Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights

Prakash Murali*
Princeton University

Norbert Matthias Linke
University of Maryland

Margaret Martonosi
Princeton University

Ali Javadi Abhari
IBM T. J. Watson Research Center

Nhung Hong Nguyen
University of Maryland

Cinthia Huerta Alderete
University of Maryland

ABSTRACT

In recent years, Quantum Computing (QC) has progressed to the point where small working prototypes are available for use. Termed Noisy Intermediate-Scale Quantum (NISQ) computers, these prototypes are too small for large benchmarks or even for Quantum Error Correction (QEC), but they do have sufficient resources to run small benchmarks, particularly if compiled with optimizations to make use of scarce qubits and limited operation counts and coherence times. QC has not yet, however, settled on a particular preferred device implementation technology, and indeed different NISQ prototypes implement qubits with very different physical approaches and therefore widely-varying device and machine characteristics.

Our work performs a full-stack, benchmark-driven hardware-software analysis of QC systems. We evaluate QC architectural possibilities, software-visible gates, and software optimizations to tackle fundamental design questions about gate set choices, communication topology, the factors affecting benchmark performance and compiler optimizations. In order to answer key cross-technology and cross-platform design questions, our work has built the first top-to-bottom toolflow to target different qubit device technologies, including superconducting and trapped ion qubits which are the current QC front-runners. We use our toolflow, TriQ, to conduct *real-system* measurements on seven running QC prototypes from three different groups, IBM, Rigetti, and University of Maryland. Overall, we demonstrate that leveraging microarchitecture details in the compiler improves program success rate up to 28x on IBM (geomean 3x), 2.3x on Rigetti (geomean 1.45x), and 1.47x on UMDTI (geomean 1.17x), compared to vendor toolflows. In addition, from these real-system experiences at QC's hardware-software interface, we make observations and recommendations about native and software-visible gates for different QC technologies, as well as communication topologies, and the value of noise-aware compilation even on lower-noise platforms. This is the largest cross-platform real-system QC study performed thus far; its results have the potential to inform both QC device and compiler design going forward.

*Prakash Murali is the corresponding author and can be reached at pmurali@cs.princeton.edu.

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6669-4/19/06…$15.00
https://doi.org/10.1145/3307650.3322273

CCS CONCEPTS

• **Computer systems organization** → **Quantum computing**; • **Software and its engineering** → *Compilers*.

ACM Reference Format:
Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. 2019. Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights. In *ISCA '19: 46th International Symposium on Computer Architecture, June 22–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3307650.3322273

1 INTRODUCTION

Quantum computing (QC) is emerging as a promising paradigm for solving classically intractable computational problems in areas such as machine learning [4, 39], cryptography [60], chemistry [32, 50] and others. QC devices represent information using *qubits* (quantum bits) and perform operations based on quantum mechanical principles such as superposition and entanglement to achieve speedups over classical algorithms.

In recent years, QC implementations have advanced considerably. QC prototypes with up to 16 qubits are available for broad public use [27] and larger 49-72 qubit systems are either announced or in use [17, 25, 30]. Much like the early days of classical (i.e. non-quantum) computing, however, QCs have not yet converged on a specific candidate device technology. Front-runner technologies today include superconducting transmon qubits [40, 57] and trapped ion qubits [8, 12, 23], with other candidate technologies also of considerable interest [33, 34, 51].

As shown in Figure 1, the current candidate QC device technologies differ widely in key physical attributes. First and foremost, the different methods of forming qubits are sufficiently distinct that even the fundamental gate operations performed on them differ widely. (In contrast, consider that classical computers built from vacuum tubes, relay circuits, or transistors all hinge on a switch abstraction that maps similarly to Boolean logic gates.) In addition to gate differences, there are also differences in how inter-qubit interactions are accomplished, and these lead to widely disparate communication approaches and connectivity topologies. Finally, because of the differences in physical implementation, there are also considerable variations in the noise and error characteristics. While all current QCs are susceptible to operation, communication, and measurement errors, the nature, magnitude, and spatiotemporal variance of these errors differs greatly from technology to technology.

This paper is the first to perform a cross-technology hardware software assessment of QC design. We assess how differences in

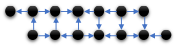
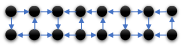
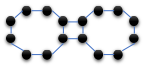
Machine	Qubits	2Q Gates	Coherence Time (us)	1Q Error (%)	2Q Error (%)	RO Error (%)	Qubit Topology
IBM Q5 Tenerife	5	6	40	0.2	4.76	6.21	
IBM Q14 Melbourne	14	18	30	1.19	7.95	9.09	
IBM Q16 Rüschlikon	16	22	40	0.22	7.14	4.15	
Rigetti Agave	4	3	15	3.68	10.8	16.37	
Rigetti Aspen1	16	18	20	3.43	8.92	5.56	
Rigetti Aspen3	16	18	20	3.79	5.37	6.65	
UMD Trapped Ion (UMDTI)	5	10	1.5×10^6	0.2	1.00	0.6	

Figure 1: Characteristics of the devices used in our study. Each device has different qubit and gate count (higher is better), coherence time (higher is better), error rates (lower is better) and topology (dense connectivity is better). Rigetti Agave has 8 qubits in a ring topology, but only 4 qubits were available during our study.

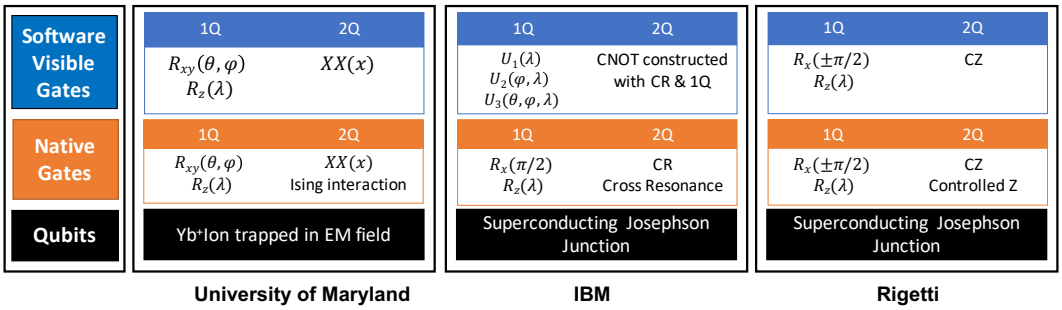


Figure 2: Qubit type, native gates and software-visible gates in the systems used in our study. Each technology lends itself to a set of native gates. Vendors may expose these gates in the software-visible interface or construct composite gates from multiple native gates.

fundamental gates, their software visibility, communication topologies, and noise characteristics all influence software performance and reliability on a range of real-system QC prototypes. To do so, we developed a full-stack cross-platform toolflow, TriQ, which allows us to start from QC programs written in a C-like high-level language [1, 31, 58], progress through optimizations and mapping stages, and output device-specific code for *seven different QC platforms from three different vendors employing two different underlying device technologies*. We show how optimizing for specific device attributes can make significant improvements in performance and success rate. We further show how such device-specific characteristics can be input into an otherwise general/portable toolflow in order to allow multi-platform optimizations to be performed portably even with deeply device-specific optimizations.

The contributions of this work include the following. First, we perform the first multi-platform comparison of QCs from different vendors, built with different device technologies. While it is far too soon to "pick a winning technology" from such comparisons, the purpose of our work is to build insights in how architecture and compiler design choices can best support the different technologies. Conversely, we also believe that these early comparative studies offer important insights for device physicists as they roadmap further technology improvements [42, 47].

Second, to support our cross-platform studies, we have developed and will open-source the first multi-vendor QC compiler which compiles from high-level languages to multiple real-system QC prototypes, with device specific optimizations. TriQ takes as input a QC program written in a high-level language, a set of characteristics pertaining to implementation gates and communication topologies, and a summary of empirical device error data. We show that although the device technologies vary, a common set of configuration parameters can express the gate set and noise data in a way that allows device-specific optimization through a portable, general toolflow. We use this toolflow to compile 12 Scaffold benchmarks onto seven real QC prototypes from 3 vendors employing two distinct qubit implementation approaches (superconducting and trapped ion). For the UMD ion trap system, TriQ is the first high-level language compiler.

Third, our evaluation offers architectural insights for future QC systems. First and foremost, we demonstrate the importance of making software-visible the device technology's natural or fundamental gates. Shielding a technology's natural gates (e.g. the XX gate for trapped ion) by abstracting them into more familiar gates (e.g. CNOT) imposes runtime and error overheads that are too significant

for current NISQ systems to easily overcome. Exposing device-specific gates also allows for additional compile-time optimizations of single-qubit operations. Second, we find that even though trapped ion technologies have intrinsically lower error rates than many superconducting systems, there is still value in performing error-aware compilation for such systems.

Fourth, our compilation approach melding device specificity with a common core toolflow offers very good results; we achieve portability without a tradeoff cost in performance or reliability. In particular, TriQ outperforms vendor compilers. On IBM devices with 5-16 qubits, TriQ provides geomean 3x (up to 28x) improvement in program success rate over the IBM Qiskit compiler [26]. On Rigetti devices with 4-16 qubits, TriQ provides geomean 1.45x (up to 2.3x) over the Rigetti Quil compiler [62]. TriQ obtains improvements over IBM and Rigetti compilers because of noise-adaptivity, optimizing qubit communication and optimizing single qubit operations. On the UMD ion trap computer (UMDTI), TriQ uses noise-adaptiveness to improve program success rates by up to 1.47x, compared to a noise-unaware baseline. In particular, our paper is the first to demonstrate noise-adaptive compilation across 3 vendors and for trapped ion qubit technology. Finally, although compile time is not a primary design goal, TriQ scales well up to 72 qubits, the largest NISQ configuration announced thus far [17].

