It is important to note, that only in the case of the iterative design with a single ebit channel (i.e. where every operation is sequential) the phase processing and distribution process delay contribution independently add to the overall circuit delay. However, if multiple data qubits are used, phase processing can run in parallel with ebit generation and CU gates on the work register, reducing the overall idle time. Similarly, when multiple ebit channels are used, distribution process can be parallelized and the delay contributions δ_P and δ_D are highly dependent on one another. In these cases, the total idle time depends on which operation becomes the bottleneck in the computation, determining the critical path.

IV. METHOD

To evaluate the impact of our proposed design approaches on circuit delay, we selected modular exponentiation circuits from available resources and implemented them using the Qiskit framework [72]. Since the performance of the designed circuits depends on the latency of primitive operations supported by the targeted hardware, we constructed both monolithic and distributed quantum computing models in accordance with existing literature and available quantum systems [23], [25]–[27], [44], [45], [49], [62], [73]–[95].

A. QPU Modeling

For both monolithic and distributed setups, we adopted a generic basis gate set $\{I, X, H, P(\theta), CX\}$, requiring transpilation only for gates with multiple controls and SWAP gates. Additionally, we did not impose a specific qubit topology, eliminating the need for extra SWAP gates to enable two-qubit interactions. The discussed parallelization approaches are compatible with any quantum hardware platform. Three widely used platforms were selected for the experiments — superconducting, ion-trap, and neutral-atom-based QPUs — where ebit generation has been experimentally demonstrated and dynamic circuits are supported. For superconducting QPUs, we used IBM Eagle and IBM Heron characteristics as benchmarks for gate execution times, measurements, and resets, leveraging their accessibility through the IBM Quantum Platform [73]. To map gate times to our chosen generic basis gate set, we computed the average single-qubit gate time (t_{q_1}), and two-qubit gate time (t_{q_2}). A detailed list of extracted backend properties is provided in Table II.

Superconducting Backend Properties				
Backend	t_{q_1}	t_{q_2}	t_{measure}	t_{reset}
Eagle sherbrooke	57 ns	533 ns	1216 ns	1276 ns
Heron r1 torino	32 ns	68 ns	1560 ns	1708 ns
Heron r2 fez	24 ns	84 ns	1560 ns	1584 ns
Heron r2 marrakesh	36 ns	68 ns	2100 ns	2236 ns

TABLE II: Average delays for single-qubit gates (t_{q_1}), two-qubit gates (t_{q_2}), as well as reset (t_{reset}) and measurement (t_{measure}) times for superconducting QPUs from IBM [73].

For ion-trap QPUs, we similarly extracted execution times for the IonQ Aria and IonQ Forte systems from the IonQ cloud [74], as reported in Table III.

Ion-Trap Backend Properties				
Backend	t_{q_1}	t_{q_2}	t_{measure}	t_{reset}
IonQ Aria 1	135 μs	600 μs	300 μs	20 μs
IonQ Aria 2	135 μs	600 μs	50 μs	15 μs
IonQ Forte	130 μs	970 μs	150 μs	50 μs

TABLE III: Average delays for single-qubit gates (t_{q_1}), two-qubit gates (t_{q_2}), as well as reset (t_{reset}) and measurement (t_{measure}) times for ion-trap QPUs from IonQ [74]

Regarding neutral atom QPUs, while QuEra systems [75] are available via cloud access, they are restricted to analog quantum computing and do not support quantum circuits. Therefore, we based our model (see Table IV) on recent advancements in gate-based quantum computing from the literature [44], [45], [49], [76]–[83].

Finally, we picked one representative system from each platform for further analysis (see Table V): IonQ Forte for ion-trap QPUs, IBM Heron (ibm_torino) for superconducting QPUs, and a neutral-atom QPU based on the reports in [49][2].

B. DQC Modeling

As there are is no DQC hardware publicly available yet, we cannot directly extract execution times for distributed

[2]The reset duration for the neutral atom QPU model is modeled as the combined time of a measurement followed by a single-qubit gate, as no specific values have been reported.

Neutral Atom Backend Properties				
Backend	t_{q_1}	t_{q_2}	t_{measure}	t_{reset}
[76]	250 ns/4.1 μs	416 ns	6 ms	N/A
[49]	$\sim$2 μs	$\sim$400 ns	$\sim$10 ms	N/A
[80]	N/A	$\sim$250 ns	N/A	N/A
[81]	N/A	$\sim$110 ns	N/A	N/A
[82]	N/A	$\sim$6.5 ns	N/A	N/A
[45]	N/A	N/A	4 ms	N/A

TABLE IV: Reported average delays for single-qubit gates (t_{q_1}), two-qubit gates (t_{q_2}), as well as reset (t_{reset}) and measurement (t_{measure}) times for neutral atom quantum computers in the literature.

Backend Properties				
Backend	t_{q_1}	t_{q_2}	t_{measure}	t_{reset}
IonQ Forte	130 μs	970 μs	150 μs	50 μs
IBM Heron	32 ns	68 ns	1560 ns	1708 ns
Neutral Atom	2 μs	400 ns	10 ms	10.002 ms

TABLE V: Average delays for single-qubit gates (t_{q_1}), two-qubit gates (t_{q_2}), as well as reset (t_{reset}) and measurement (t_{measure}) times for the selected experimental setups: Ion-trap (IonQ Forte), superconducting (IBM Heron), and neutral atom setups chosen for experiments.

operations. Instead, we have to rely on models based on experimental DQC setups. We assume a distribution model where QPUs are interconnected via quantum and classical channels, with ebits facilitating non-local gate execution. Our parallelization approach requires only two QPUs and does not assume a specific network architecture. For simplicity, we consider a direct connection between the two systems, eliminating the need for routing through quantum nodes, quantum repeaters, or entanglement swapping.

Since Qiskit does not yet support a distributed circuit model, we constructed circuits in which qubits are divided into subsets corresponding to the first and second QPU, along with qubit pairs designated for ebit channels. The ebit generation process is modeled using a Hadamard and a CNOT gate, with modified execution times ($t_{\text{ebit}_H}, t_{\text{ebit}_{CX}}$) representing the ebit generation time (t_{ebit}).

The ebit generation between distant QPUs has been experimentally demonstrated using photonic quantum communication, coupled with computing platforms based on atoms [84]–[88], ion traps [89], [90], [96], diamonds [91], and superconductors [92]–[94]. Based on these studies, we calculated a realistic range of ebit generation times (t_{ebit}) for our three chosen hardware platforms, as shown in the following paragraphs.

Our long-range ebit generation model is based on [62], which assumes a neutral-atom platform. In this model, we assume a local heralded entanglement generation between a telecom photon and a local qubit for both QPUs. The probability of successful local entanglement generation is given by:

$$p = p_{ht}\nu_h\nu_t \quad (22)$$

where p_{ht} is the photon generation probability, ν_h is the heralding- and ν_t the entangling detector efficiency. After both QPUs send their entangled photons, a photonic Bell state measurement (BSM) at a midpoint between them generates

the end-to-end ebit. The success probability of this process is:

$$p_e = \frac{1}{2}\nu_o p^2 e^{\frac{-d}{L_0}} \tag{23}$$

where ν_o is the optical BSM efficiency, d is the link length, and L_0 is the attenuation length of the optical fiber. The average time for a successful (T_s) and failed (T_f) entanglement attempt is given by:

$$T_s = \tau_p + \max\{\tau_h, \tau_t + \frac{d}{c_f} + \tau_o\} \tag{24}$$

$$T_f = \tau_p + \max\{\tau_h, \tau_t + \frac{d}{c_f} + \tau_o, \tau_c\} \tag{25}$$

where τ_p is the time required to excite the qubit, τ_h is the herald-cavity time, τ_t is the telecom-cavity time, τ_o is the optical BSM time, c_f is the speed of light in the optical fiber, $\frac{d}{c_f}$ accounts for the time it takes for the telecom photons to reach the midpoint and for the classical acknowledgement being sent back from the BSM, and τ_c represents the additional time required for a local qubit reset in case of a failed entanglement attempt.

