Designing a Million-Qubit Quantum Computer Using a Resource Performance Simulator

MUHAMMAD AHSAN, Duke University
RODNEY VAN METER, Keio University, Japan
JUNGSANG KIM, Duke University

The optimal design of a fault-tolerant quantum computer involves finding an appropriate balance between the burden of large-scale integration of noisy components and the load of improving the reliability of hardware technology. This balance can be evaluated by quantitatively modeling the execution of quantum logic operations on a realistic quantum hardware containing limited computational resources. In this work, we report a complete performance simulation software tool capable of (1) searching the hardware design space by varying resource architecture and technology parameters, (2) synthesizing and scheduling a fault-tolerant quantum algorithm within the hardware constraints, (3) quantifying the performance metrics such as the execution time and the failure probability of the algorithm, and (4) analyzing the breakdown of these metrics to highlight the performance bottlenecks and visualizing resource utilization to evaluate the adequacy of the chosen design. Using this tool, we investigate a vast design space for implementing key building blocks of Shor's algorithm to factor a 1,024-bit number with a baseline budget of 1.5 million qubits. We show that a trapped-ion quantum computer designed with twice as many qubits and one-tenth of the baseline infidelity of the communication channel can factor a 2,048-bit integer in less than 5 months.

CCS Concepts: • **Computer systems organization → Quantum computing**; • **Hardware → Quantum error correction and fault tolerance**

Additional Key Words and Phrases: Quantum architecture, architecture scalability, resource performance tradeoffs, performance simulation tool, hardware constraints

ACM Reference Format:
Muhammad Ahsan, Rodney Van Meter, and Jungsang Kim. 2015. Designing a million-qubit quantum computer using a resource performance simulator. J. Emerg. Technol. Comput. Syst. 12, 4, Article 39 (December 2015), 25 pages.
DOI: http://dx.doi.org/10.1145/2830570

1. INTRODUCTION

Although quantum computers (QCs) can in principle solve important problems such as factoring a product of large prime numbers efficiently, the prospect of constructing a practical system is hampered by the need to build reliable systems out of faulty components [Van Meter and Horsman 2013]. Fault-tolerant procedures utilizing quantum error correcting codes (QECCs) achieve adequate error performance by protecting the quantum information from noise, but this comes at the expense of substantial resource

This work was funded by Intelligence Advanced Research Projects Activity (IARPA) under the Multi-Qubit Coherent Operation (MQCO) Program and the Quantum Computer Science (QCS) program.
Authors' addresses: M. Ahsan, Department of Computer Science, Duke University, Durham, North Carolina, United States of America 27708; email: ahsan@cs.duke.edu; R. Van Meter, Faculty of Environment and Information Studies, 1-8-13 Zaimokuza Kamakura, Kanagawa-ken, Japan 248-0013; email: rdv@sfc.wide.ad.jp; J. Kim, Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, United States of America 27708; email: jungsang@duke.edu.

ACM 1550-4832/2015/12-ART39 $15.00
DOI: http://dx.doi.org/10.1145/2830570

investment [Nielsen and Chuang 2000]. The threshold theorem (or the quantum fault tolerance theorem) says that a quantum computation of arbitrary size can be performed as long as the error probability of each operation is kept below a certain threshold value and sufficient computational resources, such as the number of quantum bits (qubits), can be provided to implement adequate fault tolerance [Aharonov and Ben-Or 1997]. Although this is an encouraging theoretical result, an accurate estimate of the resource overhead remains an extremely complex task, as it depends on the details of the hardware (qubit connectivity, gate speeds and coherence time, etc.), the choice of protocols (QECC, etc.), and the nature of the target algorithms. Several application-optimized architectures have been proposed and analyzed [Metodi et al. 2005; Van Meter et al. 2008; Whitney et al. 2009; Kim and Kim 2009; Monroe et al. 2014; Galiautdinov et al. 2012; Fowler et al. 2012], yet the accurate quantification of resource performance scaling for various benchmarks remains a challenging problem.

In this work, we quantitatively define the *scalability* of a quantum architecture to mean that the resource overhead of running a quantum algorithm, while sustaining expected behavior in execution time and success probability (of order unity, $\sim O(1)$), increases linearly with the problem size. We propose a modular ion-trap-based architecture and quantify its scalability for three different benchmark circuits crucial for Shor's factoring algorithm [Shor 1997]: a *quantum carry look-ahead adder* (QCLA) [Draper et al. 2006], the CDKM *quantum ripple-carry adder* (QRCA) [Cuccaro et al. 2004], and an *approximate quantum Fourier transform* (AQFT) [Fowler and Hollenberg 2004]. This architecture features fast and reliable interconnects to ensure efficient access to computational resources and enables flexible distribution of computational resources to various workload-intensive parts of the system depending on the circuit being executed. By evaluating this architecture for a variety of benchmarks, we show that it can achieve highly optimized performance by flexible and efficient utilization of given resources over a range of interesting quantum circuits.

To quantify the performance of an architecture as a function of available resources, we develop a performance simulation tool similar to those reported in Svore et al. [2006], Whitney et al. [2007], Balensiefer et al. [2005], and Whitney et al. [2009] that (1) maps application circuits on to the quantum hardware, (2) generates and schedules the sequence of quantum logic gates from the algorithm operating on the qubits mapped to the hardware, and (3) estimates performance metrics such as total execution time and failure probability. Unlike the tools reported previously, our tool features unique capabilities to (1) simulate performance over varying hardware device parameters, (2) allow dynamic resource allocation in the architecture, (3) provide detailed breakdown of resource and performance variables, and (4) enable visualization of resource utilization over (5) a range of benchmark applications. By leveraging these unique attributes, we search the architecture space for a suitable QC design while providing valuable insights into the factors limiting performance in a large-scale QC.

This article is organized as follows: Section 2 describes benchmark quantum circuits and their characteristics. Section 3 describes the underlying quantum hardware technology and the modular, reconfigurable architecture used in our simulation. The toolset and its main features are outlined in Section 4. Simulation results along with detailed discussions are given in Section 5, and Section 6 describes extensibility of the tool. Section 7 puts our work in the context of other previous quantum architecture studies, and Section 8 summarizes the main insights gained from our study.

2. QUANTUM CIRCUITS

2.1. Universal Quantum Gates

Quantum circuits consist of a sequence of gates on qubit operands. An n-qubit quantum gate performs a deterministic unitary transformation on n operand qubits. In the terminology of computer architecture, a gate corresponds to an "instruction," and the specific sequences of gates translate into instruction-level dependencies. Similar to classical computers, it is known that an arbitrary quantum circuit can be constructed using a finite set of gates (called *universal quantum gates*), which is not unique [Nielsen and Chuang 2000]. For fault-tolerant quantum computation, one has to encode the qubits in a QECC and perform logic gates on the encoded block of *logical qubits* [Nielsen and Chuang 2000]. There are two ways of performing gates on a logical qubit: in the first procedure, the quantum gate on logical qubit(s) is translated into a bitwise operation on constituent qubits (referred to as a "transversal" gate). Since an error on one constituent qubit in the logical qubit cannot lead to an error in another constituent qubit in the same logical qubit, the error remains correctable using the QECC, and therefore a transversal gate is automatically fault tolerant. For a good choice of QECC, most of the gates in the universal quantum gate set are transversal, and therefore fault-tolerant implementation is straightforward. Unfortunately, for most of the QECCs explored to date (the class of QECCs called additive codes), it is impossible to find a transversal implementation of *all* the gates in the universal quantum gate set [Zeng et al. 2011]. A second, general procedure for constructing such gates involves fault-tolerantly preparing a very special quantum state, called the "magic state," and then utilizing quantum teleportation to transfer the operand qubit into the magic state to complete the gate operation [Zhou et al. 2000]. This operation is generally much more resource intensive and time consuming, so minimizing such operations is a crucial optimization process for the fault-tolerant circuit synthesis.

We employ the widely used Steane [[7,1,3]] code [Steane 1996]. It invests seven qubits to encode one more strongly error-protected qubit. For the universal gate set, we utilize $\{X, Z, H, \text{CNOT}, \text{Toffoli}\}$ for the adder circuits (both QCLA and QRCA) and $\{X, Z, H, \text{CNOT}, T\}$ for the AQFT circuit. For a single qubit state $\alpha|0\rangle + \beta|1\rangle$, X and Z correspond to the bit-flip and phase-flip operations that take the state to $\alpha|1\rangle + \beta|0\rangle$ and $\alpha|0\rangle - \beta|1\rangle$, respectively, and span a Pauli group of operators. H is the Hadamard operator, which converts the computational basis states $|0\rangle$ and $|1\rangle$ to the equal linear superposition of two states $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$, and vice versa. CNOT is a two-qubit gate where the state of the second qubit (called the target qubit) is flipped if and only if the state of the first qubit (called the control qubit) is $|1\rangle$. Along with the Pauli operators, H and CNOT span a Clifford group of operators (Clifford gates). In the Steane code, all operators in the Clifford group can be implemented transversally. However, in order to complete the set of universal quantum gates, a non-Clifford gate must be added. This could be either the T gate (sometimes called the $\pi/8$-gate), a single-qubit gate that shifts the phase of the $|1\rangle$ state by $\pi/4$, or the Toffoli gate (sometimes called the Controlled-Controlled-NOT), a three-qubit gate where the state of the third qubit is flipped if and only if the state of the first and second qubits are both $|1\rangle$. Using either gate is equivalent, in the sense that a Toffoli gate can be constructed using several T gates and Clifford gates [Nielsen and Chuang 2000]. Fault-tolerant implementation of a non-Clifford gate requires magic state preparation when the Steane code is used. Figure 1 shows the fault-tolerant implementation of the Toffoli gate, where the Toffoli magic state is prepared with the help of four ancilla qubits followed by the teleportation of the operand qubits into the magic state.

2.2. Benchmark Circuits

Shor's factoring algorithm consists of an arithmetic calculation called modular exponentiation [Van Meter and Itoh 2005; Vedral et al. 1996; Beckman et al. 1996], which can be constructed from adder circuits, followed by a quantum Fourier transform [Shor 1997]. Arithmetic circuits like adders (QCLA and QRCA) can easily be constructed from X, CNOT, and Toffoli gates, while AQFT can be constructed more conveniently

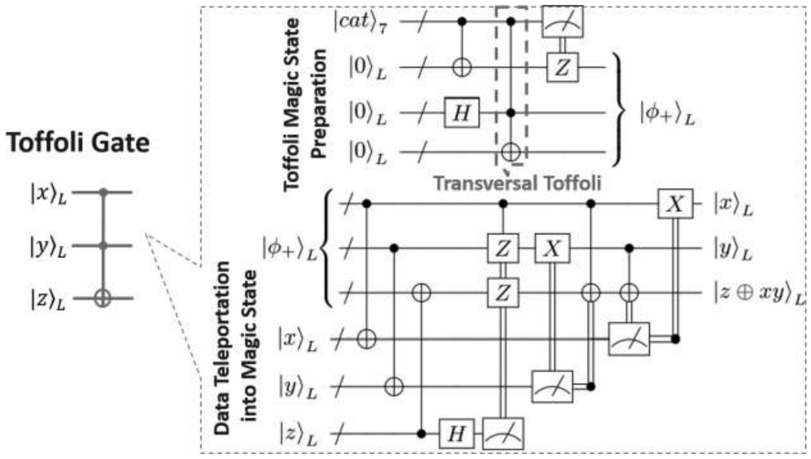


Fig. 1. Fault-tolerant circuit for Toffoli gate used in adder benchmarks. $|\alpha\rangle_L$ denotes a logical qubit block representing the state $|\alpha\rangle$.

from T gates. We consider a QC architecture where both Toffoli and T gates can be executed and optimize the architecture for executing all required quantum circuits for running Shor's algorithm.