2 QC BACKGROUND

2.1 Principles of Quantum Computing

A qubit is the fundamental unit of information in a QC system. Qubits have two basis states $|0\rangle$ and $|1\rangle$, which are the analogues of the classical 0 and 1 states. However, quantum superposition allows a qubit to be in a complex linear combination, where its state is $\alpha|0\rangle + \beta|1\rangle$, for $\alpha, \beta \in \mathbb{C}$. An n-bit QC system can potentially exist in a superposition state of 2^n basis states simultaneously, unlike classical registers which can be in exactly one of the 2^n values at any given time. Qubits can be manipulated by modifying the complex numbers associated with the basis states, using operations which are commonly called gates. To obtain classical output, a qubit is measured, collapsing its state to either $|0\rangle$ or $|1\rangle$.

In a QC application, an algorithm is mapped to gates which execute on a set of qubits which are initialized appropriately. As the program executes, qubit amplitudes are manipulated and the state space is evolved towards the desired output. Finally, the qubits are measured or readout (RO) to generate classical output for the application.

2.2 Quantum Gates

Quantum gates are instructions which operate on one or more qubits. The functionality of a gate is achieved by applying some dynamic physical interaction (such as a microwave or laser pulse) to the qubit. Complex quantum operations can be composed as a sequence of operations from a small set of universal gates. Universal QC systems, such as the ones we experiment on here, provide a universal set of single-qubit (1Q) and two-qubit (2Q) operations.

The state of a single qubit can be represented by a complex vector on a unit sphere. All single-qubit operations can be viewed as rotation operations $R_x(\theta)$, $R_y(\phi)$ and $R_z(\lambda)$ along the X, Y or Z axes on this complex sphere. Rather than fully-general rotations, QC algorithms often use a set of composite 1Q operations, such as X/NOT gate ($R_x(\pi)$), Hadamard gate ($R_y(\pi/2)R_z(\pi)$) which generates superposition, Z gate ($R_z(\pi)$) and others [44].

2Q operations generate entanglement among qubits, resulting in non-classical correlated behaviour. Correlation from entanglement potentially allows a QC's state space to grow exponentially with qubit count. This is central to QC's power and is used by QC algorithms. A common example of a 2Q gate is Controlled NOT (CNOT) which acts on a control and target qubit pair. When the control qubit is in the state $|1\rangle$, the action of the gate is to flip the state of the target qubit[1]. Another example of a 2-qubit gate is a Controlled Z gate where the target qubit is rotated by π radians along the Z axis if the control qubit is $|1\rangle$.

2.3 NISQ Systems

Noisy Intermediate-Scale Quantum (NISQ) are near-term systems with less than 500-1000 qubits [52]. This scale is typically too small to implement quantum error correction, but if used efficiently these machines may have promising applications in various domains [52] and can pave the way towards practical QC.

NISQ systems are built using a variety of qubit technologies, including superconducting qubits [57], trapped ions [8], spin qubits [24], among others. To reliably process information, these qubits should be "coherent" for sufficiently long, i.e., they should maintain the quantum state for a length of time. The qubits should also support sufficiently precise operations to allow the state to be manipulated correctly during the coherence window. To obtain useful output, qubits should also support accurate readout or measurement operations [13]. Because of the error rates in current NISQ systems, some fraction of the runs result in the wrong answer being calculated. As a result, it is common to run a QC program many times with a figure of merit being the *success rate*, the fraction of runs that resulted in a correct answer.

3 DEVICE, ARCHITECTURE TRADEOFFS

3.1 Native Gate Choices

Figure 2 shows the qubit technologies and gates (or operations) used in the systems at IBM [29], Rigetti [6] and UMD [12]. IBM and Rigetti use superconducting qubits based on Josephson junctions, while UMD uses ions trapped in an electromagnetic field.

In some ways, these different qubit implementation options are analogous to how classical computers might be implemented using vacuum tubes or CMOS transistors. On the other hand, while many classical devices can all be abstracted as on-off switches, qubit technologies are still more distinct. Each QC vendor implements a set of *native* operations that are feasible on their platform. Like classical NAND and NOR gates, all QC operations must be composable from a universal set of native gates. Typically, the vendor provides at least one 1Q and one 2Q operation. Figure 2 shows that differences in underlying device technologies lend themselves to quite different native operations.

In IBM Q systems, the fundamental 2Q interaction is a cross resonance gate where one qubit is driven at the resonant frequency

[1] A CNOT gate with control C and target T is denoted as `CNOT C, T`.

of the other [59]. On Rigetti systems, 2Q interactions are Controlled Z gates where a Z gate is applied on the target qubit when the control is $|1\rangle$. In the UMD system, entanglement is generated using Ising interaction (XX) which uses ion motion to couple the qubits.

At a higher level, each vendor also decides on what sort of programmable machine interface to support. This software-visible set of operations can include either native gates themselves or composite gates which use multiple fundamental gates.

Therefore, one key design decision is: *What sorts of operations should the machine expose to software? How do these choices of software-visible operations affect the performance and reliability of the applications run on them? And is there value in having these operations be unified across different implementations or is it more important to have them be well-tailored to underlying device characteristics?*

3.2 Communication Characteristics

As shown in Figure 1's topology column, different QC implementations come with different qubit connectivity attributes. For example, the IBM and Rigetti systems have sparse near-neighbor connectivity where 2Q operations can only be performed between *adjacent* qubits. To perform 2Q gates between non-adjacent qubits, these machines perform one or more SWAP operations to move relevant qubits until the control and target qubits for a 2Q operation are in adjacent locations. Each SWAP operation between two adjacent qubits (i.e., each hop) requires 3 2Q gates[2]. These SWAP operations are time-consuming, but even worse, each one is error-prone; a program with too many SWAPs is highly unlikely to get the right answer. UMDTI natively supports full connectivity between any pair of ions, which means that swap operations are not required on this architecture.

The distinction in communication topologies is not coincidental. Indeed, full communication connectivity is easier to achieve in ion trap technologies at least at small qubit counts. 2Q gates in superconducting technologies require a physical resonator immediately between the 2 qubits to perform the operation, whereas in ion trap approaches, the interacting ions need not be physically adjacent and do not need physical resonators between them when the laser-pulsed operation occurs.

Our work is the first to perform comprehensive real-systems measurements between high-level language applications running on superconducting and ion trap implementations. We ask and answer questions regarding how different communication topologies affect application success rate and runtime. We also explore the best methods for a common toolflow to target such widely-divergent technologies and topologies.

3.3 Noise and Coherence Characteristics

Finally, QC systems today are characterized by high and often fluctuating error rates (or "noise") in their gate operations and qubit readouts [35, 46]. NISQ systems have imprecise operations with high 1Q, 2Q and readout (RO) errors. 1Q error rates are smaller than 2Q and RO, but present. 2Q operations are fundamentally harder to perfect and are often composed to multiple 1Q operations. The superconducting qubits used in IBM and Rigetti have defects arising from their lithographic manufacturing processes and their variance is only beginning to be understood [35]. The IBM qubits are calibrated twice a day and the experimental measurements of 1Q, 2Q and RO error rates are posted online [28], as Figure 3 shows for *IBMQ14*. Across IBM and Rigetti systems, 2Q and RO error rates vary up to 9x across qubits and calibration cycles. Ion traps have less temporal variance, and UMDTI has error rates lower than the machines based on superconducting qubits. Nonetheless, despite lower error rates, there is still spatial and temporal variance of 1-3% in 2Q error rates across the qubits in the system [12]. These differences arise from the difficulty in qubit control using lasers and their sensitivity to motional mode drifts from temperature fluctuations.

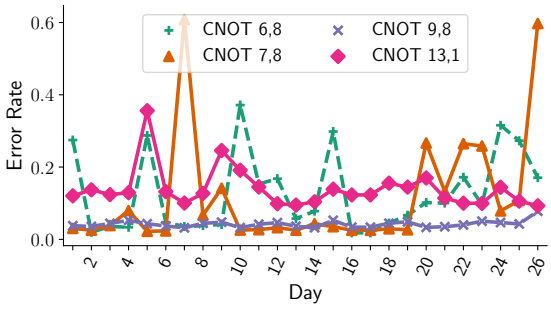


Figure 3: Daily variation of error rates of 4 hardware supported 2Q (CNOT) gates in IBMQ14. 2Q error rate in this system averages 7.95%, but varies 9x across qubits and days.

Another figure of merit in QC systems is the *coherence time*, which is a fundamental limit on the time up to which information can be reliably manipulated on a qubit. A loose analogy might be made to the DRAM refresh interval in classical systems. Figure 1 shows that IBM and Rigetti systems have short coherence time compared to UMDTI. Nonetheless, as QC systems are able to run longer and larger programs successfully, the limits imposed by finite coherence time will play a role in how algorithms are developed and how programs are compiled.

Therefore, we ask: *Can a single toolflow be built that maps well across widely varying implementations, and that optimizes algorithm mappings to the underlying implementation's error characteristics and coherence intervals? Moreover, are there a small, common set of figures of merit that can guide optimization across such divergent implementations?*

3.4 Our Work

Based on the prototype device technologies and their architectural implications, this work first develops a common toolflow, TriQ, that maps well to widely-different machine implementations. Using TriQ, we ask and answer questions about gate sets, communication trade-offs, and other architectural choices on these machines. We quantify our answers by presenting *real-system* runs on seven different QC implementations based on two distinct implementation technologies.

4 DESIGN AND OVERVIEW OF TRIQ

Many traditional compilers are structured as a language-dependent front-end, a hardware- or ISA-dependent back-end, and a set of neutral analysis passes in the middle (intermediate representation or

[2]For two qubits A and B, `SWAP(A,B) := {CNOT A,B; CNOT B,A; CNOT A,B;}`.