Overall the expected entanglement generation time is then calculated as:

$$T = \frac{p_e T_s + (1 - p_e)T_f}{p_e} \tag{26}$$

For neutral-atom systems, we use parameter values commonly found in the literature [84], [85]:

$$p_{ht} = 0.53, \quad \nu_h, \nu_t = 0.8, \quad \nu_o = 0.39,$$
$$L_0 = 22km, \quad \tau_o = \tau_a = \tau_t = 10\mu s, \quad c_f = 2\cdot 10^8 m/s,$$
$$\tau_p = 5.9\mu s, \quad \tau_h = 20\mu s, \quad \tau_d = 100\mu s$$

For distances ranging from 1–50 km, these parameters yield ebit generation times between 5–115ms.[3] A similar heralding approach has been demonstrated for superconducting systems [93], with reported ebit generation times between 10–1000μs. For ion trap QPUs, ebit generation times of approximately $\sim 5.5ms$ have been observed over short distances [90], while for longer distances (up to 230m), the reported range is 2-17s [89].

These values represent optimistic estimates, as our model does not account for additional factors such as entanglement swapping, quantum repeaters, or entanglement purification, all of which would reduce the success probability of end-to-end ebit generation and increase the overall generation time.

C. Modular Exponentiation Circuits

Only a limited number of circuits are available for Shor's algorithm [23]. The circuits presented in [25]–[27] are manually designed for small values of N, optimized for minimal width and depth to enable early demonstrations on NISQ hardware. Additionally, MQT Bench [95] provides fixed Shor circuits for $N = 9, 15, 821$ as benchmarks for design evaluations.

We used the QRISP [23] framework, which offers a high-level interface for generating circuits for arbitrary N with

various arithmetic adders [8], [11], [12], [97]. For our experiments, the default Fourier-adder [12] was used. Since the proposed approaches are independent of specific arithmetic implementations, they can seamlessly integrate with any of the mentioned circuits. For the optimized circuits, CU operations were manually designed based on the design details provided in the relevant papers. In the case of QRISP, the underlying factorization function was used to extract CU operations, which were then incorporated into the Qiskit [72] environment. In general, the choice of the parameter a in the modular exponentiation operation can influence both circuit depth and execution time. However, the parameter a was consistently set to 2 in our evaluation to keep the number of experiments manageable.

D. Delay Calculation

To determine the circuit delay in practice, a dummy *source* (Sc) and *sink* (Sk) vertex are added in the beginning and end of the WDAG, respectively, and the longest path from source to sink ($Sc \rightsquigarrow Sk$) is computed. Unlike the WDAG definition, weighted graphs for path calculations typically assign weights to edges. However, without loss of generality, these weights can be transferred from a vertex v's instruction to all its incoming edges $(., v)$, as every path passing through v must traverse at least one of its incoming edges. Since the graph is acyclic, a path will use at most one incoming edge [4].

For a DAG, the longest path can be efficiently determined using a topological sorting of its vertices. When generating quantum circuits programmatically (e.g., in Qiskit), operations are typically appended sequentially to a queue-like list, inherently forming a topological order of V. If the circuit is structured in this manner, its longest path can be computed directly. The space complexity of this calculation is $O(V)$, as it requires storing each vertex's longest distance from the source and its predecessor along the longest path. However, if only the path length is needed, storing just the distance suffices. In quantum circuits, further optimization is possible, i.e., instead of storing distances for all vertices, only distances associated with individual qubits and classical bits need to be maintained. Since each operation can act on at most all available bits, the number of incoming edges is limited to the number of qubits and classical bits. By tracking which bits correspond to an edge, we can further reduce storage requirements. Algorithm 1 presents this space-efficient approach for computing circuit delay.

In our experiments, we applied different weight mappings for this delay calculation. Specific instructions such as reset, measure, and ebit-related operations (ebit$_h$, ebit$_{CX}$) were assigned distinct weights, while all other instructions were weighted using the average values t_{q_1} and t_{q_2}. Since ebit generation is represented by a Hadamard and CNOT gate rather than a detailed physical model, we set ebit$_h$ = 0 and allocate the entire execution time to ebit$_{CX}$, as outlined

---

[3]We assume no coherence time restrictions, as in [62], which could otherwise further constrain the capacity of an ebit channel.

[4]To accurately represent the weights of the initial instructions, we utilize the dummy source node (s_k). Similarly, the outgoing edges $(v, .)$ can be used to model the weights of the instructions together with the dummy sink node (s_k).

Algorithm 1: Circuit Delay

Input:
  Circuit as topologically sorted list:
  $U = [U_1, \ldots, U_{N_U}]$,
  List of circuit bits: B

Result: Circuit delay t_C

$t[b] = 0, \forall b \in B$

for $i = 1, \ldots, N_U$ **do**
  $B_i \leftarrow U[i].\text{bits}$
  $t_i \leftarrow U[i].\text{delay}$
  $t_{\max} \leftarrow \max(\{t[b], \forall b \in B_i\})$
  foreach $b \in B_i$ **do**
    $t[b] = t_{\max} + t_i$
  end
end

$t_C \leftarrow \max(\{t[b], \forall b \in B\})$

return t_C

---

in subsection IV-B. This ensures a balanced execution time distribution across the two qubits involved in the ebit channel.

To compute execution time, we considered three approaches: circuit moments, a weighted DAG method using the NetworkX library [98], and algorithm 1. While algorithm 1 is the most efficient, circuit moments and the weighted DAG method serve as valuable intermediate representations for circuit compilation tasks, justifying their additional computational cost. For our experiments, algorithm 1 was used.

V. RESULTS

We evaluated our proposed design approach across three distinct hardware platforms: neutral atoms, superconducting qubits, and ion traps. The analysis included two circuit sets: optimized circuits [26], [99] and QRISP-generated circuits [23], with N sizes of up to $n = 64$ bits. For the evaluation, we utilized the QPU parameters of ibm_torino heron QPU for superconducting qubits, the ionq_forte QPU for ion traps, and neutral atom QPU parameters reported in [49].

Key metrics assessed for each circuit included work register size, gate counts (single- and two-qubit gates), and the average delay for executing controlled unitary (CU) operations on the selected platforms. The results are summarized in Figure 16. The effectiveness of the proposed approach was evaluated in both monolithic and distributed setups.

In the monolithic environment, experimental results demonstrated that for all circuits the alternating design: (i) achieved the same circuit depth as the corresponding regular design across all architectures and (ii) required one additional qubit compared to the iterative design. In the distributed environment, experiments revealed that the depth of alternating circuits was significantly influenced by: (i) the availability of ebit channels, and (ii) the ebit generation time, with higher ebit generation times requiring more than two ebit channels to maintain performance. Detailed experimental results are presented below.

