Architecting Noisy Intermediate-Scale Quantum Computers: A Real-System Study

Prakash Murali
Princeton University

Norbert M. Linke
Joint Quantum Institute, 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 and Instituto Nacional de Astrofísica, Óptica y Electrónica

***Abstract*—Current quantum computers have very different qubit implementations, instruction sets, qubit connectivity, and noise characteristics. Using real-system evaluations on seven quantum systems from three leading vendors, our work explores fundamental design questions concerning hardware choices, architecture, and compilation.**

Quantum computing (QC) is a fundamentally new model of computation, which exploits quantum mechanical phenomena to perform computation. QC systems use *qubits* (quantum bits) to represent information and *gates* (quantum instructions) to manipulate quantum information. While the basic principles of QC have been known since the 1980s, recent hardware progress has ushered in the era of noisy intermediate-scale quantum (NISQ) devices. These systems represent an important milestone toward large scale QC, and are expected to scale to 500–1000 qubits in coming years. In spite of being too error-prone and resource-constrained for well-known applications like Shor's factoring, NISQ systems are capable of very powerful computations. Notably, Google recently demonstrated a classically

Digital Object Identifier 10.1109/MM.2020.2985683
Date of publication 6 April 2020; date of current version 22 May 2020.

intractable computation on an NISQ system with 54 qubits.[1]

Being early-stage, NISQ devices are highly diverse in terms of hardware and architecture. Leading QC vendors including IBM, Rigetti, Google, IonQ, and others have adopted very different approaches for building hardware qubits. To support their qubit choices, vendors have also chosen different instruction sets and hardware communication topologies. Further, QC systems also have variance in hardware noise, owing to fundamental challenges in qubit control and manufacturing. While this diversity itself poses a challenge for efficient and portable application execution, there is also a huge gap between the QC hardware that is buildable now, and the resource requirements of compelling real-world applications. Many interesting applications demand large systems with several thousand quantum bits and high-precision operations, but current hardware has less than 100 qubits and error-prone operations. To fully attain practical and powerful QC, computer architecture techniques and software toolchains must be employed to narrow the algorithm-to-devices resource gap across a wide range of algorithms and devices.

While the basic principles of QC have been known since the 1980s, recent hardware progress has ushered in the era of noisy intermediate-scale quantum (NISQ) devices. These systems represent an important milestone toward large scale QC, and are expected to scale to 500–1000 qubits in coming years.

To this end, our article[2] offers one of the deepest explorations of cross-platform characteristics in QC systems, presenting a full-stack, benchmark-driven, hardware–software analysis. Viewing QC through the lens of computer architecture, we evaluate important hardware design decisions (qubit types, system size, connectivity, noise), the hardware–software interface (gate set choices), and software optimizations to tackle fundamental design questions: What instructions should QC systems expose to software? Should instructions be unified in a device-independent ISA across different qubit types? How do hardware connectivity and noise characteristics impact benchmark performance? Can hardware limitations be overcome with a compiler?

To answer these questions, we use real-system measurements to evaluate a suite of QC applications on seven systems from three leading vendors—IBM, Rigetti, and University of Maryland. The systems studied represent different points in the design space, with two leading qubit technologies (superconducting and trapped ion qubits), different connectivity topologies, programming interfaces, and noise behavior. The diversity of systems studied is important for understanding which aspects of QC design hold across different design choices and which are more implementation specific. Our work represents the most comprehensive cross-platform, real-system measurements of QC prototypes ever performed.

On the other hand, this design space diversity also poses serious challenges for accurate comparative studies. In particular, our comparisons hinge on developing a toolflow and evaluation approach common to all platforms, and yet not penalizing any particular platform while pursuing toolflow generality. Our toolflow, TriQ, is the first top-to-bottom multivendor QC compiler toolflow. TriQ optimizes high-level language programs for QC hardware by leveraging deep but parameterized knowledge of the target device characteristics, including the gate set, connectivity, and noise profile. Importantly, TriQ avoids inefficiencies in vendor toolflows, offering up to two orders of magnitude higher reliability compared to IBM's Qiskit[3] and Rigetti's Quil[4] compiler which are the default toolchains for the respective hardware. TriQ, therefore, allows us to perform architectural analysis across diverse QC systems using high-level application performance measurements and is also a common compiler toolflow.

Our experiments with TriQ reveal several architectural insights for QC systems. We quantify the importance of gate set, ISA and connectivity choices and offers design recommendations. We also evaluate the effects of hardware noise on applications and the importance of software optimizations to mitigate such noise. Our results have also attracted significant academic and industry attention with vendors including IBM and Rigetti incorporating our optimizations in their compiler toolflows. In coming years, hardware and architectural insights from our study are likely to influence QC.