2.2.1. Quantum Adders. A large number of quantum adder circuits must be called to complete the modular exponentiation that constitutes the bulk of Shor's algorithm. We select two candidate adders QRCA and QCLA, representing two vastly different addition strategies, analogous to classical adders. QRCA is a linear-depth circuit, containing serially connected CNOT and Toffoli gates: an n-bit addition will require about $2n$ qubits to perform $2n$ Toffoli and $5n$ CNOT gates [Cuccaro et al. 2004]. The sequence of these gates is inherently local, and nearest-neighbor connectivity among the qubits is sufficient to implement this circuit. On the other hand, QCLA is a logarithmic-depth [$\sim 4\log_2 n$] circuit connecting $4n$ qubits utilizing up to n concurrently executable gates [Draper et al. 2006]. This circuit roughly contains $5n - 3\log_2 n$ CNOT and Toffoli gates for n-bit addition. The exponential gain in performance (execution time) comes at the cost of sufficient availability of ancilla qubits and rapid communication channels among distant qubits to exploit parallelism. The QC hardware model considered here is unique in providing the global connectivity necessary for implementing QCLA. We study the resource-performance tradeoff in selecting QCLA versus QRCA in Section 5.

2.2.2. Approximate Quantum Fourier Transform. The quantum Fourier transform (QFT) circuit is often used as the keystone of the order-finding routine in Shor's algorithm [Nielsen and Chuang 2000]. It contains controlled-rotation gates $R_z(\pi/2^k)$, where the phase of the target qubit is shifted by $\pi/2^k$ for the $|1\rangle$ state if the control qubit is in the $|1\rangle$ state, for $1 \le k \le n$, in a n-qubit Fourier transform. Figure 2 shows that the controlled-rotation gates can first be decomposed into CNOTs and single-qubit rotations with twice the angle. These rotation operations are not in the Clifford group for $k > 1$ and must be approximated using gates from the universal quantum gate set [Nielsen and Chuang 2000]. A recent theoretical breakthrough provides an asymptotically optimal way of approximating an arbitrary quantum gate with a precision of ϵ using only $O(\log(1/\epsilon))$ Clifford group and T gates, and a concrete algorithm for generating the approximation circuit [Kliuchnikov et al. 2013; Giles and Selinger 2013].

It has been shown that a QFT circuit can yield the correct result with high enough probability even if one eliminates all small-angle-rotation gates with $k > 8$, sufficient

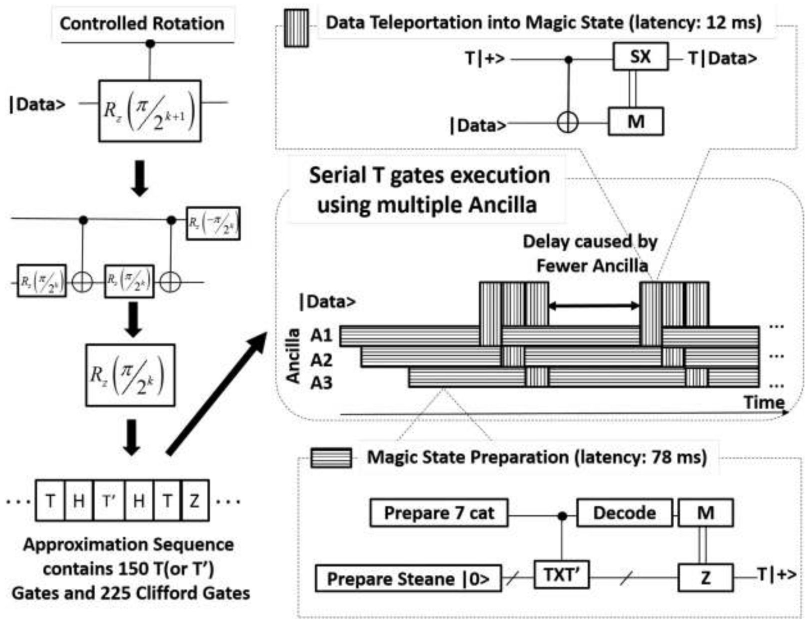

Controlled Rotation
|Data>
$R_z(\pi/2^{k+1})$
$R_z(-\pi/2^k)$
$R_z(\pi/2^k)$
$R_z(\pi/2^k)$
T H T' H T Z
Approximation Sequence contains 150 T(or T') Gates and 225 Clifford Gates
Data Teleportation into Magic State (latency: 12 ms)
T|+>
SX
T|Data>
|Data>
M
Serial T gates execution using multiple Ancilla
Delay caused by Fewer Ancilla
|Data>
Ancilla
A1
A2
A3
Time
Magic State Preparation (latency: 78 ms)
Prepare 7 cat
Decode
M
Prepare Steane |0>
TXT'
Z
T|+>

Fig. 2. Fault-tolerant circuit for a controlled-rotation gate used in AQFT benchmark circuit. Small-angle-rotation gates are approximated by a sequence of T and Clifford gates, and the T gates are performed by magic state preparation and data teleportation.

to factor numbers as large as 4,096 bits [Fowler and Hollenberg 2004]. The resulting truncated QFT is called the approximate QFT (AQFT). The depth of this benchmark circuit is linear in the size of the problem n, and the total number of controlled-rotation gates scales is $16n$. Using the method outlined in Kliuchnikov et al. [2013], we approximate rotations in our AQFT circuit with a sequence of 375 gates (containing 150 T gates), with a precision of 10^{-16}. The resulting approximation sequence consists of T (or $T^{\dagger}$) gates sandwiched between one or two Clifford gates, whose execution time is negligible compared to the T gate. The execution of the T gate proceeds in two steps: preparation of the magic state $T|+\rangle$ and teleportation of data into the magic state. Since state preparation takes a much longer time than teleportation in our system (78ms vs. 12ms, see Table III), we can employ multiple QC units to prepare magic states to simulate *pipelined execution* of T gates (Figure 2). When multiple ancilla qubits are available for the magic state preparation, we can reduce the delay in the execution of the approximation sequence. Using a simple calculation, we can show that the availability of eight logical ancilla qubits completely eliminates any delay. When the error correction procedure is inserted in the approximation sequence, its latency can be leveraged to eliminate the preparation delay with even fewer ancilla qubits.

3. QUANTUM HARDWARE AND QUANTUM ARCHITECTURE MODELS

Quantum hardware describes the physical devices used to achieve computation using a specific technology [Ladd et al. 2010] such as trapped ions, atoms, superconductors, or quantum dots. The efficiency and reliability of QC depend on the characteristics of the chosen technology, such as execution time and fidelity of physical gate operations. We describe the physics of the quantum hardware by a set of device parameters (DPs). In our simulation, the assumed baseline values for these parameters are optimistic, but they can be achieved in the near future through rapid technology advancement. Once quantum hardware technology is specified, we arrange the qubit resources according to their specific roles and their interconnection in order to assemble a large-scale QC. This is captured by the parameters of the *quantum architecture*. For example, the number of qubits dedicated to perform fault-tolerant quantum operations and the specification of communication channels are considered architecture parameters.

3.1. Quantum Hardware Model

We choose to model quantum hardware based on trapped ions for its prominent properties that have been demonstrated experimentally. First, the qubit can be represented by two internal states of the atomic ion (e.g., $^{171}\mathrm{Yb}^{+}$ ion [Olmschenk et al. 2007]), described as a two-level spin system, manipulated by focusing adequate laser beams at the target ion(s). The physical ion qubits can be individually accessible for computation [Crain et al. 2014]. These qubits can be reliably initialized to the desired computational state and measured with very high accuracy using standard techniques. Most importantly, by virtue of the very long coherence time of the ions, qubits can retain their state (memory) for a period of time unparalleled by any other quantum technology. The qubit memory error is modeled as an exponential decay in its fidelity $F \sim \exp(-at)$, where a $(=1/T_{coh})$ is determined by the coherence time of the qubit, and t is the time between quantum gates over which qubit sits idle (*no-op*). The corruption of the qubit state is modeled using a depolarizing channel [Nielsen and Chuang 2000] (equal probability of bit flip, phase flip, and bit-and-phase flip errors). Arbitrary single qubit gates, CNOT, and measurement can be performed with adequate reliability, making trapped ions a suitable candidate for large-scale universal QC.

A single-qubit quantum gate is accomplished by a simple application of laser pulse(s) on the qubit in its original location. A two-qubit gate, on the other hand, requires that both ions are brought in proximity before the laser pulse(s) are applied. In our model, there are two ways to achieve this proximity using two different types of physical resources: the *ballistic shuttling channel* (BSC) and the *entanglement link* (EL) [Monroe et al. 2014]. BSC provides a physical channel through which an ion can be physically transported from its original location to the target location by carefully controlling the voltages of the electrodes on the ion trap chip. This chip can be modeled as a 2D grid of ion-trap cells, as shown in Figure 3. The dimensions of the state-of-the-art ion-trap cell described in [Monroe and Kim 2013] fall in the ~mm size range, and we use $T_{shutt} = 1\mu s$ as the time it takes for an ion to be shuttled through a single cell. In the EL case, an entangled qubit pair (also known as the Einstein-Podolski-Rosen, or EPR, pair) is established between designated proxy “entangling ions” (e-ions) that belong to two independent ion trap chips using a photonic channel. This process is called *heralded entanglement generation* [Duan et al. 2004], since the successful EPR pair generation is announced by the desired output of detectors collecting ion-emitted photons. The resulting EPR pair is used by the actual operand ions as a resource to perform the desired gate via quantum teleportation between two ions that cannot be connected by BSC [Gottesman and Chuang 1999]. It should be noted that the generation time for the EPR pairs is currently a slow process due to technology limitations. This slowness can be compensated for by generating several EPR pairs in parallel using dedicated qubits and hardware. Table I summarizes the DPs used for all the analyses in this article.

3.2. Quantum Architecture Model

Our model is similar to the modular universal scalable ion-trap QC (MUSIQC) architecture [Monroe et al. 2014] shown in Figure 3. It features a hierarchical construction of larger blocks of qubits (called *segments*) composed of smaller units (called *Tiles*), which

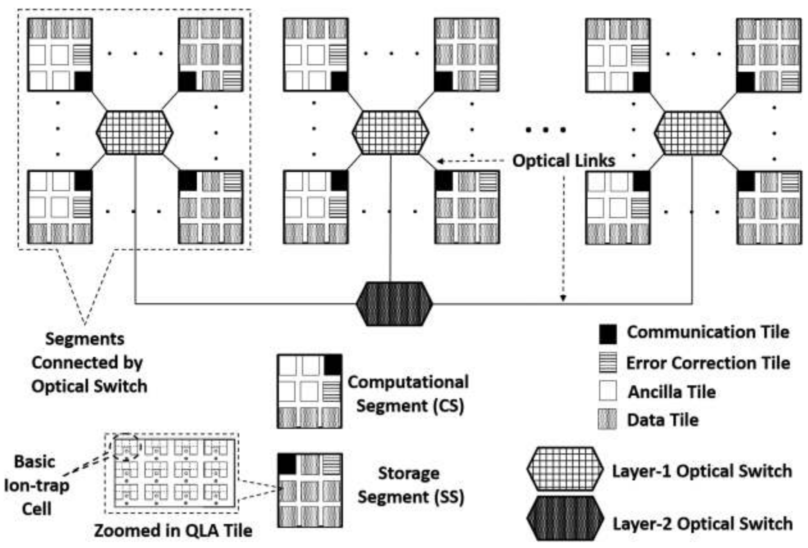


Fig. 3. Overview of the reconfigurable quantum computer architecture analyzed in our performance simulation tool.