Table 1: Compilers and optimization levels considered.

Compiler	Description
TriQ-N	TriQ. No optimization. Default qubit mapping
TriQ-1QOpt	TriQ, 1Q gate optimization. Default qubit mapping
TriQ-1QOptC	TriQ. 1Q opt. Communication-optimized mapping,
TriQ-1QOptCN	TriQ. 1Q opt. Comm- and Noise-optimized mapping.
Qiskit	IBM Qiskit compiler version 0.6.0 [26]
Quil	Rigetti Quil compiler version 1.9 [55]

IR). The abstractive power of such approaches shields the higher-level optimizations from many hardware-specific details and vice versa. In contrast, our work here requires that many more hardware and software implementation attributes are available to all or nearly all of the full-stack of the compiler.

4.1 Overview

As shown in Figure 4, the TriQ toolflow contains core functionality to perform noise-aware qubit mapping, 1Q optimizations, and communication optimizations. For all this functionality, the device-specific attributes are provided to the core functionality as compiler inputs. This includes the machine's qubit count and connectivity, its native gate set, and a summary of its noise characteristics. In essence, it operates like a multi-target compiler in which characteristics such as the ISA and operation latencies are provided as compile-time inputs. In this way, our compiler can target very different devices simply by changing input characteristics. Section 6 shows that this flexibility comes with no performance trade-off; TriQ outperforms the native vendor compilers in both performance and success rate.

Our toolflow accepts program inputs written in Scaffold, a C-like language that represents QC programs in a device- and vendor-independent manner. The ScaffCC compiler [31, 58] parses from Scaffold into an LLVM IR [36] consisting of 1Q and 2Q gates. Figure 5 shows the IR for the Bernstein-Vazirani algorithm [3], a common NISQ benchmark program. ScaffCC automatically decomposes higher-level QC operations such as Toffoli gates into native 1Q and 2Q representations. Since QC programs are usually compiled for a fixed input, ScaffCC also takes the application input data and resolves all classical control dependencies. The output IR graph includes the qubits required for each operation and data dependencies between operations.

From both the application inputs (program and input data) and the inputs about the device characteristics (resource counts, noise statistics, etc.) the core compiler passes analyze and optimize mappings before generating device-specific executable code to be run on one of seven real systems. The sequence of compiler analyses is discussed below. In our experiments, we vary which optimizations are applied. Table 1 names and summarizes the optimization approaches.

4.2 Reliability Matrix Computation

The qubit mapping and communication orchestration phases must determine good spatial placements for qubits and good routing paths for 2Q gates. As Figure 1 shows, 2Q and RO operations dominate error rates and are important to optimize for [46]. The gate errors on both superconducting and trapped ion prevent long gate sequences and are more limiting than coherence times. For these reasons, a central aspect of qubit mapping and gate orchestration decisions is optimizing for the reliability of 2Q and RO operations. The challenge is doing so in a way that ports well across very different implementations.

To inform qubit mapping and communication orchestation, TriQ uses the provided qubit topology and noise data to construct a matrix which summarizes the "end-to-end" reliability of 2Q operations between any pair of qubits, including any communication routing required to co-locate the qubits. Figure 6 shows an example. The i, j^{th} entry in this matrix estimates the reliability of performing a 2Q operation from qubit i to qubit j, including the cost of communication. By distilling the important factors of a machine's topology and 2Q errors into single matrix representation, this approach is applicable both to fully-connected machines like UMDTI as well as to machines with more limited topologies.

To fill in the reliability matrix, TriQ considers the topology of the machine, where nodes are hardware qubits and edges are hardware-supported, direct 2Q gates between them. Each edge is labelled with the reliability of the corresponding 2Q gate. To estimate the reliability score for non-local operations, TriQ performs an all-pairs swap cost computation using the Floyd-Warshall algorithm [9]. For each pair of qubits c and t, it determines the most reliable neighbor t' of t. t' is the neighbor which maximizes the product of reliability of the swap path from c to t' and the 2Q gate from t' to t (for IBM machines, we also include the error rates of any extra 1Q gates necessary for orienting the gates in the hardware-supported directions). In Figure 6, the best neighbor for a 2Q gate from 1 to 3 is 4.

For noise-unaware compilation such as TriQ-1QOptC, the 2Q gate reliability for all edges is set as the average error rate in the system. Then the reliability matrix computation in effect determines shortest paths which minimizes hop count. The noise-aware TriQ-1QOptCN approach uses the input noise/error data to set the gate reliability. Here, the shortest path computation choose the most reliable path, minimizing prevalence (severity and count) of erroneous operations.For example, if the two program qubits p_0 and p_3 from Figure 5 are mapped to qubits 1 and 3 in Figure 6, the reliability of a 2Q operation between them is 0.58. We also record the readout reliabilities for the hardware qubits in a vector, where the i^{th} entry denotes the accuracy of measuring the state of qubit i. This matrix becomes an input for the subsequent passes that follow.

4.3 Qubit Mapping

To map program qubits to hardware qubits, TriQ uses a constrained optimization method similar to [46]. The optimization creates variables for each program qubit denoting which hardware qubit it may be mapped to. The reliability matrices for 2Q and readout operations previously discussed ascribe possible operation costs for each program operation. The optimization goal is to maximize success rate, so our objective function is a function of the reliability for the program dependence graph as mapped.

Maximizing this objective implies that communicating qubits should be mapped close together, and hardware gates and readout units which have poor reliability scores should be avoided. The optimization problem can be solved by expressing the variables, constraints and objective in a Satisfiability Modulo Theory solver

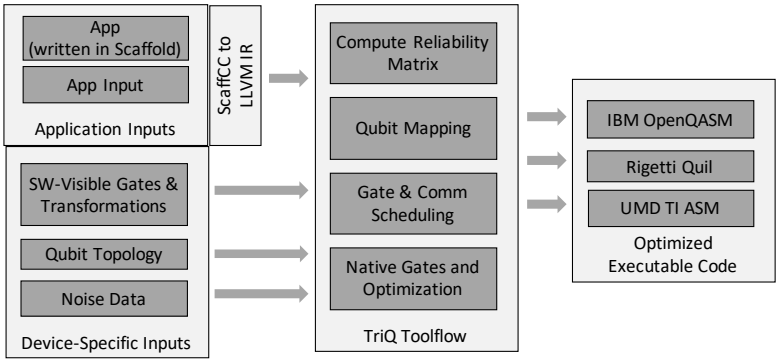


Figure 4: Overview of the TriQ toolflow. Input to TriQ consists of high-level Scaffold programs and their inputs, as well as device-specific QC system properties. Output is optimized code in one of three vendor-specific executable formats.

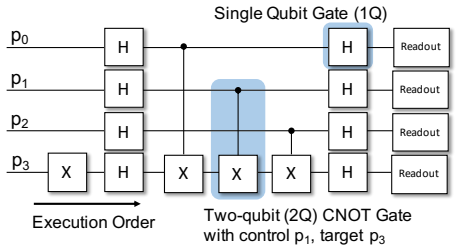
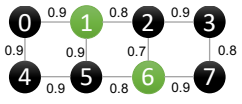


Figure 5: IR-level circuit diagram for Bernstein-Vazirani with 4 qubits (BV4). The program IR depicts program qubits and has 1Q, 2Q and RO operations.



(a) Example 8-qubit device

	0	1	2	3	4	5	6	7
0		0.9	0.58	0.33	0.9	0.65	0.42	0.24
1	0.9	-	0.8	0.46	0.65	0.9	0.58	0.33
2	0.46	0.8	-	0.9	0.33	0.46	0.7	0.58
3	0.33	0.58	0.9	-	0.24	0.33	0.51	0.8
4	0.9	0.65	0.42	0.24	-	0.9	0.58	0.33
5	0.65	0.9	0.58	0.33	0.9	-	0.8	0.46
6	0.33	0.46	0.7	0.58	0.46	0.8	-	0.9
7	0.24	0.33	0.51	0.8	0.33	0.58	0.9	-

(b) 2Q reliability matrix

Figure 6: Example 8-qubit device with 2Q gate reliabilities and corresponding 2Q reliability matrix. For a 2Q gate involving qubits 1 and 6, the highest reliability path involves first swapping 1 and 5 (reliability = 0.9^3) and then performing the 2Q gate. Thus, the (1,6) entry of the reliability matrix is $0.9^3 * 0.8$ or 0.58.

[5, 11]. Such approaches have been used to find optimal hardware mappings for classical programs [49] and for the IBMQ16 device by [46].

Our mapper is more general and more scalable than prior QC work. First, [46] uses a problem formulation which assumes that the device is a 2D grid. Since TriQ is multi-platform, it must target devices with arbitrary topology, which our 2D reliability matrix supports well. In addition, our objective function is chosen to be more scalable than prior work, while still targeting reliability. In particular, our objective function *maximizes the minimum reliability of any operation in the mapped graph*. In contrast, prior QC work has used a reliability product across the whole graph; these require more exhaustive search techniques that slow down the SMT solver runs. (We use the product-style reliability estimate in populating the reliability matrix, but use the faster maximize-the-minimum approach for the SMT solver that maps based on the matrix.) Our implementation allows the SMT solver to prune bad solutions early in the search tree: if it maps two qubits and a gate has lower reliability than the current optimum, the mapping can be discarded. In contrast, for the product-based method in [46], the solver must place all qubits and evaluate the product over some or all operations before it can discard a mapping as sub-optimal. As a result, TriQ is more scalable, and offers acceptable success rates on par with prior work. While different objective functions can be employed as desired, the current TriQ approach seems particularly useful for problems where [46] fails to compute a solution. Using the routing method in the next section, TriQ also avoids unnecessary swaps incurred by [46] and improves the likelihood of program success up to 50% on IBMQ16.