A. Analysis in the Monolithic Setup

Initially, in the monolithic environment, we constructed iterative, regular, and proposed alternating circuits for all chosen values of N and calculated their delay across architectures, as shown in Figure 17. For superconducting and ion trap QPUs, the delay of CU operations quickly outpaced the delays of phase processing and qubit resets, even for small N. Consequently, the mitigable idle time formed only a negligible part of the overall circuit delay, resulting in roughly equal performance across all monolithic designs within these architectures.

In contrast, for neutral atoms, even at the largest tested N (i.e., $n = 64$), a significant delay difference persisted between iterative and regular/alternating designs due to Equation 10 being violated. This aligns with expectations, as measurement and reset times for neutral atoms remain orders of magnitude slower than gate operations. Conversely, in superconducting QPUs, measurements and resets are only slightly slower, and in IonQ systems, they are even faster than single- and two-qubit operations. While the iterative design is the most qubit-efficient design, its delay is significantly larger compared to the other two approaches. The alternating design achieved delays equal or close to the regular design but required only one additional qubit compared to the iterative approach. To explore the differences between regular and alternating designs in neutral atoms, we computed the relative delay reduction compared to the iterative approach (see Figure 18). The alternating design consistently reduced delay by 50% up to $n = 50$. The regular design also began with a 50% reduction for small N but peaked at $n = 25$ before converging with the alternating design's performance at $n = 50$. Beyond $n = 50$, both designs exhibited equal and decreasing delay reductions.

This behavior can be understood by breaking the delay into two components: mitigable idle time and non-mitigable operational delay (CU executions and final phase processing). We isolated the mitigable portion by comparing the observed delay reduction to the theoretical bound for idle time, t_{idle} from Equation 14, as shown in Figure 19. For small N, the mitigable phase processing delay dominates, but only a small fraction of idle time can be reduced via parallelization due to short CU operations. As N increases, larger CU operations introduce more delay, enabling both alternating and regular designs to mitigate a greater portion of the idle time. The performance of the regular design peaks at $n = 25$, where it mitigates all idle time, while the alternating design achieves this at $n = 50$.

The trends in Figure 18 reflect this interplay between mitigable idle time and non-mitigable operational delay. For the alternating design, the increase in idle time mitigation matches the increasing CU operation delay, keeping the relative reduction constant. In contrast, the regular design initially mitigates idle time faster than CU delay grows, leading to a greater relative delay reduction until it reaches the limit at $n = 25$. Beyond this, the diminishing significance of idle time compared to CU delay causes a decrease in relative delay reduction.

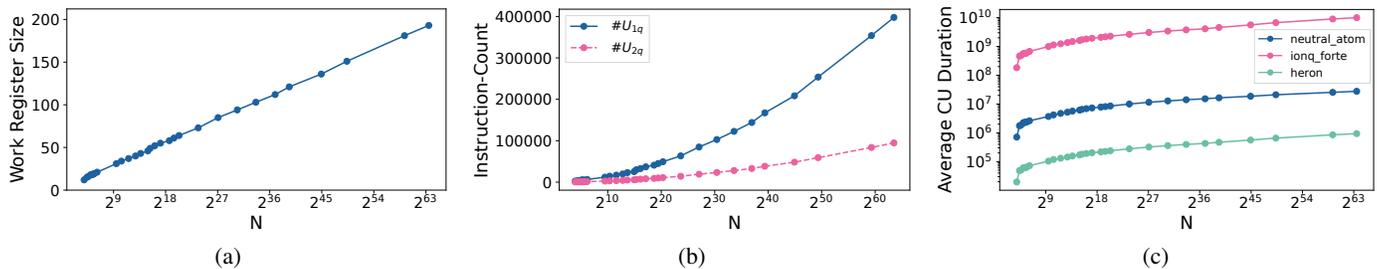These results highlight the importance of tailoring design

Fig. 16: The size of the work register (a), single- and two-qubit gate counts (b) and average delay for the neutral atom, superconducting and ion trap weights (c) for the CU gates of the experiments. The arithmetic circuits were generated using QRISP [23].
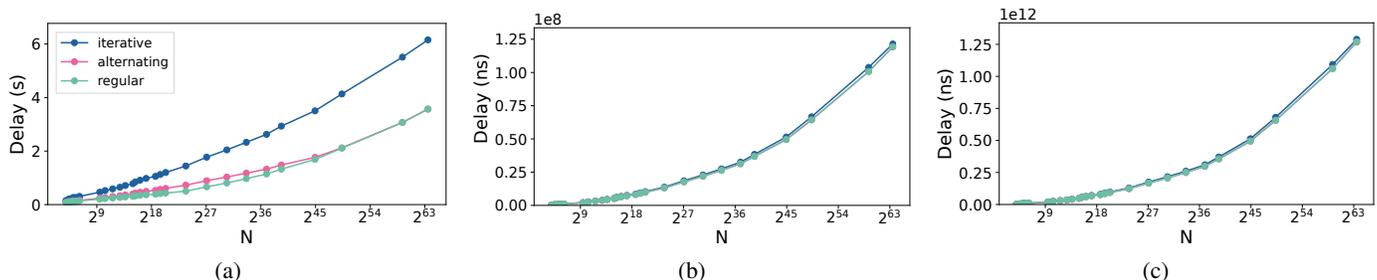
$t(CU)$

Fig. 17: Monolithic delay scaling of the iterative, alternating, regular approach for neutral atom (a), IBM superconducting (b) and IonQ ion-trap (c) weights.
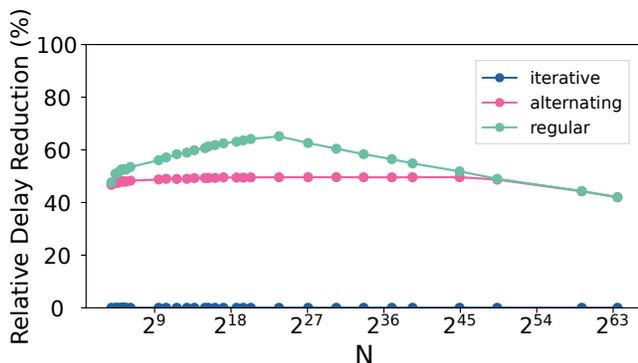

Fig. 18: Relative delay reduction of the alternating and regular design compared to the delay of the iterative design for the neutral atom QPU weighting.

Fig. 19: Relative delay mitigation compared to the idle time bound of Equation 14 for neutral atom QPU weighting.

choices to the QPU characteristics and the size of N. For superconducting and ion trap systems, the iterative design consistently performs best, as it is the most qubit-efficient approach and all designs have similar delay. For neutral atoms, the regular design is advantageous for smaller N, while for larger N, the alternating design offers comparable delay efficiency to the regular design with the added benefit of requiring only one additional qubit over the iterative approach.

B. Analysis in the Distributed Setup

For the distributed parallelization, we extended the monolithic setup by integrating two QPUs connected via one to four ebit channels. The circuits for all the selected values of N from the monolithic experiments were recompiled, with their work and data registers partitioned across separate QPUs. We then evaluated the circuit delay across architectures where the ebit generation times were determined based on the hardware specifications outlined in subsection IV-B.

Figure 20 provides an overview of all conducted experiments. For a given value of N and ebit generation time t_{ebit}, the delay of compiled circuits using one to four ebit channels is compared. A higher number of ebit channels is preferred only if it results in a lower circuit delay compared to configurations with fewer channels, even by a small margin. This approach reflects the high cost of ebit channels, emphasizing that their increased usage should be justified by more than a slight delay reduction.