Table I. Device Parameters (DPs)

Physical Operation	Time (μs)	Failure Probability
Single qubit	1	10^{-7}
Two qubits (CNOT)	10	10^{-7}
Three qubits (Toffoli)	100	10^{-7}
Measurement	100	10^{-7}
EPR pair generation	5,000	10^{-4}

are connected by an optical switch network. We use the Steane [[7,1,3]] code [Steane 1996] to encode one logical qubit using 7 physical qubits. Additional (ancilla) qubits are supplied to perform error correction and fault-tolerant operations on the logical qubit. We first construct the first layer (L1) logical qubit block containing 22 physical qubits (7 data and 15 ancilla qubits) using Steane encoding. As the size of computation grows, multiple layers of encoding are needed to minimize the impact of increasing noise: with each new layer, the qubit and gate count increase by about a factor of 7. At least two layers of encoding are essential for reliable execution of the sizable benchmark circuits analyzed in our simulations. Therefore, we cluster seven L1 blocks to construct the second layer (L2) logical qubit block containing dedicated qubits to simultaneously carry out error correction operations at the L1 level after every L1 gate. We find that the error correction operation at the L2 level occurs much less frequently, and a dedicated error-correcting ancilla resource at L2 is not necessary for each L2 logical qubit. Therefore, we allocate fewer ancilla qubits at the L2 level for error correction and rely on resource sharing to accomplish fault tolerance at the L2 level.

At the L2 level, we construct four different types of logical qubit blocks using L1 logical blocks, called *L2 Tiles*, that serve different functions in the computation [Metodi et al. 2005]. Each Tile consists of memory cells that provide storage and manipulation of qubits for quantum gates, and BSCs that allow rapid transportation of ions across the memory cells to support the qubit interaction necessary for multi-qubit gates. Tiles are specified by their tasks, such as data storage (Data Tile), state preparation for non-Clifford gates (Ancilla Tile), error correction (EC Tile), and communication

Table II. Composition of L2 Tile

Tile Type	L1 Tiles	Physical Qubits
Data	7	154
Ancilla	15	330
Error Correction	15	330
Communication	$22(=7+15)$	$484+49$

between segments (Communication Tile). At the highest layer of hierarchy, various L2 Tiles are assembled to construct two types of segments: *storage segments* (SSs) and *computational segments* (CSs). A segment is the largest unit that is internally connected using BSCs, and the connections between segments are carried out using optical interfaces. Each segment must contain at least one Communication Tile, one EC Tile, and several Data Tiles. The SSs store qubits when they undergo *no-ops*. On the other hand, CSs contain qubits necessary to perform the complex non-Clifford gates, which requires the ability to prepare the magic states and support the teleportation of data. The magic state is prepared using Ancilla Tiles, which are specific to CSs only. The transportation of data between segments is achieved by teleporting data through the EPR link established by the Communication Tiles of the segments. A network of optical switches [Kim et al. 2003] enables EPR pair generation between any pair of segments in the system. To generate an L2 logical EPR pair, an L2 CNOT consisting of 49 physical CNOT gates is first enacted using 49 physical EPR pairs [Eisert and Plenio 2000] and then error correction is applied to improve its fidelity [Jiang et al. 2009]. The detailed composition of different L2 Tile types is provided in Table II. An L2 Tile can contain up to 600 cells arranged in a 2D grid, and the time to shuttle logical qubits through the Tile is defined to be $60\mu s$.

This architecture scales by adding more qubits to the system in the form of additional segments, which demands a larger optical switch network, at a nominal increase in the latency overhead of EPR pair generation. The optical switches can be connected in a tree-like hierarchy such that the height h of the tree scales only logarithmically with the number of segments. We restrict the size of the optical switches to 1,000 ports [Kim et al. 2003], which can connect up to 20 segments at the lowest level of the optical network tree, using 49 optical ports per L2 Communication Tile. This unique feature enables global connectivity for the entire QC, with the cross-segment communication time almost independent of distance between segments. The communication time scales with the height h of the switch network as 2^{h-1}. We found that $h \leq 3$ is sufficient to connect the maximum number of segments arising in our sizable benchmark circuits, and the corresponding maximum communication time is $5\text{ms} \times 2^{h-1} = 20\text{ms}$.

This globally connected QC architecture with fast communication channels ensures rapid access to CS where computational resources for non-Clifford groups are available. This allows us to designate a finite number of CSs in the overall QC to be shared across the computation. Furthermore, the physical construction of the Data and Ancilla Tiles are nearly identical (one Ancilla Tile can serve as two Data Tiles, and vice versa), so the designation between Data and Ancilla Tiles can be dynamically adjusted during the course of the computation. The total number of segments and the allocation of Data, Ancilla, EC, and Communication Tiles per SS and CS are the architectural parameters of our QC design, denoted by total number of qubits (NTQ) and segments (NSeg), number of computational segments (NCS) (NCS $\leq$ NSeg), number of EC Tiles (NEC) and Communication Tiles (NComm) per segment, and the number of Data (NData) and Ancilla (NAnc) Tiles per CS and SS. Throughout our simulation analysis, we assume NEC $= 1$ since L2 error correction is applied sparsely to Data Tiles, and a single EC Tile can serve several L2 logical qubits. Hence, a concise description of the architecture

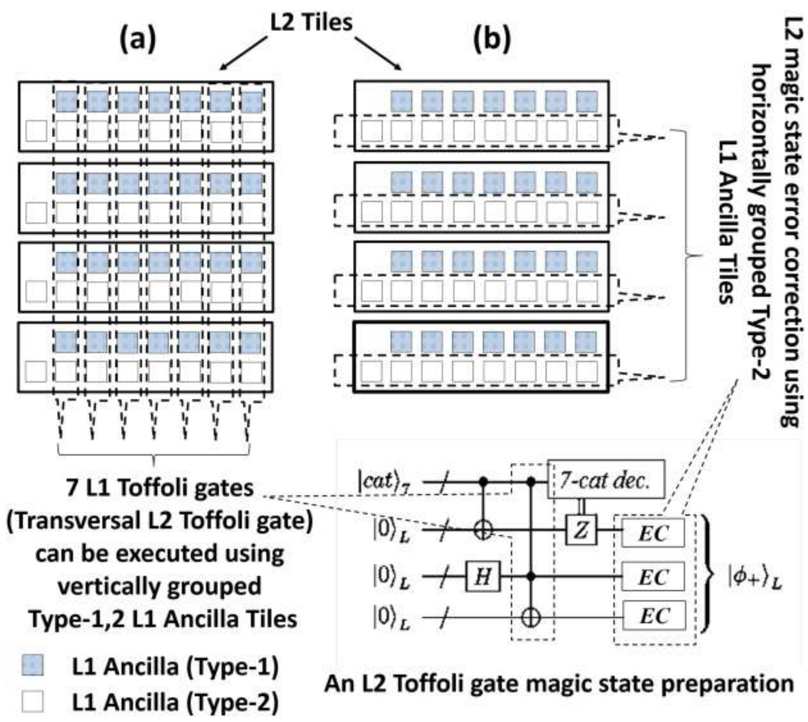

(a)
L2 Tiles
(b)
L2 magic state error correction using horizontally grouped Type-2 L1 Ancilla Tiles
7 L1 Toffoli gates (Transversal L2 Toffoli gate) can be executed using vertically grouped Type-1,2 L1 Ancilla Tiles
L1 Ancilla (Type-1)
L1 Ancilla (Type-2)
|cat⟩7
|0⟩L
|0⟩L
|0⟩L
H
7-cat dec.
Z
EC
EC
EC
|ϕ+⟩L
An L2 Toffoli gate magic state preparation

Fig. 4. An example demonstration of cross-layer resource optimization in L2 Toffoli magic state preparation circuit. In (a), Type 1 and Type 2 L1 Ancilla Tiles are vertically grouped to enact a transversal L2 Toffoli gate, leading to the preparation of the L2 magic state: $|\phi_+\rangle_L$ stored in Type 1 L1 Ancilla Tiles. In (b), Type 2 L1 Ancilla Tiles can be horizontally regrouped to perform L2 error correction on $|\phi_+\rangle_L$.

contains (1) NCS and (2) configuration of CSs specified by three numbers (NData, NAnc, NComm). For SSs, we replace NAnc with 2×NData. Our analysis framework will involve changing architectural parameters and studying their impact on resource-performance tradeoffs. The performance metrics consist of execution time T_{exec} and failure probability $P_{fail} = 1 - P_{succ}$ of the circuit, where P_{succ} is the probability that circuit execution yields a correct result.

3.3. Dynamic Resource Allocation and Cross-Layer Optimization

Our system architecture provides several unique features not considered before that provide a crucial advantage in the resource-performance optimization of the QC design. First, the L2 logical blocks are not identical instantiations of L1 logical blocks: we utilize several L1 logical blocks to construct L2 Tiles with different functionalities. Figure 4 shows the L2 Toffoli magic state preparation in the L2 Ancilla Tiles containing two types of L1 Ancilla Tiles. A Type 1 Ancilla will store a magic state, while a Type 2 Ancilla assumes multiple roles across layers of concatenation. When grouped vertically (Figure 4(a)), Type 2 Ancilla performs transversal of L1 Toffoli gates. The horizontal grouping of these Tiles (Figure 4(b)) performs error correction for the L2 magic state. This “cross-layer optimization” allows efficient utilization of the resources for those tasks (such as error correction at L2) where common resources can be shared. Second, our system allows dynamic reallocation of computational resources during the computation (Data vs. Ancilla Tiles) to adapt the architecture to the computational task at hand to improve the performance, analogous to reconfigurable computing using

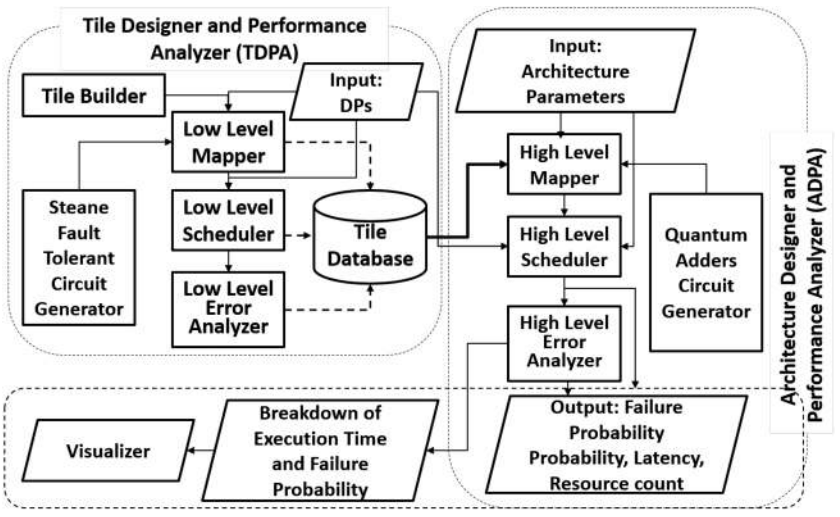

Tile Designer and Performance Analyzer (TDPA)
Tile Builder
Input: DPs
Low Level Mapper
Steane Fault Tolerant Circuit Generator
Low Level Scheduler
Tile Database
Low Level Error Analyzer
Input: Architecture Parameters
High Level Mapper
High Level Scheduler
Quantum Adders Circuit Generator
High Level Error Analyzer
Architecture Designer and Performance Analyzer (ADPA)
Visualizer
Breakdown of Execution Time and Failure Probability
Output: Failure Probability Probability, Latency, Resource count

Fig. 5. Main components of our toolbox.

field-programmable gate arrays (FPGAs) in modern classical computing. Last, utilization of fully distributed resources (such as CS) is enabled by the global connectivity that is a unique feature of our architecture.