4.4 Gate and Communication Scheduling

We schedule gates in a topologically-sorted order from the IR's gate dependencies. This ensures that before a gate is executed, all its dependent gates are completed. For example, in Figure 5, the X gate is executed first on p_3, followed by H gates which can execute in parallel on all qubits, followed by the other gates. Once scheduled, where selected qubits are non-adjacent, the compiler determines routing paths for 2Q gates. (This step is not required for fully-connected topologies such as UMDTI.) The compiler seeks to use the most reliable path for the control and target qubit pair based on Section 4.2's reliability matrix. The compiler inserts SWAP gates to move the qubits along the best path, culminating in the desired local 2Q operation. TriQ updates the qubit mapping to reflect the swaps and processes the next 2Q gate using the new mapping.

4.5 Gate Implementation, Optimization and Code Generation

To generate executable code, TriQ must translate from higher-level IR gates into the software-visible gates of the device. Using the legal transformations provided as input to the toolflow, the compiler replaces SWAPs, CNOTs, and other operations in terms of the native or software-visible gate set. For example, on superconducting machines, to implement a `SWAP A,B` gate, it is decomposed into 3 2Q gates as `CNOT A,B; CNOT B,A; CNOT A,B`.

For IBM, CNOT is a machine-supported, software-visible operation and needs no further transformation, although access to lower-level native gates might allow for further optimizations. For Rigetti, CNOT is not software visible and instead, we decompose a `CNOT`

Name	Qubits	Gates	2Q Gates
BV4	4	12	3
BV6	6	16	3
BV8	8	18	3
HS2	2	16	2
HS4	4	28	4
HS6	6	42	6
Toffoli	3	18	6
Fredkin	3	19	8
Or	3	17	6
Peres	3	16	5
QFT	2	13	5
Adder	4	23	10
Fredkin Sequence	1 to 8 invocations of Fredkin gate in loop for UMD TI		
Toffoli Sequence	1 to 7 invocations of Toffoli gate in loop for UMD TI		
Supremacy Circuits	Circuits up to 72 qubits, 2032 2Q Gates. (Scaling study)		

Figure 7: Summary of benchmarks used in our study. The supremacy circuits are used only for scaling studies.

`A,B` gate into a sequence of rotations and CZ operations: R_z`(`π`/2) B;` R_x`(`π`/2) B;` R_z`(`π`/2) B; CZ A,B;` R_z`(`π`/2) B;` R_x`(`π`/2) B;` R_z`(`π`/2) B;`. Similarly, for UMDTI, we decompose the CNOT as R_y`(`π`/2) A;` XX`(`π`/4) A,B;` R_y`(`$-\pi$`/2) A;` R_x`(`$-\pi$`/2) A;` R_z`(`$-\pi$`/2) A;`.

TriQ also applies device-specific transformations at this stage. For IBM devices with directed CNOTs, TriQ orients the IR CNOTs in correct direction using additional 1Q operations [44]. Next, TriQ optimizes sequences of 1Q gates. For each qubit, TriQ finds continuous sequences of 1Q operations. Since 1Q operations are rotations, each 1Q gate in the IR can be expressed using a unit rotation quaternion which is a canonical representation using a 4D complex number. TriQ composes rotation operations by multiplying the corresponding quaternions and creates a single arbitrary rotation. This rotation is expressed in terms of the input gate set. Furthermore, on all three vendors, Z-axis rotations are special operations that are implemented in classical hardware and are therefore error-free. TriQ expresses the multiplied quaternion as a series of two Z-axis rotations and one rotation along either X or Y axis [26, 68], thereby maximizing the number of error-free operations.

4.6 Executable Generation

Following the stages in order as described above, the toolflow generates code in a format that is executable on the targeted real-system platform. For IBM, this is OpenQASM [10], for Rigetti it is Quil [62], and for UMDTI, there is a special low-level assembly code syntax we target. We stress that the key analysis and optimization functionality is all in the core toolflow. The device-specific code generation backend is merely intended to output executable code in a syntax supported by that machine.

5 EXPERIMENTAL SETUP

Benchmarks: Figure 7 lists the QC programs in our study. Used in prior work on NISQ system evaluation and compilation [2, 38, 46, 63], these benchmarks include the Bernstein-Vazirani algorithm [3], Hidden Shift Algorithm [7], Quantum Fourier Transform [48], an adder and multi-qubit QC gates such as Toffoli and Fredkin [44]. We use these benchmarks because they constitute an essential part of many large QC applications [21, 60]. We created Scaffold programs for each benchmark and obtained LLVM IR using the ScaffCC compiler [31]. UMDTI's lower error rates support longer gate sequences, so we also created longer benchmarks by running the Toffoli and Fredkin benchmarks in a loop, similar to patterns in applications such as Grover's search [21].

To study scalability, we used supremacy circuits from Google's Cirq tool [18, 41]. These circuits are large programs designed to demonstrate the computational capability of QC systems. We experiment with supremacy circuits with up to 72 qubits and 2032 two-qubit gates (depth 128).

Compilers: Table 1 summarizes the compilers used in our study and their optimization levels. TriQ-N generates code in terms of a technology-independent gate set (CNOTs and common 1Q gates) and naively translates it to the software-visible gates on the device. TriQ-1QOpt generates code in the software-visible gate set of the machine and optimizes it using the properties of the gate set. TriQ-1QOptC uses a communication optimized mapping using a reliability matrix constructed from ideal average error rates. In TriQ-1QOptCN, the mapping is optimized using both noise and topology data by using a reliability matrix with gate errors from calibration data. For comparison, we use IBM's Qiskit compiler/mapper version 0.6.0 and Rigetti's Quil version 1.9 as baselines. These were the latest compiler versions while performing the experiments. UMDTI does not have a high-level compiler. Our compilation experiments use an Intel Skylake processor (2.4GHz, 128GB RAM) using gcc version 5.4 and Python3.5. Our toolflow uses the C++ APIs of Z3 SMT solver version 4.8.3 [11] for implementing the qubit mapping phase.

Real-System QC Experiments: As already discussed in Figure 1, we performed empirical experiments on seven operational QC machines from three organizations: IBM, Rigetti, and UMD. The IBM experiments were performed by running on the IBM Quantum Experience [27, 28]. The IBMQ APIs also provide access to the daily machine calibration data, including error rates such as 1Q, 2Q and RO errors. For the experiments on Rigetti and UMDTI, we worked with the operators of those machines to launch experimental runs, and we obtained similar calibration data directly from each vendor. Experiments on Rigetti were performed using the pyQuil APIs [55]. Compiled executables for UMDTI were prepared by us using gate calibration data from UMD. The system staff at UMD ran the compiled code on the device without modifications.

QC machine time and availability is scarce and variable across vendors. In some experiments, we were limited to testing selected compiler configurations. In addition, the Rigetti Agave, UMDTI and IBMQ5 machines did not have enough qubits at the time of our experiments to accommodate the BV6, BV8 and HS6 benchmarks.

Before each experiment, we recompile the benchmarks using the latest calibration data. For IBM and Rigetti machines, we use 8192 trials for each benchmark run. For UMDTI, we used 5000 trials per run because of less noise variability on this system. Success rate refers to the fraction of those repeated trials which give the correct answer. For example, success rate of 0.9 means that 90% of the trials produced the correct answer. For success rates, results from different graphs may vary because of experimental conditions. Results within a single graph are performed closely in time and so are comparable.

6 RESULTS

The TriQ toolflow allows us to study opportunities for multi-platform optimizations, as well as architectural implications related to different device and implementation choices. This section offers empirical real-system results on initial key questions.

6.1 Gate Specificity and Optimizations

Figure 8 shows the number of native 1Q operations using TriQ-N and TriQ-1QOpt. Because so-called "virtual Z gates" can be applied on all three vendors using runtime classical transformations, those rotations are error-free on all 3 vendors; we plot X and Y here. TriQ-N produces output code in terms of the software-visible gates, it does not perform any optimization. The native gate set plays a key role in the number of 1Q operations required by the unoptimized code. Namely, where swaps end up getting translated to sequences including native 1Q operations, then some benchmarks such as BV8 that require long swap paths on IBM will have a large number of supporting 1Q gates.

Figure 9 shows how 1Q optimizations result in improved success rates for IBMQ14 and UMDTI[3]. Even though 1Q operations are lower-error than 2Q operations, reducing the number of 1Q operations reduces faulty operations and increases success rate substantially.

1Q optimizations are clearly important, and the leverage gained from them depends partly on the native gate set provided by the vendor. Across the 3 machines, TriQ-1QOpt compared to TriQ-N offers geomean 1.4x improvement in operation count on IBMQ14, 1.4x on Rigetti and 1.6x on UMDTI. These improvements come from mapping more effectively onto the underlying native 1Q gates, as well as by exploiting the error-free Z-axis rotations. These 1Q optimizations pay off in success rate improvements: up to 1.26x improvement in success rate (geomean 1.09x on IBM, 1.03x on UMDTI) compared to TriQ-N.

The higher gains that UMDTI sees are directly related to the gate set provided. Namely, the underlying hardware supports a arbitrary $R_{x,y}\theta, \phi$ 1Q rotation gate, which it makes software-visible. Using this operation the compiler can simplify a long sequence of 1Q gates into a single rotation operation. This demonstrates the power of appropriate software-visible gates that can be implemented efficiently and at low error rates on the underlying hardware.