We can also see from Figure 20 that faster ebit generation times require fewer ebit channels, as the same ebit generation rate can be achieved with fewer channels. Additionally, for
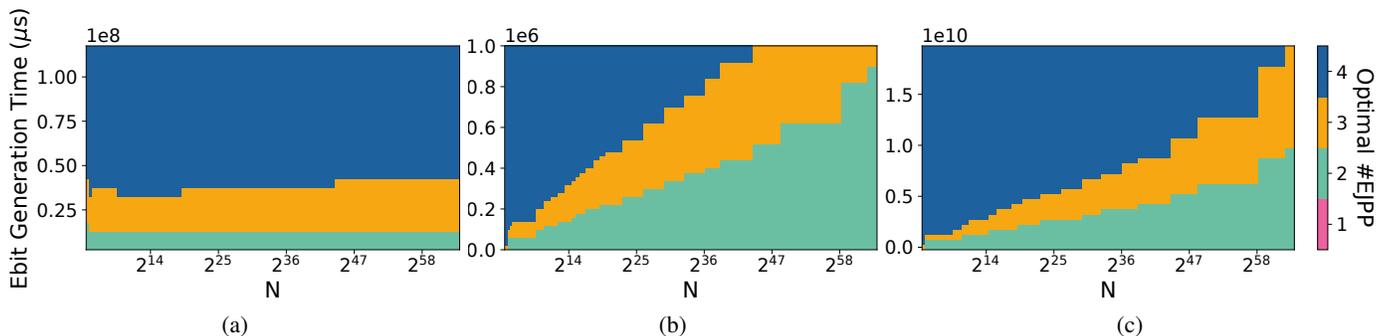
Fig. 20: Distributed delay scaling behavior for different ebit generation times and number N to factor. The heatmaps show the optimal choice for the number of ebit channels (denoted as EJPP) for the neutral atom (a), IBM superconducting (b) and IonQ ion-trap (c) weights.
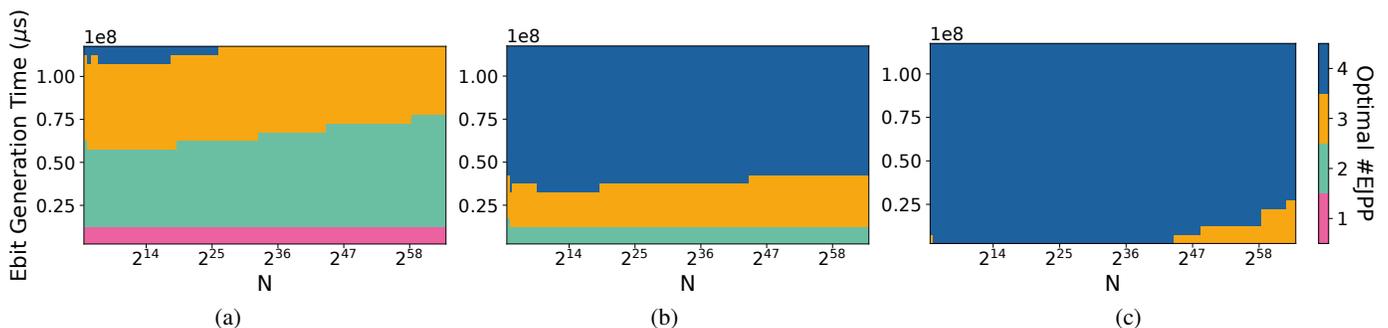

Fig. 21: Distributed delay scaling behavior for different ebit generation times and number N to factor. The heatmaps show the optimal choice for the number of ebit channels (denoted as EJPP) for the neutral atom setup and the iterative(a), alternating (b) and regular design (c). The difference in phase processing idle time from the monolithic designs lead to different behaviors when distributed.

larger values of N, fewer ebit channels are needed because the required ebit generation rate decreases; slower ebit generation becomes acceptable since CU operations take longer to complete.

In the ion trap and superconducting setups, distributing the three monolithic design choices shows no significant differences, which aligns with the monolithic analysis where all three approaches exhibited similar delays. However, the neutral atom setup behaves differently, as illustrated in Figure 21. In the iterative approach, fewer ebit channels are required compared to the alternating and regular cases. With very fast ebits, even a single ebit channel is sufficient. Conversely, the regular approach typically requires four ebit channels, with three being sufficient for larger N and fast ebits. The alternating case falls in between, requiring two ebit channels for fast ebits and gradually more for slower ebits. This behavior is linked to the parallel execution of ebit generation and phase processing during CU operations. In the iterative setup, phase processing takes a longer time, allowing ebit generation to be slower without causing idle time. In contrast, the regular setup minimizes idle time during phase processing, necessitating faster ebit generation to avoid distribution-related idle time.

To analyze the transition points for optimal ebit channel configurations, we first examined the delay implications of multiple ebit channels by evaluating their behavior as N

scales, with fixed ebit times for a specific hardware setup. Figure 22a illustrates the delay for one to four ebit channels in the alternating design with $t_{ebit} = 0.55$ms for the superconducting hardware configuration. Notably, the setup with a single ebit channel shows a significantly higher delay than the others. This is expected, as a single ebit channel offers no parallelization, resulting in prolonged idle times. Figure 22b presents the reduction in delay relative to the single ebit channel setup. For small N, ebit generation constitutes the majority of the delay. With k ebit channels, k ebit generations can occur simultaneously, reducing the delay to approximately $1/k$ of that with no parallelization. However, as N increases, the performance gains from additional ebit channels diminish.

We also explored a scenario where the hardware is not fixed, and instead, the number to be factored is predefined, requiring the selection of the optimal distribution hardware (ebit channel count and speed). Figure 23 shows the delay scaling and relative delay reduction (compared to the delay with one ebit channel) for a fixed 45 bit number N. For very small ebit generation times, multiple ebit channels provide minimal benefit, as the impact of mitigable delay on the overall delay is negligible. However, as ebit generation times increase, parallelization becomes advantageous, with only two ebit channels being sufficient for effective parallelism. As seen in Figure 23b, when the ebit generation time approaches the
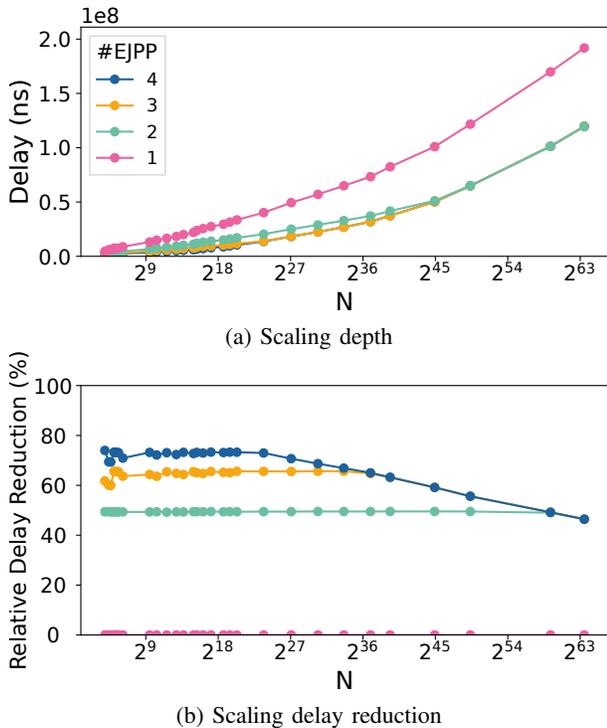
(a) Scaling depth