4. TOOL DESCRIPTION

4.1. Design Flow and Tool Components

The toolbox flow shown in Figure 5 has two main components: Tile Designer and Performance Analyzer (TDPA) and Architecture Designer and Performance Analyzer (ADPA). Both components share common critical tasks, namely, *mapping*, *scheduling,* and *error analysis*, but the application of these tasks differs according to their objectives and the constraints.

4.1.1. TDPA. TDPA works in the back end of the tool and simulates the fault-tolerant construction of the logical qubit operations using specified DPs. It builds Tiles using the **Tile Builder** by allocating sufficient qubits that can perform the operations specified in the **Steane Fault-Tolerant Circuit Generator** and maps qubits in the circuit to the physical qubits in the Tile using the **Low-Level Mapper**. Then, the **Low-Level Scheduler** generates the sequence of quantum gate operations to be executed in the circuit, including transversal gates, magic state preparation for non-Clifford gates, error correction, and EPR pair generation. Each logical operation is broken down into constituent physical operations, whose performance is simulated on the Tile by adding up the execution time of each gate subject to circuit dependencies and resource constraints. The **Low-Level Error Analyzer** computes the failure probability of the specified fault-tolerant quantum gates based on the scheduled circuit by counting the number of ways in which physical errors can propagate to cause a logical error in the qubit [Aliferis et al. 2005; Ahsan et al. 2013]. The Tile parameterized by DP and the computed performance metrics is stored in the **Tile Database**. Table III shows the performance of the unified L2 Tile (which can act as Data, EC, Ancilla, and Communication Tiles) computed by the TDPA for baseline DPs given in Table I.

4.1.2. ADPA. ADPA is the front end of the tool that interfaces with the user. It takes architecture parameters specified by the user (e.g., NCS, NEC, and NComm) as inputs

Table III. L2 Tile Performance Numbers: The L1 Error Correction Takes 687μs and Fails with Probability 1.66×10^{-10}

Logical Operation	$T_{exec}(\mu s)$	$1 - P_{succ}$
Pauli (X, Z)	1	1.15×10^{-18}
Hadamard	4	1.15×10^{-18}
CNOT	10	4.74×10^{-18}
Transversal (bitwise) Toffoli	4, 210	1.1×10^{-17}
7-qubit Cat-State prep.	6, 500	3.75×10^{-18}
Measurement	11, 900	6.14×10^{-17}
L2 error correction	48, 900	4.58×10^{-16}
State prep. ($\lvert\bar{0}\rangle$,$\lvert\bar{+}\rangle$)	34, 500	1.6×10^{-16}
State prep. ($T\lvert\bar{+}\rangle$)	78, 100	4.23×10^{-16}
EPR pair generation	$T_{gen} + 50, 800$	1.08×10^{-11}

and (1) builds and connects segments using Tiles supplied by TDPA to implement the benchmark application on hardware configuration and (2) evaluates performance of the benchmark for given architecture parameters. First, the **Quantum Circuit Generator** generates the benchmark circuits from the given algorithms (QCLA, QRCA, and AQFT). Then, the **High-Level Mapper** maps logical qubits to Tiles in the segments, maximizing the locality by analyzing their connectivity patterns in the circuit, assigning frequently interacting qubits to the Tiles in the same segment. This is achieved by solving an *optimal linear arrangement problem* using an efficient graph-theoretic algorithm [Juvan and Mohar 1992] to generate the initial map of the Data Tiles in the segments. Using this map, the **High-Level Scheduler** generates the sequence of gates for the circuit execution by solving the standard *resource-constraint scheduling problem* in which resources and constraints are given by architecture parameters. The Scheduler minimizes the execution time by reducing the *circuit critical path* through maximum utilization of available resources (Ancilla and Communication Tiles) in the segments. The non-Clifford gates require operands to be available in the same CS before being scheduled. Therefore, the operand located in remote Data Tiles needs to be teleported into the local Data Tile of the CS, while Ancilla Tiles prepare the magic state for execution. NCS determines how many non-Clifford gates can be scheduled in parallel, while NComm determines how quickly Tiles can be teleported across the segments. Therefore, the delays in gate scheduling depend mainly on architecture parameters. The critical path of the circuit consists of these delays and the gate execution time. The complete list of latencies arising due to insufficient resources and architecture configuration is as follows:

—*Ancilla Delay (D_{ANC})*: Delay due to the magic state preparation (fewer NCSs or fewer Ancilla Tiles per segment)
—*Shuttling Delay (D_{SHUT})*: Delay due to the transportation of operand qubits of the gate, through BSC inside the segment
—*Tel Delay (D_{TEL})*: Delay due to the logical EPR pair generation for communication (Fewer NComms)
—*Cross-Seg-Swap Delay (D_{SWP})*: Delay due to the cross-segment swapping (fewer NComms or large number of smaller segments)

The Scheduler also minimizes P_{fail} by scheduling error correction on Data Tiles at regular intervals when they sit idle (*no-op*). Once a complete schedule of logical operations is obtained, the **High-Level Error Analyzer** computes the overall P_{fail}. Since we cannot correct for logical failure of the operation, the Error Analyzer simply computes $\prod_{i=1}^{i=N}(1 - P_{Li})$, where P_{Li} is the failure probability of the ith logical gate and N is the total number of logical operations in the circuit. The Error Analyzer also tracks the operational source of each P_{Li} so that $1 - P_{succ}$ can be broken down into the following important noise components:

—*Shuttling Error* P_{SHUT}: Errors due to the qubit shuttling through noisy BSC
—*Teleportation Noise* P_{TEL}: Errors due to the infidelity of an EPR pair for communication
—*Memory Noise* P_{MEM}: Errors due to the fidelity degradation of qubit during *no-op*
—*Gate Noise* P_{GATE}: Errors due to the noisy quantum gates and measurements

4.2. Splitting Performance Metrics

Our tool can output the constituents of T_{exec} by keeping track of the different types of latency overhead composing the critical path of the quantum circuit execution. Our iterative scheduler selects a gate for execution during each iteration and updates the critical path. When a gate is selected for execution, the operand qubits should be available for computation; otherwise, it will be delayed. If we let T_{start} be the time at which the gate was executable but operands were available at T'_{start} where $T'_{start} \geq T_{start}$, then the time at which the gate execution is complete is given by $T_{finish} = T'_{start} + T_{exec}$. The gate execution selected in scheduling iteration i will update the critical path if $T_{finish} > T^{i-1}_{TotalExec}$, where $T^{i-1}_{TotalExec}$ is the total execution time (the length of the critical path) computed in iteration $i - 1$.

When the gate lies on the critical path, the Scheduler computes $\Delta T = T_{finish} - T^{i-1}_{TotalExec}$ and $\Delta D = T'_{start} - T_{start}$, which can be broken down as $\Delta D = D_{ANC} + D_{SHUT} + D_{TEL} + D_{SWP}$. To define the components of the critical path in iteration $i - 1$, we split $T^{i-1}_{TotalExec}$ into its components as $T^{i-1}_{TotalExec} = T^{i-1}_{ANC} + T^{i-1}_{SHUT} + T^{i-1}_{TEL} + T^{i-1}_{SWP} + T^{i-1}_{GATE}$. For iteration i, these components are updated as follows:

—*Ancilla Preparation Overhead*:
$T^{i}_{ANC} = T^{i-1}_{ANC} + \frac{D_{ANC}}{\Delta D + T_{exec}} \times \Delta T$
—*L2 Shuttling Overhead*:
$T^{i}_{SHUT} = T^{i-1}_{SHUT} + \frac{D_{SHUT}}{\Delta D + T_{exec}} \times \Delta T$
—*Teleportation Overhead*:
$T^{i}_{TEL} = T^{i-1}_{TEL} + \frac{D_{TEL}}{\Delta D + T_{exec}} \times \Delta T$
—*Segment Swap Overhead*:
$T^{i}_{SWP} = T^{i-1}_{SWP} + \frac{D_{SWP}}{\Delta D + T_{exec}} \times \Delta T$
—*Gate Overhead*:
$T^{i}_{GATE} = T^{i-1}_{GATE} + \frac{T_{exec}}{\Delta D + T_{exec}} \times \Delta T$

In the case of the critical path, we also update the total execution time $T^{i}_{TotalExec} = T_{finish}$. An example breakdown of execution time is shown in Figure 6, where magic state preparation for Toffoli gates and cross-segment swapping delay the execution of gates in turn and make up the bulk of the critical path. Our tool can also decompose the failure probability P_{fail} into its components since the Error Analyzer tracks noise sources from the schedule. For any operation type *op*, we compute $P^{op}_{fail} = 1 - \prod_{i=1}^{i=n^{op}} (1 - P^{op}_{Li})$, where *op* can be *shuttling*, *memory*, *teleportation,* or *gate*, and n^{op} and P^{op}_{Li} are the total operation count and failure probability for *op*, respectively.

4.3. Tool Validation and Performance

Individual components of the tool can easily be verified for correctness by running these for known circuits and comparing their output with anticipated results. Overall validation can be performed by using visualization and the breakdown of performance

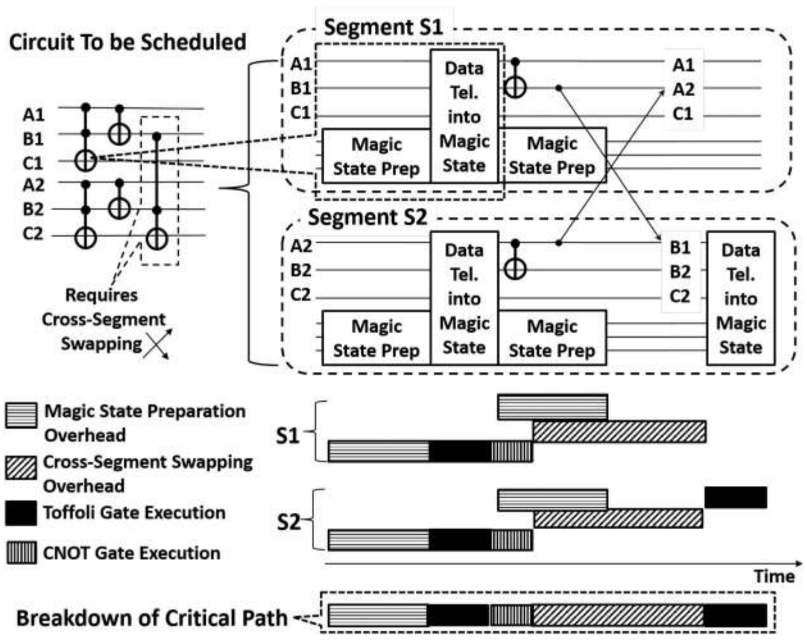

Circuit To be Scheduled
A1
B1
C1
A2
B2
C2
Requires Cross-Segment Swapping
Segment S1
Segment S2
Data Tel. into Magic State
Magic State Prep
Magic State Preparation Overhead
Cross-Segment Swapping Overhead
Toffoli Gate Execution
CNOT Gate Execution
S1
S2
Time
Breakdown of Critical Path

Fig. 6. An example shows tracking of latency overheads composing the critical path of the circuit.

metrics for different types of benchmarks. Tool efficiency mainly arises from taking advantage of the repetitive nature of fault-tolerant procedures and circuit breakdown of universal quantum gates. Performance of low-level circuit blocks is precomputed and stored in the database, and used for simulating the behavior of high-level circuits. For instance, TDPA can be run offline to generate parameterized Tiles, which are used to efficiently run components of ADPA such as the High-Level Scheduler, Error Analyzer, and Visualizer. Similarly, the initial mapping of L2 qubits on these Tiles is generated from a computationally intensive optimization algorithm [Juvan and Mohar 1992], but once generated, can be efficiently processed by the High-Level Scheduler to generate subsequent gate schedules. Thus, we can run the High-Level Mapper offline as well. Consequently, the running time of the tool is decided by that of the High-Level Scheduler and Error Analyzer, which mainly depends on architecture resources and benchmark size. The results discussed in Section 5 show that the performance improvement saturates once resource investment exceeds a certain value, and the maximum size of the overall system we have to simulate is mainly dictated by the size of the application circuit.