6.2 Importance of Qubit Connectivity

Figure 10a and 10b show the 2Q gate counts for IBMQ14 and Rigetti Agave, for TriQ-1QOpt and the communication-optimized TriQ-1QOptC. Where 1Q optimizations are local, 2Q optimizations are dominated by improvements in mapping the program onto communication paths that are either shorter or higher-reliability or both. TriQ-1QOptC is noise-unaware, but optimizes communication by tailoring the executable to the device topology. It reduces the number of 2Q operations by up to 22x in IBMQ14 (geomean 2.1x) and up to 3.5x reduction in Rigetti Agave (geomean 1.3x). (Since UMDTI is fully-connected, these topology optimizations are not applicable here.)

Comparing corresponding applications from Figure 10a and 10b, Rigetti Agave (line topology) requires more 2Q operations than IBMQ14 (grid topology) because the topology is more restricted. Toolflow functionality, i.e. TriQ-1QOptC, can overcome this to some degree by finding good placements, but not entirely.

Figure 10c shows how communication optimizations can be parlayed into higher success rate, particularly for IBMQ14. TriQ-1QOptC enables programs such as BV6, BV8 and Toffoli to succeed while they fail with TriQ-1QOpt. The figure also shows the importance of program-machine topology match. Programs with topology well-matched to the device topology have more chances of succeeding because they can be executed without swaps, reducing their 2Q gate count. For IBMQ14 (grid topology, see Figure 1), such programs include BV4 (4-qubit star) and HS2,4,6 (disjoint 2-qubit edges). On UMDTI, the fully connected topology (see Figure 1) accommodates all program 2Q patterns. Figure 9b shows that programs with varied topology have similar success rates on UMDTI.

6.3 Importance of Noise-Adaptivity

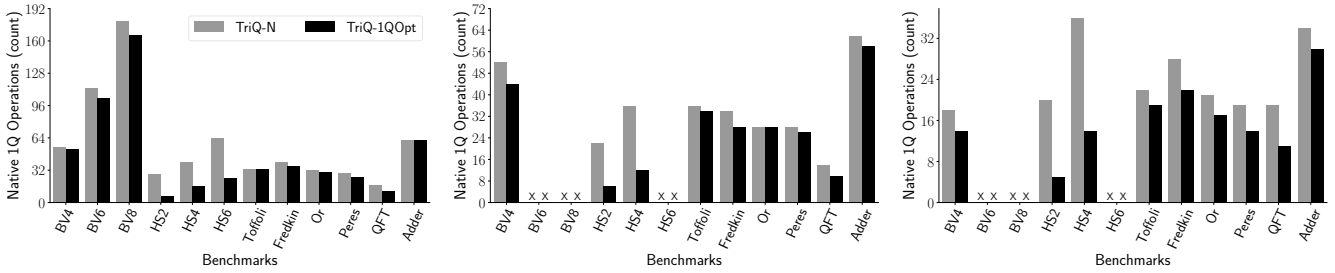
Noise-unaware communication optimization is useful, but not always sufficient. For example, Figure 10c shows that for QFT, TriQ-1QOptC performs worse than TriQ-1QOpt. Why is that? The machine's calibration data indicates that in doing noise-unaware mapping solely for communication distance, this compilation inadvertently resulted in qubit mappings that use less reliable hardware. This motivates the TriQ-1QOptCN approach.

Figure 11 shows the success rate and 2Q gate count for TriQ-1QOptC, TriQ-1QOptCN, and Qiskit. The large reductions in 2Q gate counts because of communication optimization give our methods significant benefits. Qiskit uses lexicographic mapping of qubits and performs swap optimization uses a greedy stochastic algorithm. It underperforms our methods because it always uses the first few qubits in the device regardless of noise and program communication requirements. 2Q optimizations again parlay into success rate benefits. TriQ-1QOptCN succeeds on all 12 benchmarks, and outperforms TriQ-1QOptC by up to 2.8x (geomean 1.4x). In Toffoli, it under performs slightly, but the loss is comparable to the noise-margin in these runs and not significant. TriQ-1QOptC fails to produce the correct answer in the Fredkin benchmark, while TriQ-1QOptCN succeeds. In contrast, the IBM Qiskit compiler (noise-unaware) fails to produce the correct answer on 7/12 benchmarks. To compute improvement factors, we used the measured probability of the correct answer produced by Qiskit (other incorrect answers had higher probability). TriQ-1QOptCN obtains up to 28x improvement (geomean 3.0X) over Qiskit.

Figure 11c and 11d show how TriQ-1QOptCN can improve in success rate on Rigetti. For most benchmarks we see significant improvements. We obtain up to 2.3x improvement over the Quil compiler (geomean 1.45x). Quil uses a simple initial qubit mapping, with insufficient communication optimization and no noise-awareness.

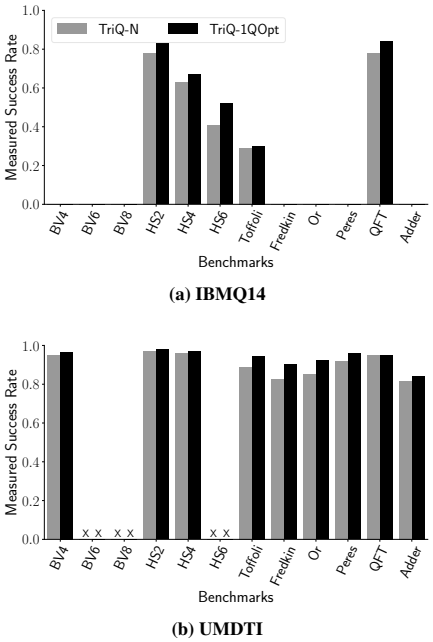
Finally, Figure 11e and 11f show the success rate improvements of TriQ-1QOptCN for UMDTI. This machine has a fully-connected topology and low error rates. For the applications that fit into its 5 qubits, success rates are high across the board; therefore, we created more challenging 3-qubit benchmarks with longer gate sequences. Namely, we used iterations of Toffoli or Fredkin gate sequences to

[3]The Rigetti machines were not available at the time these were gathered.



(a) IBMQ14 (b) Rigetti Agave (c) UMDTI

Figure 8: Importance of single qubit gate optimization. Counts of native 1Q operations (actual X and Y pulses applied on the qubits) when the compiler optimizes gates using properties of the gate set. TriQ-1QOpt reduces operation count by up to 4.6x by decomposing and optimizing composite gates. In (b) and (c), HS6, BV6 and BV8 are marked "X" because of size restrictions on Agave and UMDTI.



(a) IBMQ14

(b) UMDTI

Figure 9: Importance of single qubit gate optimization. Measured success rate for TriQ-N and TriQ-1QOpt. Across machines, TriQ-1QOpt coalesces operations and maximizes the use of error-free Z rotations, providing up to 1.26x improvement in success rate. In (a), the bars with zero height correspond to failed runs where the correct answer did not dominate in the output distribution. In (b), HS6, BV6 and BV8 are marked "X" because of size restrictions on UMDTI.

test the effects of noise on programs with varying lengths. TriQ-1QOptCN offers 1.47x improvement for Toffoli and 1.35x improvement for Fredkin sequences, compared to TriQ-1QOptC. The gains increase with increasing program length. TriQ-1QOptCN obtains these improvements because it places frequently interacting pairs of program qubits on the best hardware.

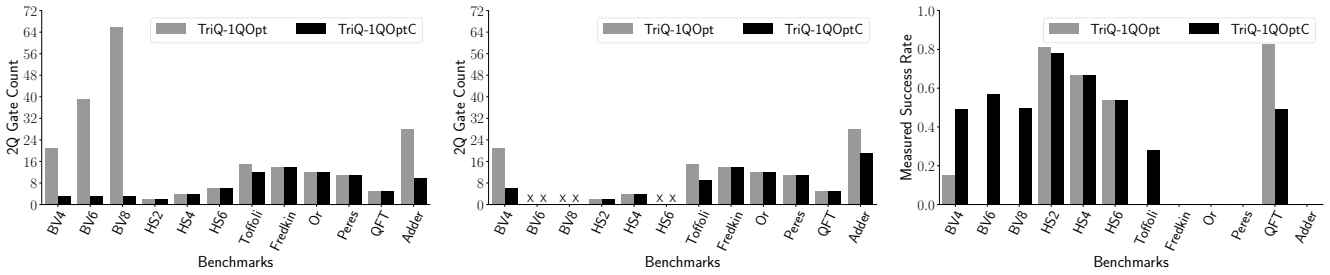
Whereas superconducting qubits can have fundamental physical defects [35], each qubit in an ion trap identical and defect-free. However, noise in ion trap machines stems from difficulties in qubit control and sensitivity to motional mode drifts; these cause 1-3% fluctuations in error rates. Our UMDTI experiment is the first to demonstrate that longer programs can obtain performance improvements by adapting to these small noise variations. For larger ion traps, reduced interaction strengths and therefore higher error rates are expected between ions which are farther apart [37, 45]. This suggests that our noise-adaptive methods will be even more important then.

6.4 Putting it all together

We used TriQ to compile the 12 benchmarks for all the seven systems using all the optimizations i.e., TriQ-1QOptCN. Since TriQ-1QOptCN outperforms vendor compilers, it allows us to understand benchmark performance without introducing compiler inefficiencies.