(b) Scaling delay reduction

Fig. 22: Scaling of the delay (a) and delay reduction (b) for different N and fixed ebit generation time of 0.55ms for the ion trap setup.
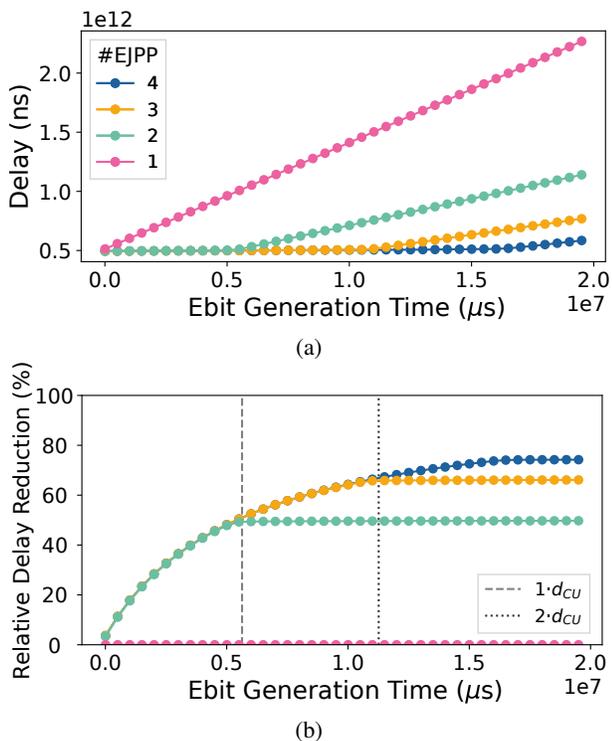

(a)

(b)

Fig. 23: Scaling of the delay (a) and delay reduction (b) for different ebit times and a fixed N with $n = 45$ bits for the ion trap setup.

average $t(CU)$ delay, three and four ebit channels start to outperform two. This is in line with the condition outlined in Equation 16, where ebit generation exceeds the duration of a single CU operation, making alternating between two channels inadequate. Similarly, when the ebit generation time approaches $2t(CU)$, three ebit channels no longer suffice, and four ebit channels are necessary to mitigate as much idle time as possible.
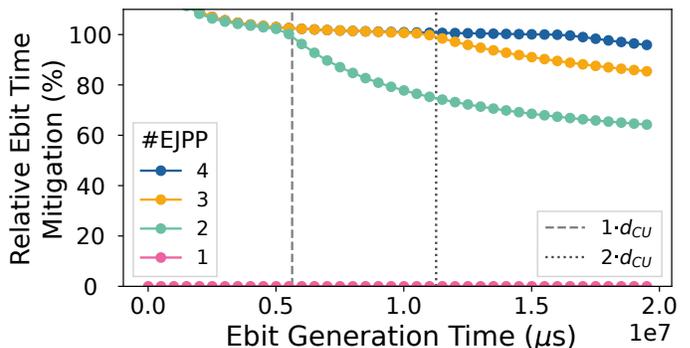

Fig. 24: Relative mitigation of idle time compared to the bound of idle time from distribution (Equation 17) for different ebit times and a fixed N with $n = 45$ bits for the ion trap setup. For small ebit times, circuit transpilation in overlaps between CU operations lead to mitigation above 100%.

Figure 24 demonstrates the extent to which parallelization mitigates the theoretical time bound (Equation 21) for distributed computing, including ebit generation and the EJPP start- and end-processes. For small ebit generation times, setups with multiple ebit channels can eliminate all idle time except for the unavoidable initial ebit generation and start process. As the ebit generation time increases, we again observe the thresholds of $t(CU)$ and $2t(CU)$, where two and three ebit channels no longer provide optimal efficiency. For large ebit times, the time required for distribution dominates the delay and the results converge with those shown in Figure 23b. In this regime, with k ebit channels, ebit generations are split into sets of k, which can be executed in parallel, reducing the circuit delay to $1/k$. This also reveals diminishing returns when moving from k to $k + 1$ ebit channels.

These findings highlight how the performance of Shor's algorithm implementations is influenced by available hardware and problem size, providing valuable insights for making optimal design decisions. A mid-level perspective of multiple designs allows for more flexibility on leveraging a targeted hardware efficiently. As our study shows, the difference in execution time between different designs and ebit channel parallelization strategies strongly depends on the hardware platform, problem size, and distribution setup. An analysis like ours can help determine how much of a targeted hardware's available compute resources can be effectively utilized for performance improvements. For instance, there may be a trade-off between having multiple slower ebit channels or fewer faster channels, depending on the scale of Shor's algorithm being run. Other possible use cases are scenarios of multiple programs running on a single QPU or a QPU cluster. In such cases, a workload manager [67] could evaluate whether the

benefit of more ebit channels for one program justifies limiting the resources available for others. Integrating our proposed analysis tools and designs to such tools would improve their allocation and scheduling capabilities.

VI. CONCLUSION

In this work, we investigated mid-level designs for Shor's algorithm from a timing perspective and introduced a new design for reducing idle time while preserving qubit efficiency. By reordering tasks to enable simultaneous execution, we improved overall execution time and extended these optimizations to distributed quantum systems. In distributed setups, we identified idle time bottlenecks caused by distribution protocols and proposed mitigation strategies using multiple ebit channels.

To systematically account for hardware-specific execution times, we adapted static timing analysis (STA) techniques from classical circuit design for quantum circuit optimization. This allowed us to incorporate gate execution delays and communication overhead in our design process. We evaluated our approaches on neutral atom, superconducting, and ion-trap platforms—both monolithic and distributed—by integrating modular arithmetic circuits from the QRISP framework. Our results confirmed that the proposed alternating design effectively reduces idle time and demonstrated how hardware constraints can inform circuit optimization in both monolithic and distributed settings.

Our approach complements existing arithmetic circuit optimization techniques by enhancing execution efficiency through task scheduling and idle time reduction. While prior research on Shor's algorithm has primarily focused on minimizing circuit depth and qubit count, we show that task reordering at a mid-level abstraction provides additional performance gains, especially when considering execution times of actual hardware.

Similarly, our mid-level approach to distributed execution does not replace necessary low-level task distribution but instead adds another layer of refinement for ordering tasks and managing information exchange. As large-scale quantum networks emerge, efficient scheduling and synchronization of distributed tasks will become increasingly critical.

Beyond Shor's algorithm, our findings suggest that similar parallelization and idle time reduction techniques could benefit other quantum algorithms with layered computational structures. QPE-based algorithms are a natural extension due to their structural similarities, but broader applications in quantum computing may also benefit from improved task scheduling and distribution-aware execution strategies.

Our study underscores the importance of hardware-aware quantum circuit design and serves as a foundation for integrating STA-inspired methods into quantum circuit compilation and design automation. A promising direction for future research is exploring additional timing-based optimization techniques for other quantum algorithms, particularly in error-corrected quantum systems, where logical operations introduce further time constraints.

An important next step is to validate our designs on real hardware once distributed quantum computing systems be-come publicly available. While our study focuses on relatively large circuits, factoring 2048-bit RSA keys remains well beyond current capabilities. Reaching this scale will likely necessitate error-corrected quantum systems [5], reinforcing the relevance of our techniques for logical quantum computing as a key area for future research.