Figure 7 shows the running time of the tool as a function of circuit size. The data is collected by running the tool on a computer system containing an Intel$^{(R)}$Core$^{(TM)}$ $i3$ 2.4GHz processor and 2GB RAM. To incorporate the dependency of tool running time on the available architecture resources, we choose the configuration containing maximum resources in order to obtain the typical worst-case running time of the tool. In this configuration, we allocate maximum Ancilla and Communication Tiles per Data Tile and allow all Segments to act as Computational Segments. Under these conditions, Figure 7 shows that the performance simulation of a 2,048-bit circuit can be completed in less than 1.5 minutes. Thanks to the efficiency in the performance simulation, our tool can explore a large QC design space in a reasonable amount of time.

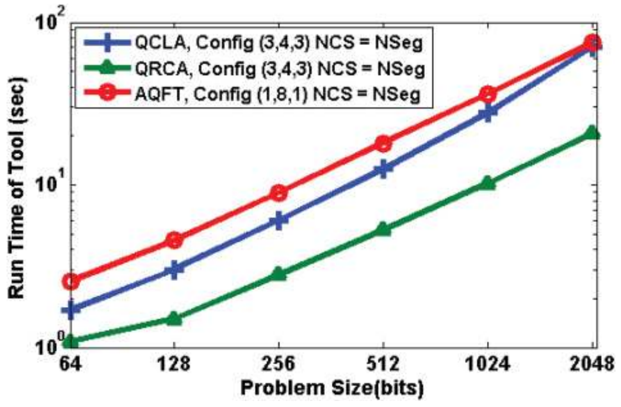


Fig. 7. The running time of ADPA (excluding Visualizer) as a function of benchmark application size. With full Visualization, the running times increase no more than seven times.

5. SIMULATION RESULTS

We first analyze the relationship between resources (qubits) and performance as a function of benchmark size for scalability. We consider the system architecture *resource-performance* (RP) *scalable* if the increase in resources necessary to achieve the expected behavior of the performance (execution time) grows linearly with the size of the benchmark while maintaining $P_{succ} \sim O(1)$. In the absence of hardware resource constraints, the expected execution time for the QRCA and AQFT grows linearly, while that for QCLA grows logarithmically, as the problem size grows. The execution time could grow much more quickly in the presence of resource constraints, in which case the system is not considered RP scalable.

In the first step, we present a set of simulations to quantify the RP scalability of the proposed MUSIQC architecture for benchmark circuits and analyze the constituents of performance metrics. In the next step, we study the impact of limited resources and architecture parameters on the performance of fixed-size benchmarks. This will provide guidelines to find an optimized design under limited resources. In the last set, optimum designs are obtained under resource constraints, for effectively executing the benchmark circuits.

5.1. Resource-Performance Scalability

Figures 8(a), 9(a), and 10(a) show T_{exec} and the total number of physical qubits (NTQ) plotted against benchmark size for QCLA, QRCA, and AQFT, respectively, and corresponding P_{fail} values are shown in Table IV. We consider two architecture configurations (NData, NAnc, NComm) = (3,4,1) and (12,4,3) for the adders, and configurations (1,4,1) and (1,8,1) for AQFT. When benchmark size increases by a factor of x, we expect P_{fail} to increase by at least the same amount (total logical gate operations increase x-fold), while execution time scales as the depth of the circuit. Indeed, Table IV confirms this trend in P_{fail} for all benchmarks. On the other hand, as the problem size doubles, both T_{exec} for QRCA and AQFT increase twofold (linear curve for T_{exec} in Figures 9(a) and 10(a)). For QCLA, T_{exec} increases roughly by a constant amount (logarithmic curve in Figure 8(a)) as expected. Since this performance is achieved for the same increase in total number of qubits as that of the problem size, our architecture shows RP scalability.

By demonstrating RP scalability for two different architecture configurations with NCS = NSeg, we have presented varying performance levels. The impact of these configurations can be understood by analyzing the breakdown of performance metrics

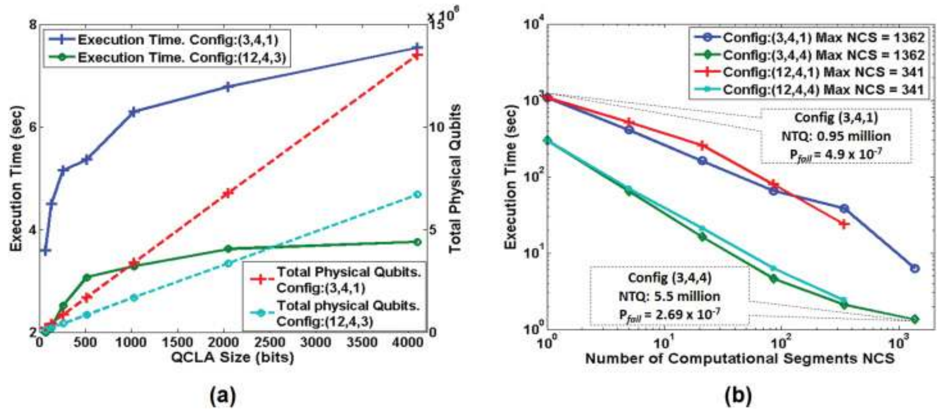


Fig. 8. QCLA execution time (a) scaling under no constraints on physical resources; T_{exec} and total physical qubits (NTQ) consumed are plotted as a function of benchmark size, NCS = NSeg. (b) Variation with NCS for different NComms, showing tradeoffs between resources and T_{exec}.

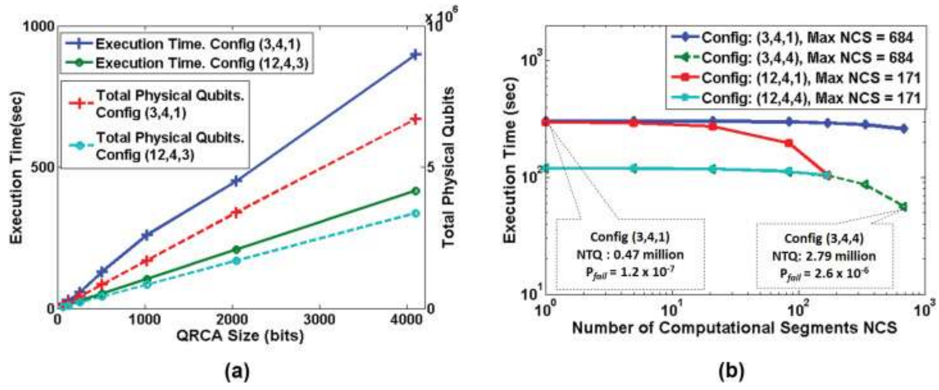


Fig. 9. QRCA execution time (a) scaling under no constraints on physical resources; T_{exec} and total physical qubits (NTQ) consumed are plotted as a function of benchmark size, NCS = NSeg. (b) Variation with NCS for different NComms, showing tradeoffs between resources and T_{exec}.

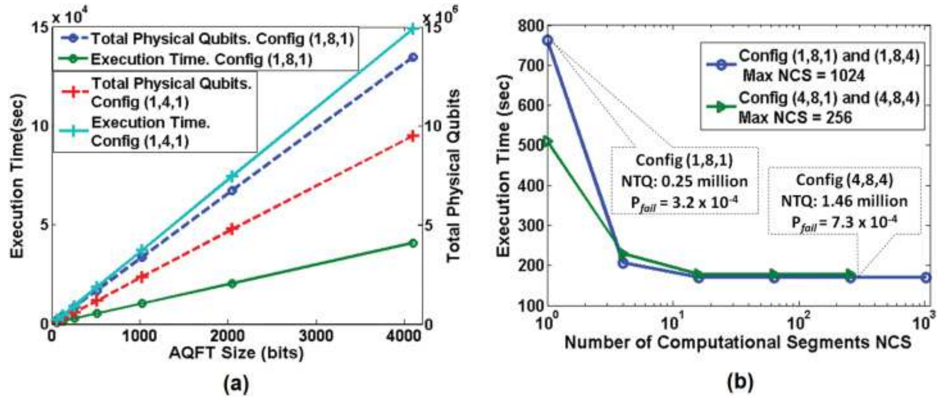


Fig. 10. AQFT execution time (a) scaling under no constraints on physical resources; T_{exec} and total physical qubits (NTQ) consumed are plotted as a function of benchmark size, NCS = NSeg. (b) Variation with NCS for different NComms, showing tradeoffs between resources and T_{exec}.

Table IV. Failure Probability P_{fail} for Corresponding Data Points of Figures 8(a), 9(a), and 10(a)

	Failure Probability					
	QCLA (10^{-8})		QRCA(10^{-8})		AQFT(10^{-5})	
Size	Config: (3,4,1)	Config: (12,4,3)	Config: (3,4,1)	Config: (12,4,3)	Config: (1,4,1)	Config: (1,8,1)
64	1.38	0.78	0.62	0.16	1.9	4.3
128	2.81	1.6	1.27	0.35	3.9	8.9
256	5.77	3.4	2.62	0.72	8.04	18.1
512	11.7	7.5	5.91	2.0	16.3	36.2
1024	23.1	15	12.9	4.5	32.7	73.5
2048	47.1	30.3	27.7	7.9	65.4	147.6
4096	93.8	61.1	57.2	18.7	130.1	295.2

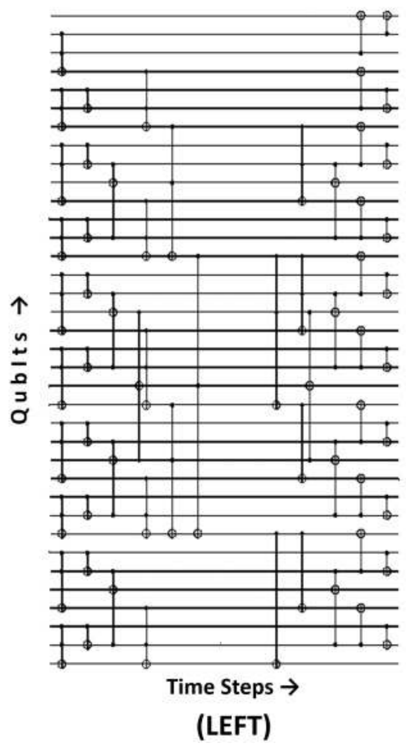
Table V. Breakdown of Performance Metrics: *Texec* of Figures 8(a), 9(a), and 10(a) and P_{fail} of Table IV for 1,024-bit Benchmark (NCS = NSeg)