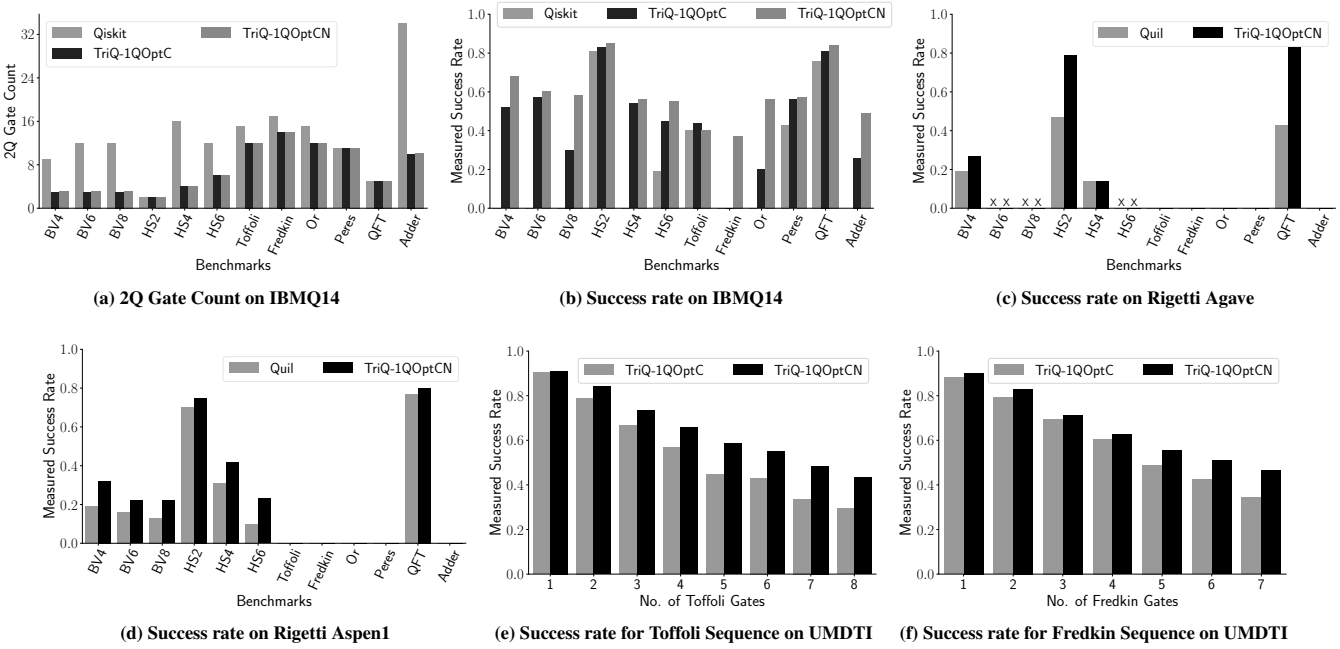
Figure 12 shows the success rate measured for each of the compiled executables. On benchmarks that fit on the current UMDTI machine, its low gate errors and good topology give it an advantage over other machines. For the superconducting machines, application-device topology match matters. For example, the Toffoli, Fredkin and Or benchmarks (3-qubit triangle) fit the triangular topology of IBMQ5 (see Figure 1). This offers higher performance than the grid topology of IBMQ16. Assuming reasonable application-topology match, having more qubits is better because it allows more flexibility in choosing a mapping which avoids the noisy regions of the machine.

Rigetti's best gates are comparable to IBM (IBMQ5 and Rigetti on HS2 and QFT). Application-topology mismatch and higher noise variation affect the success rate of larger benchmarks. Comparing the results on the newer Aspen1 and Aspen3 machines and the older Agave machine indicates significant improvements in qubit and gate reliability. On Aspen1 and Aspen3, more powerful native operations can be exploited to reduce the number of 2Q operations for some of our benchmarks. These operations were not software-visible on Aspen1 and Aspen3 in our experiments; exposing them to the compiler would enable higher success rates.



(a) 2Q Gate Counts for IBMQ14 (b) 2Q Gate Counts for Rigetti Agave (c) Success Rates for IBMQ14

Figure 10: Importance of communication optimization. (a) and (b) compare the 2Q gate counts in IBMQ14 and Rigetti without and with communication optimization. (c) provides corresponding IBMQ14 success rates. On IBMQ14, TriQ provides up to 22x reduction in 2Q gate count and enables more programs to succeed compared to the default mapping. On Rigetti, TriQ obtains up to 3.5x improvement in 2Q gate count. On some benchmarks such as QFT on IBMQ14, noise-unaware communication optimization places compute on high-error resources, which leads to low success rate. In (b), BV6, BV8 and HS6 are marked "X" because of size restrictions on Agave. In (c) runs with zero height bars correspond to failed runs where the correct answer did not dominate in the output distribution.



(a) 2Q Gate Count on IBMQ14 (b) Success rate on IBMQ14 (c) Success rate on Rigetti Agave

(d) Success rate on Rigetti Aspen1 (e) Success rate for Toffoli Sequence on UMDTI (f) Success rate for Fredkin Sequence on UMDTI

Figure 11: Importance of noise-adaptivity. (a) and (b) compare TriQ-1QOptC, TriQ-1QOptCN and IBM Qiskit compiler for IBMQ14. By optimizing gate errors, communication and single qubit gates simultaneously, TriQ-1QOptCN obtains up to 28x improvement over Qiskit and 2.8x over TriQ-1QOptC. (c) and (d) compare Rigetti's Quil compiler and TriQ-1QOptCN on Rigetti Agave and Aspen1. TriQ-1QOptCN obtains up to 2.3x improvement over Quil. (e) and (f) compare TriQ-1QOptC and TriQ-1QOptCN on UMDTI, where noise-adaptivity provides up to 1.47x improvement. In (b), (c) and (d) runs with zero height bars correspond to failed runs where the correct answer did not dominate in the output distribution.

6.5 Scalability of our Toolflow

Finally, although compile time has not been a prominent goal for this work, we wanted to test the degree to which our optimization approaches would scale up to larger NISQ machines that do not yet exist. Toward this goal, we measured the toolflow runtime to perform full optimizations (including noise-awareness) for proposed Quantum Supremacy circuits, mapping onto the 72-qubit system announced by Google [17] (but not yet operational). We assigned

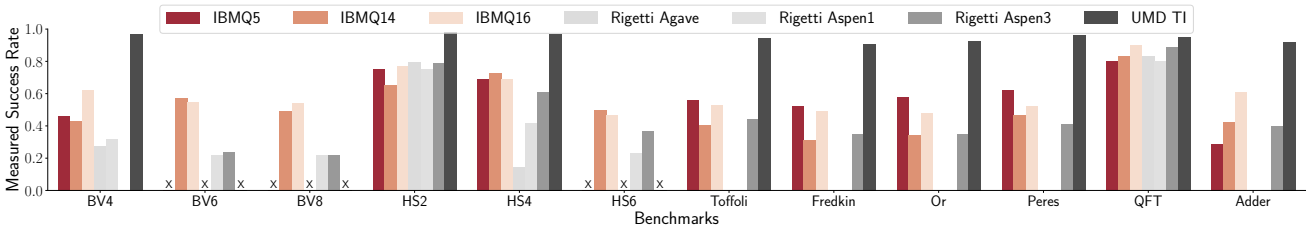


Figure 12: Success rate for 12 benchmarks on seven systems. Success rates varies drastically across systems and is influenced by error rates, qubit connectivity, and application-machine topology match. Benchmarks that are too large to be mapped onto a machine are marked "X". ***This comparison is intended to understand the impact of architectural design choices such as gate set and connectivity on benchmark performance and is not intended to pick a winning technology, vendor or implementation. Individual benchmark performance numbers may change over time — these measurements represent a snapshot of the performance of these systems when we performed the experiments.***

error rates for each gate by sampling from 100 days of error data from IBM devices. TriQ-1QOptCN scales well up to 72 qubits and is three orders of magnitude faster than [46]. The scaling is independent of gate count because the solver only creates variables for distinct two-qubit gates between a set of n program-qubits, bounding the number of variables by $O(n^2)$. This evaluation gives us confidence that our approaches will scale through NISQ machines of considerable size and capability.

7 ARCHITECTURE IMPLICATIONS

Section 6's results have already offered important insights regarding the interplay of hardware design choices (e.g. qubit technologies, communication topologies etc.) and the software toolflows and execution stack. In particular, QC sits at an exciting inflection point where decisions about the hardware-software interface are timely and important.

Native gates and software-visible operations: Our experiments show that when native gates are software-visible, the compiler can optimize programs heavily, in ways that are well-suited to the underlying device often resulting in lower error rates. This provides both execution time and success rate benefits. Ideally, vendors should expose the most fundamental 1Q and 2Q operations, allowing the compiler to perform fine-grained optimization across many gates. Such optimizations are especially relevant in systems such as UMDTI and Rigetti, where several native gates are required to construct a single CNOT gate. Providing powerful native operations such as UMDTI's flexible 1Q rotations provides further benefits, enabling a large number of operations to be simplified into fewer operations on the qubits. From a technology perspective, it may be difficult for some qubit technologies to support very flexible native operations. When such limits are reached, our results show the value of exposing to software what native operations *are* available, to allow ample compiler optimization. These observations are also corroborated by IBM's recent announcement that they will expose pulse-level qubit control via a programmable interface [43]. As an analogy to classical microprocessors, this is akin to making micro-operations software-visible [69].

Communication topology: Our experiments show that communication topology has a significant impact on performance and success rate. Comparing near-neighbor vs. fully-connected implementations like IBMQ14 vs. UMDTI, our results show that machines with richer qubit connectivity allow a wider variety of programs to execute successfully. Our results also demonstrate that when full connectivity cannot be reached (e.g. offering it is physically easier in ion traps, compared to superconducting systems) compilers like ours can optimize communication for the achievable topology.

Noise rates and variability: Clearly, lower error rates are preferable—crucial for benchmark success rate. Nonetheless, our studies show the degree to which noise-aware toolflows can be beneficial even on low-error platforms. Even small variations in low error rates like UMDTI's have significant impact on application success. Comparing devices such as Rigetti Aspen3, IBMQ16 and IBMQ14, it is also important that large connected segments of the system have good error rates; contiguous regions of low-error gates are the most useful. Finally, it is already the norm in QC to compile programs for a particular input size, and our work further demonstrates the value of also recompiling applications to account for up-to-date noise data as well.