Finally, as distributed quantum computing advances, new challenges will arise in synchronization, task coordination, and ebit generation constraints. Our methods provide an initial step toward addressing these challenges by reducing idle time, and future research can further refine task scheduling and synchronization strategies to enhance distributed quantum execution, contributing to the broader goal of scalable, distributed quantum computation.

REFERENCES

[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Annu. Symp. Found. Comput. Sci.*, 1994, pp. 124–134.

[2] V. Bhatia and K. Ramkumar, "An efficient quantum computing technique for cracking rsa using shor's algorithm," in *2020 IEEE 5th international conference on computing communication and automation (ICCCA)*. IEEE, 2020, pp. 89–94.

[3] J. A. Buchmann, D. Butin, F. Göpfert, and A. Petzoldt, "Post-quantum cryptography: state of the art," *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, pp. 88–108, 2016.

[4] D. J. Bernstein and T. Lange, "Post-quantum cryptography," *Nature*, vol. 549, no. 7671, pp. 188–194, 2017.

[5] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[6] D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, "Efficient networks for quantum factoring," *Physical Review A*, vol. 54, no. 2, p. 1034, 1996.

[7] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, no. 1, p. 147, 1996.

[8] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," *arXiv preprint quant-ph/0410184*, 2004.

[9] T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder," *arXiv preprint quant-ph/0406142*, 2004.

[10] A. Pavlidis and D. Gizopoulos, "Fast quantum modular exponentiation architecture for shor's factorization algorithm," *arXiv preprint arXiv:1207.0511*, 2012.

[11] C. Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, p. 74, 2018.

[12] T. G. Draper, "Addition on a quantum computer," *arXiv preprint quant-ph/0008033*, 2000.

[13] S. Beauregard, "Circuit for shor's algorithm using 2n+3 qubits," 2003.

[14] Y. Takahashi and N. Kunihiro, "A quantum circuit for shor's factoring algorithm using 2n + 2 qubits," *Quantum Info. Comput.*, vol. 6, no. 2, p. 184–192, mar 2006.

[15] T. Häner, M. Roetteler, and K. M. Svore, "Factoring using 2n+2 qubits with toffoli based modular multiplication," *arXiv preprint arXiv:1611.07995*, 2016.

[16] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: a survey," *Computer Networks*, vol. 254, p. 110672, 2024.

[17] A. S. Cacciapuoti, M. Caleffi, R. Van Meter, and L. Hanzo, "When entanglement meets classical communications: Quantum teleportation for the quantum internet," *IEEE Transactions on Communications*, vol. 68, no. 6, pp. 3808–3833, 2020.

[18] K. Azuma, S. E. Economou, D. Elkouss, P. Hilaire, L. Jiang, H.-K. Lo, and I. Tzitrin, "Quantum repeaters: From quantum networks to the quantum internet," *Reviews of Modern Physics*, vol. 95, no. 4, p. 045006, 2023.

[19] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler design for distributed quantum computing," vol. 2, pp. 1–20, 2021.

[20] D. Ferrari, S. Carretta, and M. Amoretti, "A modular quantum compilation framework for distributed quantum computing," vol. 4, pp. 1–13, 2023.

[21] S. Harris and D. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2015.

[22] L. Lavagno, I. L. Markov, G. Martin, and L. K. Scheffer, *Electronic design automation for IC implementation, circuit design, and process technology*. CRC Press, 2017.

[23] R. Seidel, S. Bock, R. Zander, M. Petrič, N. Steinmann, N. Tcholtchev, and M. Hauswirth, "Qrisp: A framework for compilable high-level programming of gate-based quantum computers," *arXiv preprint arXiv:2406.14792*, 2024.

[24] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[25] U. Skosana and M. Tame, "Demonstration of Shor's factoring algorithm for N = 21 on IBM quantum processors," *Sci Rep*, vol. 11, no. 1, p. 16599, Aug 2021.

[26] M. Amico, Z. H. Saleem, and M. Kumph, "Experimental study of shor's factoring algorithm using the ibm q experience," *Phys. Rev. A*, vol. 100, p. 012305, Jul 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.100.012305

[27] T. Monz, D. Nigg, E. A. Martinez, M. F. Brandl, P. Schindler, R. Rines, S. X. Wang, I. L. Chuang, and R. Blatt, "Realization of a scalable shor algorithm," *Science*, vol. 351, no. 6277, pp. 1068–1070, 2016.

[28] F. Hua, Y. Jin, Y. Chen, S. Vittal, K. Krsulich, L. S. Bishop, J. Lapeyre, A. Javadi-Abhari, and E. Z. Zhang, "Exploiting qubit reuse through mid-circuit measurement and reset," *arXiv preprint arXiv:2211.01925*, 2022.

[29] S. Brandhofer, I. Polian, and K. Krsulich, "Optimal qubit reuse for near-term quantum computers," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2023, pp. 859–869.

[30] A. D. Córcoles, M. Takita, K. Inoue, S. Lekuch, Z. K. Minev, J. M. Chow, and J. M. Gambetta, "Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits," *Physical Review Letters*, vol. 127, no. 10, p. 100501, 2021.

[31] C. Ryan-Anderson, N. Brown, M. Allman, B. Arkin, G. Asa-Attuah, C. Baldwin, J. Berg, J. Bohnet, S. Braxton, N. Burdick *et al.*, "Implementing fault-tolerant entangling gates on the five-qubit code and the color code," *arXiv preprint arXiv:2208.01863*, 2022.

[32] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. Allman, C. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer *et al.*, "Demonstration of the trapped-ion quantum ccd computer architecture," *Nature*, vol. 592, no. 7853, pp. 209–213, 2021.

[33] E. Bäumer, V. Tripathi, D. S. Wang, P. Rall, E. H. Chen, S. Majumder, A. Seif, and Z. K. Minev, "Efficient long-range entanglement using dynamic circuits," *PRX Quantum*, vol. 5, no. 3, p. 030339, 2024.

[34] E. Bäumer and S. Woerner, "Measurement-based long-range entangling gates in constant depth," *arXiv preprint arXiv:2408.03064*, 2024.

[35] K. C. Smith, E. Crane, N. Wiebe, and S. Girvin, "Deterministic constant-depth preparation of the aklt state on a quantum processor using fusion measurements," *PRX Quantum*, vol. 4, no. 2, p. 020315, 2023.

[36] K. C. Smith, A. Khan, B. K. Clark, S. Girvin, and T.-C. Wei, "Constant-depth preparation of matrix product states with adaptive quantum circuits," *arXiv preprint arXiv:2404.16083*, 2024.

[37] R. B. Griffiths and C.-S. Niu, "Semiclassical fourier transform for quantum computation," *Phys. Rev. Lett.*, vol. 76, pp. 3228–3231, Apr 1996.

[38] E. Bäumer, V. Tripathi, A. Seif, D. Lidar, and D. S. Wang, "Quantum fourier transform using dynamic circuits," *Physical Review Letters*, vol. 133, no. 15, p. 150602, 2024.

[39] E. Knill, G. Ortiz, and R. D. Somma, "Optimal quantum measurements of expectation values of observables," *Phys. Rev. A*, vol. 75, p. 012328, Jan 2007. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.75.012328

[40] M. Dobšíček, G. Johansson, V. Shumeiko, and G. Wendin, "Arbitrary accuracy iterative quantum phase estimation algorithm using a single ancillary qubit: A two-qubit benchmark," *Phys. Rev. A*, vol. 76, p. 030306, Sep 2007. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.76.030306

[41] S. Parker and M. B. Plenio, "Efficient factorization with a single pure qubit and log n mixed qubits," *Physical Review Letters*, vol. 85, no. 14, p. 3049, 2000.