Bench-mark	Config	Execution Time Breakdown (%age)					Failure Probability Breakdown (%age)			
		$\%T_{ANC}$	$\%T_{SHT}$	$\%T_{TEL}$	$\%T_{SWP}$	$\%T_{GATE}$	$\%P_{TEL}$	$\%P_{SHUT}$	$\%P_{MEM}$	$\%P_{GATE}$
QCLA	3,4,1	1.8	0	31.0	61.2	5.8	89.3	9.4	0.6	0.5
	12,4,3	41.1	0	13.5	23.9	21.3	86.5	12.0	0.9	0.5
QRCA	3,4,1	6.1	0	80.0	10.7	3.1	80.6	8.3	10.4	0.4
	12,4,3	76.3	0	0.12	0.03	23.5	74.9	10.7	11.9	2.3
AQFT	1, 4, 1	28.9	47.7	0	0	23.2	0	98.7	0	1.2
	1, 8, 1	0.8	22.5	0	0	76.6	0	99.4	0	0.5

in Table V. The significant contribution of the overhead T_{ANC} for adder configuration (12,4,3) and AQFT configuration (1,4,1) shows that magic state preparation is the dominant component of T_{exec} due to insufficient Ancilla Tiles in CS. This overhead can be substantially reduced either by increasing NAnc (configuration (1,8,1) for AQFT) or by increasing the ratio of NData to NAnc (configuration (3,4,1) for adders). However, the configuration (3,4,1) exposes EPR pair generation overhead captured by T_{TEL} and T_{SWP}. This is due to the large number of cross-segment CNOT gates and qubit swapping operations required to bring all Toffoli operands to the same segment. Hence, for adders, frequent cross-segment communication explains the higher contribution of teleportation error (P_{TEL}) in the failure probability. For AQFT, configuration (1,8,1) highlights shuttling overhead T_{SHUT} as T gates make up the bulk of the operations. The scheduling of T gates in the long approximation sequence leads to a large number of interactions between Data and Ancilla Tiles through BSC in the segment. This intensive localized communication makes (P_{SHUT}) the only noticeable component of P_{fail}. In conclusion, we have shown RP scalability of the architecture when performance is bottlenecked by different hardware constraints for various benchmarks. This shows that the architecture can utilize additional resources efficiently to achieve adequate performance when running quantum circuits of larger sizes.

5.2. Resource-Performance Tradeoffs

Now that we have quantified RP scalability, we examine the impact of reduced resources on the performance by constraining architecture parameters. We fix the size of the benchmarks to 1,024 bits and vary NComm, NCS, and the configuration to observe changes in T_{exec}. Figure 10(b) shows that for AQFT, (1) T_{exec} does not change with NComm since it is not restricted by cross-segment communication resources, and (2) T_{exec} initially decreases sharply as NCS increases but flattens when NCS reaches



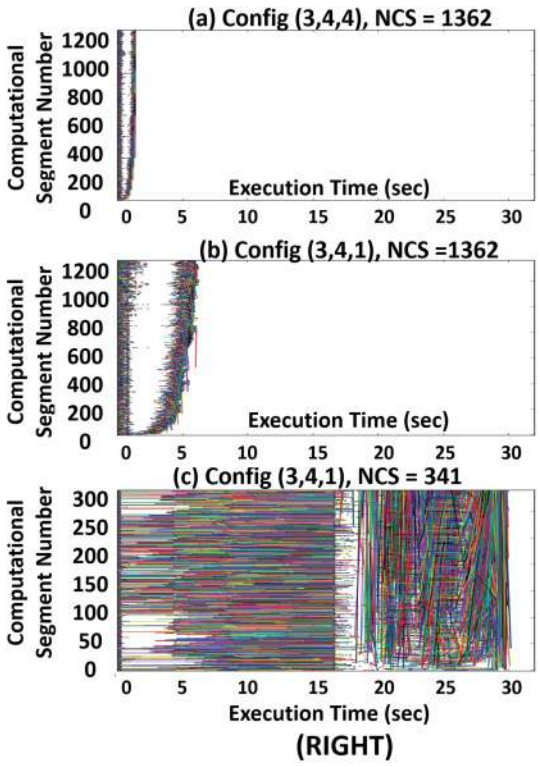


Fig. 11. (LEFT) Sample QCLA circuit, and (RIGHT) visual representation of latency overhead for Computational Segments of 1,024-bit QCLA architecture with (a) configuration (3,4,4), NCS = NSeg = 1,362, (b) configuration (3,4,1), NCS = NSeg = 1,362, and (c) configuration (3,4,1), NCS = 341.

∼16, mainly due to insufficient parallelism in the circuit. The QRCA curves in Figure 9(b) remain unchanged as NCS increases until NCS approaches NSeg where T_{exec} shows noticeable decline. However, the overall decrease remains within a factor of only about 3, due to the serial nature of the circuit dependencies. By comparing curves for different configurations ((12,4,1) vs. (3,4,1) for QRCA and (1,8,x) vs. (4,8,x) for AQFT), we find that clustering more Data Tiles in the CS generally reduces T_{exec} due to fewer delays in cross-segment operand swapping.

Figure 8(b) shows that T_{exec} of QCLA decreases exponentially with NCS. In contrast to QRCA, the large number of concurrently executable Toffoli gates in QCLA demands much higher NCS. Furthermore, T_{exec} also decreases with higher NComm as the large number of cross-segment teleportations consume more communication resources. Figures 11(a) through 11(c) provide a pictorial description of these trends generated through our visualization tool. The visualization highlights different types of latencies arising from the resource constraints in scheduling within each CS. The visualization tool draws a line, in execution time (horizontal axis) – qubit location (vertical axis) plane, between each point where a logical qubit in the circuit requires additional resources to proceed to the next step (such as a magic state for non-Clifford group gates prepared in an Ancilla Tile, or EPR pairs for teleportation in Communication Tiles), and a point where the required resource becomes available. As a consequence, the horizontal lines represent delay in magic state preparation, while nonhorizontal lines indicate delays in cross-segment teleportation. The corresponding latencies can be derived by projecting these lines on the horizontal axis. When sufficient NCS and

NComm resources are provided, as in Figure 11(a), there is little delay and the circuit execution is fast. However, when NComm is reduced from 4 to 1, as in Figure 11(b), teleportation latency causes a 4.5-fold increase in T_{exec}. The same amount of increase occurs in Figure 11(c) when NCS is reduced from 1,362 to 341. Long horizontal lines indicate delays due to fewer Ancilla Tiles available for Toffoli magic state preparation, which increases the overall T_{exec} by another factor of about 6.

By comparing Figures 8, 9, and 10, it is easy to conclude that QCLA is the most resource-hungry benchmark, while AQFT is the least. We also note that P_{fail} values do not tend to improve substantially when we provide more architecture resources. It can be shown that a substantial decrease in P_{fail} can be achieved by improving the relevant device parameters (DPs) that contribute to the dominant noise sources [Ahsan and Kim 2015]. These sources can correctly be identified once we have invested sufficient resources for scheduling error correction and chosen optimized architecture configuration that minimizes T_{exec}. In the following subsection, we concentrate on T_{exec} only and return to optimizing P_{fail} in the last subsection.

5.3. Performance Scaling Under Limited Resources

The increase in T_{exec} due to constrained resources can be compensated to some extent by designing an optimized architecture that allocates more resources toward the root cause that limits the performance. The choice of optimized architecture configuration varies across benchmarks, since both performance bottlenecks and resource utilization depend strongly on the structure of the application circuit. By plotting optimized T_{exec} against benchmark size using a fixed resource budget, we can determine (1) the largest circuit size that can be scheduled and executed, (2) the trend for the optimized performance as a function of problem size, and (3) the choice of configuration that generally obtains the optimized performance for the specified benchmark circuit. To adequately obtain these insights, we consider an example where we restrict our total qubit resource (NTQ) to 1.5 million physical qubits. We also restrict the number of physical qubits contained in the segment; small segment (SSeg) will contain up to 5,000 physical qubits, while large segment (LSeg) will contain up to 10,000 physical qubits. For each benchmark, we obtain two plots, one for each segment size.

Figure 12 shows T_{exec} of running the benchmark circuits on this system, as a function of problem size. The T_{exec} for each data point was minimized by tool-assisted search through all feasible combinations of architecture configurations and NCS parameters. These optimized architecture designs are shown in Table VI. It is interesting to compare Ancilla and Communication Tiles invested in the optimized designs. For example, the configurations (19,12,6) and (8,8,2) for QRCA contain a higher Ancilla-to-Communication Tiles ratio, as compared to (30,8,5) and (5,4,5) for QCLA. This indicates that in contrast to the QRCA case, both Ancilla and Communication Tiles are equally vital for the QCLA performance. The AQFT architecture (1,8,1) with sufficient NCS is a natural choice, since the only relevant resource for this circuit is the large number of Ancilla Tiles to schedule the long sequence of T gates necessary to approximate the small-angle rotations.

We observe that regardless of the segment size, T_{exec} of the least resource-demanding benchmark of the three, namely, AQFT, scales perfectly with problem size at least up to 4,096 bits. This is due to the fact that the optimized configuration for AQFT can easily be met within the qubit resource budget. However, in the case of adders, segment size is an important parameter that impacts the performance. Larger segments open up a greater search space for optimizing design selection for resource-intensive circuits. For both QCLA and QRCA, T_{exec} scales better for larger segments. For smaller segments, QCLA performance shows significant degradation as the problem size begins

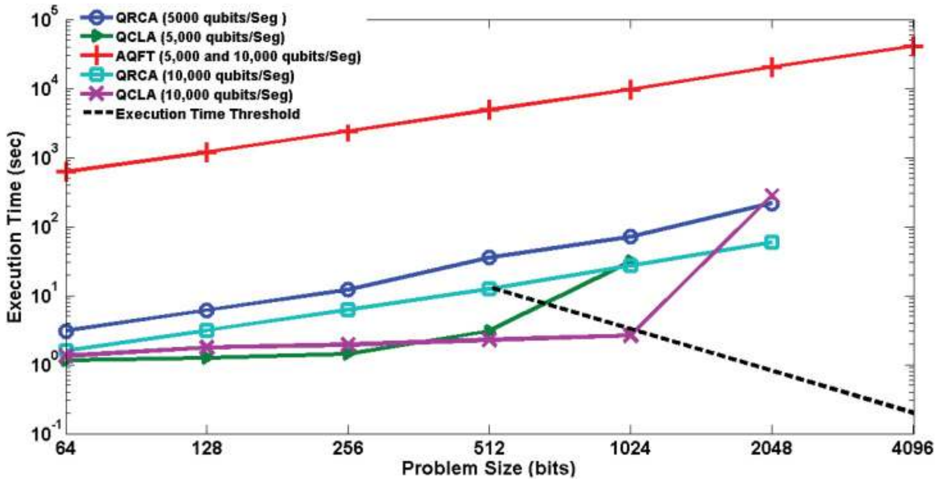


Fig. 12. T_{exec} for optimized architectures plotted against benchmark size for different segment sizes. The resource budget is 1.5 million physical qubits. Optimized architecture configurations are shown in Table VI.