Execution stack for QC: The TriQ toolflow has core functionality which is portable across diverse platforms, but also allows full top-to-bottom optimizations for device and application characteristics provided as compile-time inputs. Our results show the value of these device-aware and application-aware characteristics in exploiting successfully the limited resources of NISQ machines. Exploiting low-error hardware and native gates was crucial to our success rates. This points to the conclusion that QC machines may not yet be ready for abstractions or virtualizations that abstract too much of the information flow between software and hardware.

8 RELATED WORK

With prototype QCs quite recent, [38] provided the first experimental comparison on IBMQ5 and UMDTI using 4 hand-optimized benchmarks on 2 5-qubit machines, and observed the importance of program-device topology match. Our work extends to more and considerably larger prototypes. We are also the first to do multi-platform characterizations via a a top-to-bottom compiler toolflow for real-system executions across platforms. Where [38] hand-placed each qubit, our work is the first to perform cross-platform comparisons of optimizations leveraging qubit, topology and noise properties within automated multi-platform compilers. These cross-platform

optimizations and observations inform Section 7's summary of the work's hardware and architectural implications.

At the other end of the stack, QC programming languages and compilers have seen ongoing attention. In addition to Scaffold, other examples are Quipper [19, 20], and LIQUi|⟩ [67]. IBM Qiskit is a Python-based framework to program and compile code for the IBM systems, generating OpenQASM [10]. Likewise PyQuil [55, 56] is a Python-based framework for Rigetti systems, generating Quil [62]. ProjectQ [54, 64] is another Python-based framework to describe quantum circuits and compile them for different machines. Qiskit and Quil consider communication optimization while ProjectQ does not currently support non-grid topology [53]. Through its top-to-bottom empirical results, our work shows the importance of both device-specific and application-specific optimizations through the toolflow. We demonstrate how to achieve this with a core set of passes that apply across diverse platforms, by taking device characteristics as input.

[22, 61, 66, 70, 71] develop methods for optimizing communication on current or small systems, but do not consider noise data. Compared to the open source implementation of [71], TriQ reduces 2Q gate count by 1.2x (geomean), up to 2X. In [65], they propose the use of noise-aware qubit mapping and movement policies on the 20-qubit IBM system and report real executions on IBMQ5. For BV4, [65] reports a success rate of 0.23 on the 5-qubit IBM system. Since the machine state influences success rate, in order to make a fair comparison, we evaluated TriQ on 6 days having different error conditions. We obtained 2X better success rate ranging from 0.43 to 0.51 (average 0.47) indicating that our optimizations are effective. [46] developed a noise-aware compiler for the Scaffold language, targeted for systems with grid topologies and demonstrated the benefits of noise-adaptive compilation on IBMQ16. However, none of these provided multi-platform optimizations. Ours is the first work to build a full-stack toolflow in support of cross-platform empirical experiments comparing multiple QC prototypes.

9 CONCLUSIONS

After decades of gradual progress, NISQ QC prototypes are now available for experiments. Several machines exist in the 5-50 qubit range, representing widely-divergent design points regarding qubit technologies, topologies, and error rates. This diversity offers opportunities for cross-platform design studies that elucidate how device technologies influence other hardware design choices, and how compiler and software choices can offer optimizations that mitigate challenging hardware characteristics. To study key system design questions, our work built TriQ, a top-to-bottom toolflow which compiles high-level language programs for multiple target systems. Using real-systems measurements on seven devices, our experiments with TriQ show several examples of how leveraging hardware details in the compiler can provide a significant boost in program success rates. Our empirical cross-platform and cross-technology study offers forward-looking insights for compiler and architecture design for NISQ systems.

ACKNOWLEDGMENTS

This work is funded in part by EPiQC, an NSF Expedition in Computing, under grants CCF-1730082. We thank Christopher Monroe, Kevin Akiva Landsman and Daiwei Zhu from University of Maryland for access to the ion trap system. We thank Ryan Karle, Marcus da Silva, Amy Brown, Tushar Mittal and Nima Alidoust from Rigetti. We thank Ken Brown and Fred Chong for their insightful comments and suggestions.

REFERENCES

[1] Ali Javadi Abhari, Arvin Faruque, Mohammad Javad Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, Fred Chong, Margaret Martonosi, Martin Suchara, Ken Brown, Massoud Pedram, and Todd Brun. 2012. *Scaffold: Quantum Programming Language*. Report TR-934-12. Princeton University.

[2] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. 2013. A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 32, 6 (June 2013), 818–830. https://doi.org/10.1109/TCAD.2013.2244643

[3] Ethan Bernstein and Umesh Vazirani. 1993. Quantum Complexity Theory. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing (STOC '93)*. ACM, 11–20. https://doi.org/10.1145/167088.167097

[4] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549 (13 Sep 2017). http://dx.doi.org/10.1038/nature23474

[5] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. *vZ* - An Optimizing SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, Christel Baier and Cesare Tinelli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 194–199.

[6] S. A. Caldwell, N. Didier, C. A. Ryan, E. A. Sete, A. Hudson, P. Karalekas, R. Manenti, M. P. da Silva, R. Sinclair, E. Acala, N. Alidoust, J. Angeles, A. Bestwick, M. Block, B. Bloom, A. Bradley, C. Bui, L. Capelluto, R. Chilcott, J. Cordova, G. Crossman, M. Curtis, S. Deshpande, T. El Bouayadi, D. Girshovich, S. Hong, K. Kuang, M. Lenihan, T. Manning, A. Marchenkov, J. Marshall, R. Maydra, Y. Mohan, W. O'Brien, C. Osborn, J. Otterbach, A. Papageorge, J.-P. Paquette, M. Pelstring, A. Polloreno, G. Prawiroatmodjo, V. Rawat, M. Reagor, R. Renzas, N. Rubin, D. Russell, M. Rust, D. Scarabelli, M. Scheer, M. Selvanayagam, R. Smith, A. Staley, M. Suska, N. Tezak, D. C. Thompson, T.-W. To, M. Vahidpour, N. Vodrahalli, T. Whyland, K. Yadav, W. Zeng, and C. Rigetti. 2018. Parametrically Activated Entangling Gates Using Transmon Qubits. *Phys. Rev. Applied* 10 (Sep 2018), 034050. Issue 3. https://doi.org/10.1103/PhysRevApplied.10.034050

[7] Andrew M. Childs and Wim van Dam. 2007. Quantum Algorithm for a Generalized Hidden Shift Problem. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1225–1232. http://dl.acm.org/citation.cfm?id=1283383.1283515

[8] J. I. Cirac and P. Zoller. 1995. Quantum Computations with Cold Trapped Ions. *Phys. Rev. Lett.* 74 (May 1995), 4091–4094. Issue 20. https://doi.org/10.1103/PhysRevLett.74.4091

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[10] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. Open Quantum Assembly Language. arXiv:1707.03429

[11] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340.

[12] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. 2016. Demonstration of a small programmable quantum computer with atomic qubits. *Nature* 536 (03 Aug 2016). http://dx.doi.org/10.1038/nature18648

[13] David P. DiVincenzo. 2000. The Physical Implementation of Quantum Computation. *Fortschritte der Physik* 48, 9-11 (2000), 771–783. https://doi.org/10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/1521-3978

[14] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. 2018. eQASM: An Executable Quantum Instruction Set Architecture. arXiv:arXiv:1808.02449

[15] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. 2017. An Experimental Microarchitecture for a Superconducting Quantum Processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17)*. ACM, 813–825. https://doi.org/10.1145/3123939.3123952

[16] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. 2018. A Microarchitecture for a Superconducting

Quantum Processor. *IEEE Micro* 38, 3 (May 2018), 40–47. https://doi.org/10.1109/MM.2018.032271060

[17] Google. 2018. A Preview of Bristlecone, Google's New Quantum Processor. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html. Accessed: 2018-08-05.

[18] Google. 2018. Cirq. https://github.com/quantumlib/Cirq. Accessed: 2018-11-29.

[19] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A Scalable Quantum Programming Language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*. ACM, 333–342. https://doi.org/10.1145/2491956.2462177

[20] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A Scalable Quantum Programming Language. *SIGPLAN Not.* 48, 6 (June 2013), 333–342. https://doi.org/10.1145/2499370.2462177

[21] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing (STOC '96)*. ACM, 212–219. https://doi.org/10.1145/237814.237866

[22] Gian Giacomo Guerreschi and Jongsoo Park. 2017. Two-step approach to scheduling quantum circuits. arXiv:1708.00023

[23] T. P. Harty, D. T. C. Allcock, C. J. Ballance, L. Guidoni, H. A. Janacek, N. M. Linke, D. N. Stacey, and D. M. Lucas. 2014. High-Fidelity Preparation, Gates, Memory, and Readout of a Trapped-Ion Quantum Bit. *Phys. Rev. Lett.* 113 (Nov 2014), 220501. Issue 22. https://doi.org/10.1103/PhysRevLett.113.220501

[24] Charles D. Hill, Eldad Peretz, Samuel J. Hile, Matthew G. House, Martin Fuechsle, Sven Rogge, Michelle Y. Simmons, and Lloyd C. L. Hollenberg. 2015. A surface code quantum computer in silicon. *Science Advances* 1, 9 (2015). https://doi.org/10.1126/sciadv.1500707 arXiv:http://advances.sciencemag.org/content/1/9/e1500707.full.pdf

[25] IBM. 2018. IBM Announces Advances to IBM Quantum Systems and Ecosystem. https://www-03.ibm.com/press/us/en/pressrelease/53374.wss. Accessed: 2018-08-05.