[42] M. Mosca and A. Ekert, "The hidden subgroup problem and eigenvalue estimation on a quantum computer," in *NASA International Conference on Quantum Computing and Quantum Communications*. Springer, 1998, pp. 174–188.

[43] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'brien, "Experimental realization of shor's quantum factoring algorithm using qubit recycling," *Nature photonics*, vol. 6, no. 11, pp. 773–776, 2012.

[44] J. W. Lis, A. Senoo, W. F. McGrew, F. Rönchen, A. Jenkins, and A. M. Kaufman, "Midcircuit operations using the omg architecture in neutral atom arrays," *Physical Review X*, vol. 13, no. 4, p. 041035, 2023.

[45] T. Graham, L. Phuttitarn, R. Chinnarasu, Y. Song, C. Poole, K. Jooya, J. Scott, A. Scott, P. Eichler, and M. Saffman, "Midcircuit measurements on a single-species neutral alkali atom quantum processor," *Physical Review X*, vol. 13, no. 4, p. 041051, 2023.

[46] S. Niu, E. Kokcu, A. Mitra, A. Szasz, A. Hashim, J. Kalloor, W. A. de Jong, C. Iancu, and E. Younis, "Ac/dc: Automated compilation for dynamic circuits," 2024. [Online]. Available: https://arxiv.org/abs/2412.07969

[47] M. DeCross, E. Chertkov, M. Kohagen, and M. Foss-Feig, "Qubit-reuse compilation with mid-circuit measurement and reset," *Physical Review X*, vol. 13, no. 4, p. 041057, 2023.

[48] M. Mohseni, A. Scherer, K. G. Johnson, O. Wertheim, M. Otten, N. A. Aadit, Y. Alexeev, K. M. Bresniker, K. Y. Camsari, B. Chapman, S. Chatterjee, G. A. Dagnew, A. Esposito, F. Fahim, M. Fiorentino,

A. Gajjar, A. Khalid, X. Kong, B. Kulchytskyy, E. Kyoseva, R. Li, P. A. Lott, I. L. Markov, R. F. McDermott, G. Pedretti, P. Rao, E. Rieffel, A. Silva, J. Sorebo, P. Spentzouris, Z. Steiner, B. Torosov, D. Venturelli, R. J. Visser, Z. Webb, X. Zhan, Y. Cohen, P. Ronagh, A. Ho, R. G. Beausoleil, and J. M. Martinis, "How to build a quantum supercomputer: Scaling from hundreds to millions of qubits," 2025. [Online]. Available: https://arxiv.org/abs/2411.10406

[49] K. Wintersperger, F. Dommert, T. Ehmer, A. Hoursanov, J. Klepsch, W. Mauerer, G. Reuber, T. Strohm, M. Yin, and S. Luber, "Neutral atom quantum computing hardware: performance and end-user perspective," *EPJ Quantum Technology*, vol. 10, no. 1, p. 32, 2023.

[50] J. M. Amini, H. Uys, J. H. Wesenberg, S. Seidelin, J. Britton, J. J. Bollinger, D. Leibfried, C. Ospelkaus, A. P. VanDevender, and D. J. Wineland, "Toward scalable ion traps for quantum information processing," *New Journal of Physics*, vol. 12, no. 3, p. 033031, mar 2010. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/12/3/033031

[51] H. J. Kimble, "The quantum internet," *Nature*, vol. 453, no. 7198, pp. 1023–1030, 2008.

[52] W. J. Munro, K. Azuma, K. Tamaki, and K. Nemoto, "Inside quantum repeaters," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 21, no. 3, pp. 78–90, 2015.

[53] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels," *Phys. Rev. Lett.*, vol. 70, pp. 1895–1899, Mar 1993.

[54] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio, "Optimal local implementation of nonlocal quantum gates," *Physical Review A*, vol. 62, no. 5, p. 052317, 2000.

[55] J.-Y. Wu, K. Matsui, T. Forrer, A. Soeda, P. Andrés-Martínez, D. Mills, L. Henaut, and M. Murao, "Entanglement-efficient bipartite-distributed quantum computing," *Quantum*, vol. 7, p. 1196, Dec. 2023.

[56] A. C.-C. Yao, "Quantum circuit complexity," in *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*. IEEE, 1993, pp. 352–361.

[57] D. Bhoumik, R. Majumdar, A. Saha, and S. Sur-Kolay, "Distributed scheduling of quantum circuits with noise and time optimization," *arXiv preprint arXiv:2309.06005*, 2023.

[58] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, no. 2, p. 025004, 2018.

[59] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 360–374.

[60] H. Deng, Y. Zhang, and Q. Li, "Codar: A contextual duration-aware qubit mapping for various nisq devices," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[61] I. Shaik and J. van de Pol, "Optimal layout synthesis for quantum circuits as classical planning," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[62] M. Caleffi, "Optimal routing for quantum networks," *Ieee Access*, vol. 5, pp. 22 299–22 312, 2017.

[63] M. Ghaderibaneh, C. Zhan, H. Gupta, and C. Ramakrishnan, "Efficient quantum network communication using optimized entanglement swapping trees," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–20, 2022.

[64] S. Shi and C. Qian, "Concurrent entanglement routing for quantum networks: Model and designs," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 62–75.

[65] T. Coopmans, S. Brand, and D. Elkouss, "Improved analytical bounds on delivery times of long-distance entanglement," *Physical Review A*, vol. 105, no. 1, p. 012608, 2022.

[66] S. Rodrigo, D. Spanò, M. Bandic, S. Abadal, H. Van Someren, A. Ovide, S. Feld, C. G. Almudéver, and E. Alarcón, "Characterizing the spatio-temporal qubit traffic of a quantum intranet aiming at modular quantum computer architectures," in *Proceedings of the 9th ACM International Conference on Nanoscale Computing and Communication*, 2022, pp. 1–7.

[67] D. Ferrari, M. Bandini, and M. Amoretti, "Execution management of distributed quantum computing jobs," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 2. IEEE, 2024, pp. 150–154.

[68] A. Yimsiriwattana and S. J. L. Jr., "Distributed quantum computing: a distributed Shor algorithm," in *Quantum Information and Computation II*, vol. 5436, 2004, pp. 360 – 372.

[69] R. V. Meter, W. Munro, K. Nemoto, and K. M. Itoh, "Arithmetic on a distributed-memory quantum multicomputer," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 3, no. 4, pp. 1–23, 2008.

[70] N. Neumann, R. van Houte, and T. Attema, "Imperfect distributed quantum phase estimation," in *Computational Science – ICCS 2020*, V. Krzhizhanovskaya *et al.*, Eds. Cham: Springer, 2020, pp. 605–615.

[71] P. Andres-Martinez, T. Forrer, D. Mills, J.-Y. Wu, L. Henaut, K. Yamamoto, M. Murao, and R. Duncan, "Distributing circuits over heterogeneous, modular quantum computing network architectures," *Quantum Science and Technology*, vol. 9, no. 4, p. 045021, 2024.

[72] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.

[73] I. Quantum, "Ibm quantum experience," 2025. [Online]. Available: https://quantum-computing.ibm.com

[74] "Ionq quantum cloud," https://cloud.ionq.com/backends, accessed: 2025-01-10.