Table VI. Optimized Architecture Configurations for Figure 12

	Optimized Architecture Configuration				
	QCLA		QRCA		AQFT
Size	10,000 qubtis/ Seg	5,000 qubits/ Seg	10,000 qubtis/ Seg	5,000 qubits/ Seg	5000 or 10,000 qubits/Seg
64	(30,8,5) NCS = 8	(5,4,5) NCS = 50	(19,12,6) NCS = 7	(8,8,2) NCS = 17	(1,8,1) NCS ≥ 16
128	(30,8,5) NCS = 17	(5,4,5) NCS = 101	(19,12,6) NCS = 14	(8,8,2) NCS = 33	(1,8,1) NCS ≥ 16
256	(30,8,5) NCS = 34	(5,4,5) NCS = 203	(19,12,6) NCS = 27	(8,8,2) NCS = 65	(1,8,1) NCS ≥ 16
512	(30,8,5) NCS = 68	(12,4,3) NCS = 170	(19,12,6) NCS = 54	(8,8,2) NCS = 131	(1,8,1) NCS = 16
1024	(30,8,5) NCS = 137	(12,4,3) NCS = 245	(19,12,6) NCS = 108	(8,8,2) NCS = 257	(1,8,1) NCS = 16
2048	(48,4,2) NCS = 25	*NA*	(19,12,6) NCS = 96	(12,4,3) NCS = 244	(1,8,1) NCS = 16
4096	*NA*	*NA*	*NA*	*NA*	(1,8,1) NCS = 16

to increase. The logarithmic depth of the most resource-intensive benchmark (QCLA) is restricted to 256 bits and 1,024 bits for SSeg and LSeg, respectively. After that, T_{exec} shows a sudden rise with problem size and even surpasses the corresponding QRCA performance, as lack of resources leads to substantial delays in executing the parallel gate operations that enable the logarithmic-depth adder. The largest adder benchmark that can be scheduled on this hardware is the 2,048-bit adder, where the performance is substantially slower and QRCA outperforms QCLA by a factor of 4.

5.4. Design Optimization by Tuning Device Parameters

5.4.1. Reducing the Execution Time. The largest integer factorized using a classical computer to date is 768 bits, consuming 30 months on a cluster of several hundred processors for the factorization [Kleinjung et al. 2010]. Kleinjung et al. estimated that factorizing a 1,024-bit number is 1,000 times harder than a 768-bit number, and larger integers are unlikely to be factored in the near future. Based on these reasons, we choose a performance criterion so that a QC is considered practical if integers larger than 768 bits can be factored in less than 5 months.

The total execution time of Shor's algorithm is heavily dominated by the modular exponentiation circuit constructed using adders. Efficient implementation of modular exponentiation for 512-, 1,024-, and 2,048-bit integers requires 1, 4, and 16 million calls to an adder circuit, respectively [Van Meter and Itoh 2005]. Given the size limit of our QC, the adders are assumed to be executed sequentially since parallel implementation of adders will require additional qubit resources. The upper bound for the execution time of an adder circuit to complete the modular exponentiation in 5 months is shown as a dotted line in Figure 12. If the execution time of an adder falls below this line, an integer of this size can be factored in less than 5 months.

Figure 12 shows that a 1,024-bit number can be factored in less than 5 months with a 1.5-million-qubit QC, as the T_{exec} of the QCLA lies below the black dotted line. However, the T_{exec} of 2,048-bit QRCA and QCLA (58.5s and 270s, respectively) are both significantly higher than the required execution time of 0.8s. We first attempt to reduce T_{exec} by increasing our resource budget. With twice as many qubits, the optimized architecture configuration for QCLA changes from (48,4,2), NCS = 25, to (30,8,5), NCS = 273, and shows nearly a 100x decrease in T_{exec} (from 270s to 2.76s). This remarkable reduction arises as the additional delays in QCLA execution time due to limited parallel operation are eliminated, and logarithmic depth performance is restored. The QRCA shows only a nominal reduction from 58.5s to 50.1s, as the resource limitation is not the main cause of slow execution time. The logarithmic depth QCLA is the only choice that can meet the threshold execution time criterion, if T_{exec} can be further reduced by a factor of 4. Unfortunately, we find that any further increase in resources (and new architecture configuration) fails to gain additional reduction in T_{exec}.

The failure to achieve performance improvement through additional resources highlights the role of DPs in the design space [Ahsan and Kim 2015]. The set of DPs that affect the speed of the quantum circuit include the latency of physical operations. We find that when physical gate (and Measurement) times and qubit shuttling latency are reduced 10x, the execution time declines to $0.68s < 0.8s$, meeting the required adder execution time. The total execution time for 2,048-bit integer factorization in this case can be approximated as the sum of time spent in 16 million calls to the adder ($0.68s \times 16 \times 10^6 \approx 128$ days) and the T_{exec} of a single run of the 4,096-bit AQFT (less than a day). Therefore, the proposed QC design can factor a 2,048-bit number in less than 5 months.

5.4.2. Reducing the Failure Probability. In order to ensure that the entire Shor's algorithm is reliably executed, we require the sum of the failure probability of a 2,048-bit modular exponentiation circuit and a 4,096-bit AQFT to be sufficiently low. Table IV shows already adequate P_{fail} in the range $\sim O(10^{-4}) - O(10^{-3})$ for AQFT when a baseline value of $1\mu s$ was assigned to the qubit shuttling latency (T_{shutt}). As T_{shutt} reduces to $0.1\mu s$ in our current design to lower the execution time of the adder, P_{fail} of the 4,096-bit AQFT falls well below 10^{-4}. Therefore, the goal of reducing the failure probability of the full Shor's algorithm translates into curtailing the failure probability of the modular exponentiation circuit. This in turn requires the P_{fail} of each adder call to be less than

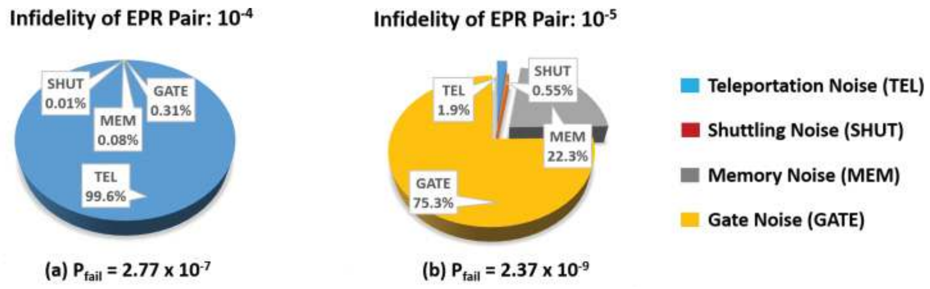


Fig. 13. The breakdown of and reduction in the failure probability of 2,048-bit QCLA for the configuration [30,8,5], NCS = 273, with $T_{shutt} = 1\mu s$ and the baseline physical measurement and multi-qubit gate times reduced by 90%. The original failure probability shown in (a) is reduced from 2.77×10^{-7} to 2.37×10^{-9} in (b) only by decreasing the infidelity of the EPR pair for cross-segment communication.

a certain threshold value so that overall failure probability is reduced far enough to meet the design criterion. The 2,048-bit integer factorization consumes 16 million calls to the adder, and therefore we require $P_{fail} << 6.67 \times 10^{-8}$ for each adder execution. The P_{fail} for the design optimized for T_{exec} ((30,8,5), NCS = 273) is 2.77×10^{-7}.

In order to lower the failure probability, we can either add one more layer of encoding or reduce the noise level in the physical device components. Adding a layer of encoding will require at least a 7x increase in qubit resources, which enormously expands the scale of integration. In addition, the T_{exec} is also significantly inflated (Table III shows that L2 error correction takes 70x more time than L1 error correction). Therefore, increasing the number of layers of encoding achieves a reduction in P_{fail} at the expense of far greater T_{exec} and resources. On the other hand, reducing the noise level in physical device components can decrease P_{fail} without compromising T_{exec}. We exploit the tool-supplied breakdown of P_{fail} based on the fidelity of component physical operations to systematically improve the success probability of the factorizing task.

Figure 13 shows the breakdown of failure probability for 2,048-bit QCLA, where over 99% of the failure originates from Teleportation Noise. The device parameter that directly affects the Teleportation Noise is the infidelity of EPR pairs for cross-segment communication. By reducing the infidelity from 10^{-4} to 10^{-5}, we gain more than a 100x reduction in the failure probability as shown in Figure 13(b). A further decrease in P_{fail} can be obtained by tuning DPs affecting the Gate and Memory Noise.

In conclusion, we showed that we can lower the failure probability of a 2,048-bit QCLA circuit to 2.37×10^{-9} ($<< 6.67 \times 10^{-8}$) by tuning the DPs. This gives an overall failure probability of modular exponentiation of about 3.8%. The P_{fail} of the 4,096-bit AQFT is negligible compared to this value, and the overall failure probability does not exceed 4%. Hence, we have shown that the optimized adder architecture with the appropriately tuned DPs can be used to construct reliable QC to execute the 2,048-bit Shor algorithm.

6. TOOL ENHANCEMENTS AND EXTENSIONS

Our architecture case study hinges on our versatile tool, which can be extended along several directions for future research. In this section, we describe how the modular implementation of the tool supports its general extensibility to encompass a variety of quantum application circuits, QECCs, and device technologies.

6.1. Other Quantum Error-Correcting Codes

By generating relevant new Tiles, our tool can handle prominent QECCs other than concatenated code used in this study (Steane [[7,1,3]] code). For example, a vastly different class of QECCs known as Topological Quantum Code (TQC) [Kitaev 2003] can be incorporated in the general Tile framework used in our tool (Section 3.2), which is adequately equipped to handle long-distance and nearest-neighbor qubit interactions. The 2D-grid-based layout of our Tile acts as a substrate for the fault-tolerant TQC operations. The "hole"-induced creation of the logical qubit(s) in the substrate is accomplished by the Low-Level Scheduler, which sequences the deactivation (or activation) of appropriate stabilizer measurements. The same scheduler can also generate a single-qubit logical gate, which constitutes operations local to the TQC Tile. For a two-qubit logical gate such as CNOT, the "braiding" procedure can be simulated by the High-Level Scheduler, which controls the movement of logical qubit data (holes in case of TQC) across the Tiles. Finally, the error decoding and recovery procedure can be implemented by simply integrating the well-known minimum weight matching algorithm [Edmonds 1965] with the existing Error Analyzer of the tool to calculate overall logical failure probability.

6.2. Different Quantum Algorithm Circuits

Our benchmark circuits compose crucial components of not only the Shor algorithm but also a wide range of other important applications such as quantum chemistry, solving system of linear equations, and triangle finding algorithm [Jordan 2011]. The oracle part in a general quantum algorithm can be separately compiled using off-the-shelf quantum compilers (e.g., JavadiAbhari et al. [2014]), which offer optimized reversible circuit synthesis. The resulting circuit can be merged with the quantum part of the algorithm by using the circuit integration feature of the Quantum Application Circuit Generator in ADPA. This way, a general quantum algorithm can be simulated for the investigation of resource-performance tradeoffs.

6.3. Alternative Quantum Device Technologies and Architectures

The Tile construct used to abstract trapped-ion hardware is powerful enough to adequately model QC architecture based on other hardware technologies such as superconductors, quantum dots, and neutral atoms [Ladd et al. 2010]. The technology-specific storage and manipulation of qubits are described by a new set of DPs. The hardware constraints characterizing physical layout can be captured by the weighted directed (or undirected) graph, wherein standard graph traversal algorithms can be leveraged to model the movement of quantum information required for the physical gates between nonlocal qubits. The convenient implementation of these algorithms in the current tool software will enable TDPA to produce Tiles that can schedule fault-tolerant operation for various device technologies. Once the performance of technology-specific Tiles is parameterized using new DPs, ADPA specifies the mechanism of which application-level qubit operands contained in different Tiles communicate to allow nonlocal logical gates. As such, the Tiles are connected by the communication channels compatible with given hardware technology. For example, ballistic shuttling in trapped-ion, successive swapping in superconductors and optical interconnection in photons can be modeled either as a *Tile Port,* which interfaces one Tile to another, or as a dedicated Communication Tile, presented in Section 3.2. Once the communication component is appropriately modeled by ADPA, application-level QC architecture can be defined for the chosen quantum physical hardware.