[26] IBM. 2018. IBM Qiskit. https://qiskit.org/. Accessed: 2018-08-05.

[27] IBM. 2018. IBM Quantum Devices. https://quantumexperience.ng.bluemix.net/qx/devices. Accessed: 2018-05-16.

[28] IBM. 2018. IBM Quantum Experience. https://github.com/Qiskit/qiskit-api-py. Accessed: 2018-11-16.

[29] IBM. 2018. IBMQ Backend Information. https://github.com/Qiskit/ibmq-device-information. Accessed: 2018-11-01.

[30] Intel. 2018. CES 2018: Intel's 49-Qubit Chip Shoots for Quantum Supremacy. https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy. Accessed: 2018-08-05.

[31] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. 2014. ScaffCC: A Framework for Compilation and Analysis of Quantum Computing Programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers (CF '14)*. ACM, Article 1, 10 pages. https://doi.org/10.1145/2597917.2597939

[32] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* 549 (13 Sep 2017). http://dx.doi.org/10.1038/nature23879

[33] Torsten Karzig, Christina Knapp, Roman M. Lutchyn, Parsa Bonderson, Matthew B. Hastings, Chetan Nayak, Jason Alicea, Karsten Flensberg, Stephan Plugge, Yuval Oreg, Charles M. Marcus, and Michael H. Freedman. 2017. Scalable designs for quasiparticle-poisoning-protected topological quantum computation with Majorana zero modes. *Phys. Rev. B* 95 (Jun 2017), 235305. Issue 23. https://doi.org/10.1103/PhysRevB.95.235305

[34] A.Yu. Kitaev. 2003. Fault-tolerant quantum computation by anyons. *Annals of Physics* 303, 1 (2003), 2 – 30. https://doi.org/10.1016/S0003-4916(02)00018-0

[35] P. V. Klimov, J. Kelly, Z. Chen, M. Neeley, A. Megrant, B. Burkett, R. Barends, K. Arya, B. Chiaro, Yu Chen, A. Dunsworth, A. Fowler, B. Foxen, C. Gidney, M. Giustina, R. Graff, T. Huang, E. Jeffrey, Erik Lucero, J. Y. Mutus, O. Naaman, C. Neill, C. Quintana, P. Roushan, Daniel Sank, A. Vainsencher, J. Wenner, T. C. White, S. Boixo, R. Babbush, V. N. Smelyanskiy, H. Neven, and John M. Martinis. 2018. Fluctuations of Energy-Relaxation Times in Superconducting Qubits. *Phys. Rev. Lett.* 121 (Aug 2018), 090502. Issue 9. https://doi.org/10.1103/PhysRevLett.121.090502

[36] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization (CGO '04)*. IEEE Computer Society, Washington, DC, USA, 75–. http://dl.acm.org/citation.cfm?id=977395.977673

[37] Bjoern Lekitsch, Sebastian Weidt, Austin G. Fowler, Klaus Mølmer, Simon J. Devitt, Christof Wunderlich, and Winfried K. Hensinger. 2017. Blueprint for a microwave trapped ion quantum computer. *Science Advances* 3, 2 (2017). https://doi.org/10.1126/sciadv.1601540 arXiv:http://advances.sciencemag.org/content/3/2/e1601540.full.pdf

[38] Norbert M. Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. 2017. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3305–3310. https://doi.org/10.1073/pnas.1618020114 arXiv:http://www.pnas.org/content/114/13/3305.full.pdf

[39] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. 2013. Quantum algorithms for supervised and unsupervised machine learning. arXiv preprint arXiv:1307.0411.

[40] J. Majer, J. M. Chow, J. M. Gambetta, Jens Koch, B. R. Johnson, J. A. Schreier, L. Frunzio, D. I. Schuster, A. A. Houck, A. Wallraff, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf. 2007. Coupling superconducting qubits via a cavity bus. *Nature* 449 (27 Sep 2007). http://dx.doi.org/10.1038/nature06184

[41] Igor L. Markov, Aneeqa Fatima, Sergei V. Isakov, and Sergio Boixo. 2018. Quantum Supremacy Is Both Closer and Farther than It Appears. arXiv:1807.10749

[42] Margaret Martonosi and Martin Roetteler. 2019. Next Steps in Quantum Computing: Computer Science's Role. arXiv:arXiv:1903.10541 arXiv:1903.10541.

[43] David C. McKay, Thomas Alexander, Luciano Bello, Michael J. Biercuk, Lev Bishop, Jiayin Chen, Jerry M. Chow, Antonio D. Córcoles, Daniel Egger, Stefan Filipp, Juan Gomez, Michael Hush, Ali Javadi-Abhari, Diego Moreda, Paul Nation, Brent Paulovicks, Erick Winston, Christopher J. Wood, James Wootton, and Jay M. Gambetta. 2018. Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments. arXiv:1809.03452

[44] N. David Mermin. 2007. *Quantum Computer Science: An Introduction*. Cambridge University Press.

[45] Thomas Monz, Philipp Schindler, Julio T. Barreiro, Michael Chwalla, Daniel Nigg, William A. Coish, Maximilian Harlander, Wolfgang Hänsel, Markus Hennrich, and Rainer Blatt. 2011. 14-Qubit Entanglement: Creation and Coherence. *Phys. Rev. Lett.* 106 (Mar 2011), 130506. Issue 13. https://doi.org/10.1103/PhysRevLett.106.130506

[46] Prakash Murali, Jonathan Baker, Ali Javadi Abhari, Fred Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*.

[47] NASEM. 2019. Quantum Computing: Progress and Prospects. https://doi.org/10.17226/25196

[48] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press.

[49] Tony Nowatzki, Michael Sartin-Tarm, Lorenzo De Carli, Karthikeyan Sankaralingam, Cristian Estan, and Behnam Robatmili. 2013. A General Constraint-centric Scheduling Framework for Spatial Architectures. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*. ACM, 495–506. https://doi.org/10.1145/2491956.2462163

[50] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* 5 (23 Jul 2014). http://dx.doi.org/10.1038/ncomms5213 Article.

[51] Jarryd J. Pla, Kuan Y. Tan, Juan P. Dehollain, Wee H. Lim, John J. L. Morton, David N. Jamieson, Andrew S. Dzurak, and Andrea Morello. 2012. A single-atom electron spin qubit in silicon. *Nature* 489 (19 Sep 2012). https://doi.org/10.1038/nature11449

[52] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. arXiv:1801.00862

[53] Project Q. 2018. Bug Report: StatePreparation causes "Circuit cannot be mapped without using Swaps" on IBM. https://github.com/ProjectQ-Framework/ProjectQ/issues/279. Accessed: 2018-10-27.

[54] Project Q. 2018. Project Q. https://projectq.ch/. Accessed: 2018-05-16.

[55] Rigetti. 2018. PyQuil. https://github.com/rigetticomputing/pyquil. Accessed: 2018-08-01.

[56] Rigetti. 2018. Rigetti Forest. http://forest.rigetti.com. Accessed: 2018-08-01.

[57] Chad Rigetti, Jay M. Gambetta, Stefano Poletto, B. L. T. Plourde, Jerry M. Chow, A. D. Córcoles, John A. Smolin, Seth T. Merkel, J. R. Rozen, George A. Keefe, Mary B. Rothwell, Mark B. Ketchen, and M. Steffen. 2012. Superconducting qubit in a waveguide cavity with a coherence time approaching 0.1 ms. *Phys. Rev. B* 86 (Sep 2012), 100506. Issue 10. https://doi.org/10.1103/PhysRevB.86.100506

[58] ScaffCC. 2018. ScaffCC Compiler. https://github.com/epiqc/ScaffCC. Accessed: 2018-05-16.

[59] Sarah Sheldon, Easwar Magesan, Jerry M. Chow, and Jay M. Gambetta. 2016. Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Phys. Rev. A* 93 (Jun 2016), 060302. Issue 6. https://doi.org/10.1103/PhysRevA.93.060302

[60] P. Shor. 1999. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.* 41, 2 (1999), 303–332. https://doi.org/10.1137/S0036144598347011 arXiv:https://doi.org/10.1137/S0036144598347011

[61] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintao Pereira. 2018. Qubit Allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization (CGO*

2018). ACM, 113–125. https://doi.org/10.1145/3168822

[62] Robert S. Smith, Michael J. Curtis, and William J. Zeng. 2016. A Practical Quantum Instruction Set Architecture. arXiv:1608.03355

[63] Mathias Soeken, Thomas Haner, and Martin Roetteler. 2018. Programming Quantum Computers Using Design Automation. arXiv:1803.01022

[64] Damian S. Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: an open source software framework for quantum computing. *Quantum* 2 (Jan. 2018), 49. https://doi.org/10.22331/q-2018-01-31-49

[65] Swamit S. Tannu and Moinuddin K. Qureshi. 2018. A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. arXiv:1805.10224

[66] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. 2018. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology* 3, 2 (2018), 025004. http://stacks.iop.org/2058-9565/3/i=2/a=025004

[67] Dave Wecker and Krysta M. Svore. 2014. LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing. arXiv:1402.4467

[68] Wikipedia. 2018. Conversion between Quaternions and Euler Angles. https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles. Accessed: 2018-11-27.

[69] M. V. Wilkes. 1989. The Early British Computer Conferences. MIT Press, Cambridge, MA, USA, Chapter The Best Way to Design an Automatic Calculating Machine, 182–184. http://dl.acm.org/citation.cfm?id=94938.94976

[70] Xin Zhang, Hong Xiang, Tao Xiang, Li Fu, and Jun Sang. 2018. An efficient quantum circuits optimizing scheme compared with QISKit. arXiv:1807.01703

[71] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2017. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. arXiv:1712.04722