[75] Q. C. Inc., "Quera neutral atom quantum computers," 2025. [Online]. Available: https://www.quera.com

[76] A. Radnaev, W. Chung, D. Cole, D. Mason, T. Ballance, M. Bedalov, D. Belknap, M. Berman, M. Blakely, I. Bloomfield *et al.*, "A universal neutral-atom quantum computer with individual optical addressing and non-destructive readout," *arXiv preprint arXiv:2408.08288*, 2024.

[77] D. Bluvstein, H. Levine, G. Semeghini, T. T. Wang, S. Ebadi, M. Kalinowski, A. Keesling, N. Maskara, H. Pichler, M. Greiner *et al.*, "A quantum processor based on coherent transport of entangled atom arrays," *Nature*, vol. 604, no. 7906, pp. 451–456, 2022.

[78] H. Levine, A. Keesling, G. Semeghini, A. Omran, T. T. Wang, S. Ebadi, H. Bernien, M. Greiner, V. Vuletić, H. Pichler *et al.*, "Parallel implementation of high-fidelity multiqubit gates with neutral atoms," *Physical review letters*, vol. 123, no. 17, p. 170503, 2019.

[79] W. Xu, A. V. Venkatramani, S. H. Cantú, T. Šumarac, V. Klüsener, M. D. Lukin, and V. Vuletić, "Fast preparation and detection of a rydberg qubit using atomic ensembles," *Physical Review Letters*, vol. 127, no. 5, p. 050501, 2021.

[80] S. J. Evered, D. Bluvstein, M. Kalinowski, S. Ebadi, T. Manovitz, H. Zhou, S. H. Li, A. A. Geim, T. T. Wang, N. Maskara *et al.*, "High-fidelity parallel entangling gates on a neutral-atom quantum computer," *Nature*, vol. 622, no. 7982, pp. 268–272, 2023.

[81] S. Jandura and G. Pupillo, "Time-optimal two-and three-qubit gates for rydberg atoms," *Quantum*, vol. 6, p. 712, 2022.

[82] Y. Chew, T. Tomita, T. P. Mahesh, S. Sugawa, S. de Léséleuc, and K. Ohmori, "Ultrafast energy exchange between two single rydberg atoms on a nanosecond timescale," *Nature Photonics*, vol. 16, no. 10, pp. 724–729, 2022.

[83] M. A. Norcia, W. B. Cairncross, K. Barnes, P. Battaglino, A. Brown, M. O. Brown, K. Cassella, C.-A. Chen, R. Coxe, D. Crow, J. Epstein, C. Griger, A. M. W. Jones, H. Kim, J. M. Kindem, J. King, S. S. Kondov, K. Kotru, J. Lauigan, M. Li, M. Lu, E. Megidish, J. Marjanovic, M. McDonald, T. Mittiga, J. A. Muniz, S. Narayanaswami, C. Nishiguchi, R. Notermans, T. Paule, K. A. Pawlak, L. S. Peng, A. Ryou, A. Smull, D. Stack, M. Stone, A. Sucich, M. Urbanek, R. J. M. van de Veerdonk, Z. Vendeiro, T. Wilkason, T.-Y. Wu, X. Xie, X. Zhang, and B. J. Bloom, "Midcircuit qubit measurement and rearrangement in a ^{171}Yb atomic array," *Phys. Rev. X*, vol. 13, p. 041034, Nov 2023. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.13.041034

[84] M. Uphoff, M. Brekenfeld, G. Rempe, and S. Ritter, "An integrated quantum repeater at telecom wavelength with single atoms in optical fiber cavities," *Applied Physics B*, vol. 122, pp. 1–15, 2016.

[85] J. Hofmann, M. Krug, N. Ortegel, L. Gérard, M. Weber, W. Rosenfeld, and H. Weinfurter, "Heralded entanglement between widely separated atoms," *Science*, vol. 337, no. 6090, pp. 72–75, 2012.

[86] S. Ghosh, N. Rivera, G. Eisenstein, and I. Kaminer, "Creating heralded hyper-entangled photons using rydberg atoms," *Light: Science & Applications*, vol. 10, no. 1, p. 100, 2021.

[87] T. van Leent, M. Bock, R. Garthoff, K. Redeker, W. Zhang, T. Bauer, W. Rosenfeld, C. Becher, and H. Weinfurter, "Long-distance distribution of atom-photon entanglement at telecom wavelength," *Physical Review Letters*, vol. 124, no. 1, p. 010510, 2020.

[88] Y. Zhou, P. Malik, F. Fertig, M. Bock, T. Bauer, T. van Leent, W. Zhang, C. Becher, and H. Weinfurter, "Long-lived quantum memory enabling atom-photon entanglement over 101 km of telecom fiber," *PRX Quantum*, vol. 5, no. 2, p. 020307, 2024.

[89] V. Krutyanskiy, M. Galli, V. Krcmarsky, S. Baier, D. Fioretto, Y. Pu, A. Mazloom, P. Sekatski, M. Canteri, M. Teller *et al.*, "Entanglement of

trapped-ion qubits separated by 230 meters," *Physical Review Letters*, vol. 130, no. 5, p. 050803, 2023.

[90] L. Stephenson, D. Nadlinger, B. Nichol, S. An, P. Drmota, T. Ballance, K. Thirumalai, J. Goodwin, D. Lucas, and C. Ballance, "High-rate, high-fidelity entanglement of qubits across an elementary quantum network," *Physical review letters*, vol. 124, no. 11, p. 110501, 2020.

[91] E. Bersin, M. Sutula, Y. Q. Huan, A. Suleymanzade, D. R. Assumpcao, Y.-C. Wei, P.-J. Stas, C. M. Knaut, E. N. Knall, C. Langrock *et al.*, "Telecom networking with a diamond quantum memory," *PRX Quantum*, vol. 5, no. 1, p. 010303, 2024.

[92] J. Ang, G. Carini, Y. Chen, I. Chuang, M. Demarco, S. Economou, A. Eickbusch, A. Faraon, K.-M. Fu, S. Girvin *et al.*, "Arquin: architectures for multinode superconducting quantum computers," *ACM Transactions on Quantum Computing*, vol. 5, no. 3, pp. 1–59, 2024.

[93] S. Krastanov, H. Raniwala, J. Holzgrafe, K. Jacobs, M. Lončar, M. J. Reagor, and D. R. Englund, "Optically heralded entanglement of super-conducting systems in quantum networks," *Physical Review Letters*, vol. 127, no. 4, p. 040503, 2021.

[94] P. Kurpiers, P. Magnard, T. Walter, B. Royer, M. Pechal, J. Heinsoo, Y. Salathé, A. Akin, S. Storz, J.-C. Besse *et al.*, "Deterministic quantum state transfer and remote entanglement using microwave photons," *Nature*, vol. 558, no. 7709, pp. 264–267, 2018.

[95] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, 2023, MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.

[96] D. Main, P. Drmota, D. Nadlinger, E. Ainley, A. Agrawal, B. Nichol, R. Srinivas, G. Araneda, and D. Lucas, "Distributed quantum computing across an optical network link," *Nature*, pp. 1–6, 2025.

[97] S. Wang, A. Baksi, and A. Chattopadhyay, "A higher radix architecture for quantum carry-lookahead adder," *Scientific Reports*, vol. 13, no. 1, p. 16338, 2023.

[98] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[99] U. Skosana and M. Tame, "Demonstration of shor's factoring algorithm for n= 21 on ibm quantum processors," *Scientific reports*, vol. 11, no. 1, p. 16599, 2021.