7. COMPARISON WITH THE RELATED ARCHITECTURE WORK

A generation of quantum architectures for large monolithic ion traps had been analyzed using area as a metric for resource utilization [Metodi et al. 2005; Thaker et al. 2006; Whitney et al. 2009]. Unfortunately, the sizable trap chip envisioned in these studies is difficult to fabricate due to limitations of fabrication technology [Guise et al. 2014].

These constraints force us to adopt a modular quantum architecture and a resource metric, both of which are largely decoupled from the trap size. In this article, we analyzed a multicore architecture (MUSIQC), which invests in communication qubits to generate EPR pairs in order to connect variable-size ion traps (Segments). We used qubits per trap (qubits/Segment) and total qubits in the computer as the resource metrics in our simulations. We find that the MUSIQC architecture directly translates scalability cost into the qubits required for computation and communication regardless of physical size of the trap. By analyzing this modular architecture, we present a practical framework for designing and evaluating the future generation of quantum architectures.

8. CONCLUSION

We presented a complete performance simulation toolset capable of designing a resource-efficient, scalable QC. Our tool is capable of analyzing the performance metrics of a flexible, reconfigurable computer model and deepens our insights into the quantum architecture design by providing a comprehensive breakdown of performance metrics and visualization of resource utilization. Using this tool, we were able to quantify, for the first time, the resource-performance scalability of a proposed architecture, featuring unique properties such as (1) cross-layer optimization, where qubit resources providing L2-level functions are shared throughout the computer; (2) resource-constrained hardware performance, where optimized architectural design for resource allocation is considered as a function of the problem size; and (3) complete visualization of the resource utilization that provides a means to validate the optimality of the performance, (4) over a hardware architecture that provides global connectivity among all the qubits in the system.

Due to the macromodeling approach used in our tool, we achieve highly efficient runtime for the performance simulation, which allows us to carry out a comprehensive search for an optimized system design under given resource constraints, over a range of architecture configurations and benchmark circuits. Our benchmarks included crucial building blocks of Shor's algorithm, including the approximate quantum Fourier transform and two types of quantum adders. We found that the optimized designs vary across the benchmark applications depending on the types of gates used, the depth and parallelism of circuit structure, and resource budget. By comparing their performance across these benchmark circuits, we present a concrete quantum computer design capable of executing the 2,048-bit Shor algorithm in less than 5 months. A free copy of our toolset can be obtained by contacting the first author and is currently available at http://www.cs.duke.edu/∼ahsan/SQrIpT.

REFERENCES

D. Aharonov and M. Ben-Or. 1997. Fault-tolerant quantum computation with constant error fault-tolerant quantum computation with constant error fault-tolerant quantum computation with constant error. In *Proceedings of the 29th Annual Symposium on Theory of Computing*, 176–188.

M. Ahsan, B.-S. Choi, and J. Kim. 2013. Performance simulator based on hardware resources constraints for ion trap quantum computer. In *IEEE 31st International Conference on Computer Design (ICCD'13)*, 411–418.

M. Ahsan and J. Kim. 2015. Optimization of quantum computer architecture using a resource-performance simulator. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*. EDA Consortium, San Jose, CA, 1108–1113.

P. Aliferis, D. Gottesman, and J. Preskill. 2005. Quantum accuracy threshold for concatenated distance-3 codes. *arXiv:quant-ph/0504218* (2005).

S. Balensiefer, L. Kreger-Stickles, and M. Oskin. 2005. QUALE: Quantum architecture layout evaluator. In *Proceedings of the SPIE*, Vol. 5815, 103–114. DOI:http://dx.doi.org/10.1117/12.604073

D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill. 1996. Efficient networks for quantum factoring. *Phys. Rev. A* 54 (1996), 1034–1063.

S. Crain, E. Mount, S.-Y. Baek, and J. Kim. 2014. Individual addressing of trapped $^{171}Yb^{+}$ ion qubits using a microelectromechanical systems-based beam steering system. *Appl. Phys. Lett.* 105 (2014), 181115.

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

T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore. 2006. A logarithmic-depth quantum carry-lookahead adder. *Quantum Inf. Comput.* 6 (2006), 351–369.

L.-M. Duan, B. B. Blinov, D. L. Moehring, and C. Monroe. 2004. Scaling trapped ions for quantum computation with probabilistic ion-photon mapping. *Quant. Inf. Comp.* 4 (2004), 165–173.

J. Edmonds. 1965. Paths, trees, and flowers. *Can. J. Math.* 17, 3 (1965), 449–467.

A. G. Fowler and L. C. L. Hollenberg. 2004. Scalability of shor's algorithm with a limited set of rotation gates. *Phys. Rev. A* 70, 3 (2004), 032329.

A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A* 86, 3 (2012), 032324. DOI:http://dx.doi.org/10.1103/PhysRevA.86.032324.

A. Galiautdinov, A. N. Korotkov, and J. M. Martinis. 2012. Resonator-zero-qubit architecture for superconducting qubits. *Phys. Rev. A* 85 (2012), 042321.

B. Giles and P. Selinger. 2013. Exact synthesis of multiqubit Clifford+T circuits. *Phys. Rev. A* 87 (2013), 032332.

D. Gottesman and I. L. Chuang. 1999. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature* 402 (October 1999), 390–393.

N. D. Guise, S. D. Fallek, K. E. Stevens, K. R. Brown, C. Volin, A. W. Harter, J. M. Amini, R. E. Higashi, S. T. Lu, H. M. Chanhvongsak, et al. 2014. Ball-grid array architecture for microfabricated ion traps. *arXiv preprint arXiv:1412.5576* (2014).

P. Papadopoulos J. Eisert, K. Jacobs, and M. B. Plenio. 2000. Optimal local implementation of non-local quantum gates. *Phys. Rev. A* 62 (2000), 052317.

A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi. 2014. ScaffCC: A framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 1.

L. Jiang, J. M. Taylor, K. Nemoto, W. J. Munro, R. Van Meter, and M. D. Lukin. 2009. Quantum repeater with encoding. *Phys. Rev. A* 79, 3 (Mar 2009), 032325. DOI:http://dx.doi.org/10.1103/PhysRevA.79.032325

S. Jordan. 2011. Quantum algorithm zoo. Retrieved June 27, 2013 from http://math.nist.gov/quantum/zoo/.

M. Juvan and B. Mohar. 1992. Optimal linear labelings and eigenvalues of graphs. *Discrete Appl. Math.* 36 (1992), 153–168.

J. Kim and C. Kim. 2009. Integrated optical approach to trapped ion quantum computation. *Quantum Inf. Comput.* 9 (2009), 181–202.

J. Kim, C. J. Nuzman, B. Kumar, D. F. Lieuwen, J. S. Kraus, A. Weiss, C. P. Lichtenwalner, A. R. Papazian, R. E. Frahm, N. R. Basavanhally, D. A. Ramsey, V. A. Aksyuk, F. Pardo, M. E. Simon, V. Lifton, H. B. Chan, M. Haueis, A. Gasparyan, H. R. Shea, S. Arney, C. A. Bolle, P. R. Kolodner, R. Ryf, D. T. Neilson, and J. V. Gates. 2003. 1100 X 1100 port MEMS-based optical crossconnect with 4-dB maximum loss. *IEEE Photon. Technol. Lett.* 15 (2003), 1537–1539.

A. Y. Kitaev. 2003. Fault-tolerant quantum computation by anyons. *Ann. Phys.* 303, 1 (2003), 2–30.

T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, et al. 2010. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology (CRYPTO'10)*. Springer, 333–350.

V. Kliuchnikov, D. Maslov, and M. Mosca. 2013. Asymptotically optimal approximation of single qubit unitaries by clifford and T circuits using a constant number of ancillary qubits. *Phys. Rev. Lett.* 110 (2013), 190502.

T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. OBrien. 2010. Quantum computers. *Nature* 464, 7285 (2010), 45–53.

T. S. Metodi, D. D. Thaker, A. W. Cross, F. T. Chong, and I. L. Chuang. 2005. A quantum logic array microarchitecture: Scalable quantum data movement and computation. In *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'38)*, 12–23.

C. Monroe and J. Kim. 2013. Scaling the ion trap quantum processor. *Science* 339 (2013), 1164.

C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. 2014. Large scale modular quantum computer architecture with atomic memory and photonic interconnects. *Phys. Rev. A* 89 (2014), 022317.

M. A. Nielsen and I. L. Chuang. 2000. *Quantum Computation and Quantum Information*. Cambridge University Press.

S. Olmschenk, K. C. Younge, D. L. Moehring, D. N. Matsukevich, P. Maunz, and C. Monroe. 2007. Manipulation and detection of a trapped Yb^{+} hyperfine qubit. *Phys. Rev. A* 76 (2007), 052314.

P. W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* 26, 5 (1997), 1484–1509.

A. M. Steane. 1996. Error correcting codes in quantum theory. *Phys. Rev. Lett.* 77 (1996), 793–797.

K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov. 2006. A layered software architecture for quantum computing design tools. *Computer* 39 (2006), 74–83. DOI:http://dx.doi.org/10.1109/MC.2006.4

D. D. Thaker, T. S. Metodi, A. W. Cross, I. L. Chuang, and F. T. Chong. 2006. Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. In *ACM SIGARCH Computer Architecture News*, Vol. 34. IEEE Computer Society, 378–390.

R. Van Meter and C. Horsman. 2013. A blueprint for building a quantum computer. *Commun. ACM* 56, 10 (2013), 84–93.

R. Van Meter and K. M. Itoh. 2005. Fast quantum modular exponentiation. *Phys. Rev. A* 71, 5 (May 2005), 052320.

R. Van Meter, W. J. Munro, K. Nemoto, and K. M. Itoh. 2008. Arithmetic on a distributed-memory quantum multicomputer. *J. Emerg. Technol. Comput. Syst.* 3, Article 2 (2008), 23 pages.

V. Vedral, A. Barenco, and A. Ekert. 1996. Quantum networks for elementary arithmetic operations. *Phys. Rev. A* 54 (1996), 147–153.

M. Whitney, N. Isailovic, Y. Patel, and J. Kubiatowicz. 2007. Automated generation of layout and control for quantum circuits. In *Proceedings of the 4th International Conference on Computing Frontiers*, 83–94.

M. G. Whitney, N. Isailovic, Y. Patel, and J. Kubiatowicz. 2009. A fault tolerant, area efficient architecture for Shor's factoring algorithm. *ACM SIGARCH Comput. Arch. News* 37, 3 (2009), 383–394.

B. Zeng, A. Cross, and I. L. Chuang. 2011. Transversality versus universality for additive quantum codes. *IEEE Trans. Inf. Theory* 57, 9 (2011), 6272–6284.

X. Zhou, D. W. Leung, and I. L. Chuang. 2000. Methodology for quantum logic gate construction. *Phys. Rev. A* 62 (2000), 052316.

Received May 2015; revised July 2015; accepted September 2015