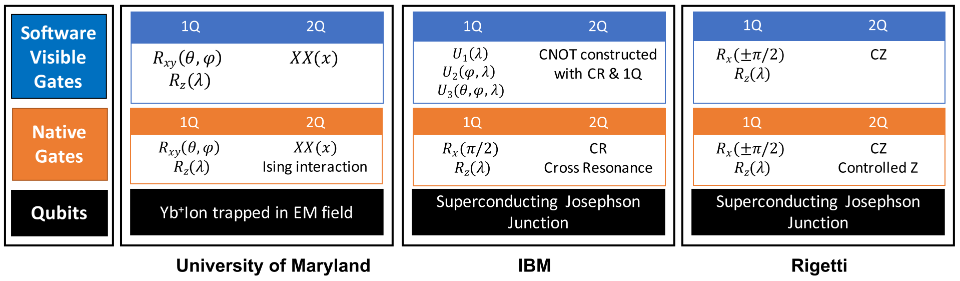


Figure 1. Hardware qubit technology, native gate set, and software-visible gate set in the systems used in our study. Each qubit technology lends itself to a set of native gates. For programming, vendors expose these gates in a software-visible interface or construct composite gates with multiple native gates.

BACKGROUND ON QC

A qubit is the fundamental building block a QC system. Unlike a classical bit which is restricted to be either in the state 0 or 1 at any instant, a qubit can exist in a *superposition* state where it is a probabilistic combination of the two basis states. This property allows an n-bit QC system to represent 2^n basis states simultaneously, unlike classical registers which can be in exactly one of the 2^n values at any given time. To manipulate information, QC gates are implemented to operate on one or more qubits, using some physical interaction such as a microwave or laser pulse. Similar to universal gates in classical systems, QC computations can be expressed using a small universal set of single (1Q) and two-qubit (2Q) gates. In particular, 2Q gates create *entanglement* which is a key property exploited by algorithms. To obtain classical output from the system, qubits are *measured* or *readout*, collapsing the superposition state to either 0 or 1.

QC ARCHITECTURE CHOICES AND TRADEOFFS

NISQ systems have very diverse hardware and architecture. While classical metrics such as performance (time) and area are important to evaluate these options, a key figure of merit in the current NISQ regime is the likelihood of correct execution of applications. Owing to the noise, a single execution of an application may be corrupted by noise. Hence, programs are typically run multiple times and the *success rate* is measured as the fraction of trials which yields the correct answer. Toward understanding how system design affects success rate and performance, we briefly discuss the key design choices here and refer the reader to our original paper for more details.[2]

Figure 1 shows the *different hardware qubit technologies* used in IBM, Rigetti, and UMD systems. IBM and Rigetti use superconducting qubits, while UMD uses trapped ion qubits. On one hand, these choices are similar to how classical computers can be realized using vacuum tubes, relay circuits or CMOS transistors. On the other hand, qubit technologies are very different and do not lend themselves to abstraction similar to the ON–OFF switch abstraction in classical technologies. For example, on IBM's superconducting qubits, the two-qubit interactions are achieved using the cross-resonance effect, where one qubit is driven at the resonant frequency of another qubit using a coupled hardware resonator. In contrast, in UMD's trapped ion qubits, two-qubit interactions are achieved using collective motional modes of an ion chain, mediated through laser pulses.

Owing to these fundamental differences, vendors implement *different native gates* or microoperations that are feasible on their platform. Figure 1 shows these native 1Q and 2Q gates. Even among superconducting qubits, the native interactions may be different. For example, Rigetti uses the controlled Z operation as the fundamental 2Q operation instead of the cross-resonance gate in IBM. Using these native gates, vendors choose a *software-visible* programming interface which includes either native gates themselves or composite gates which use multiple native gates. These choices for software-visible gates also differ widely across vendors.

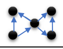
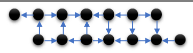
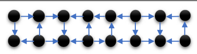
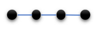
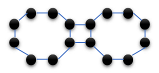
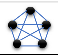
Machine	Qubits	2Q Gates	Coherence Time (us)	1Q Error (%)	2Q Error (%)	RO Error (%)	Qubit Topology
IBM Q5 Tenerife	5	6	40	0.2	4.76	6.21	
IBM Q14 Melbourne	14	18	30	1.19	7.95	9.09	
IBM Q16 Rüschlikon	16	22	40	0.22	7.14	4.15	
Rigetti Agave	4	3	15	3.68	10.8	16.37	
Rigetti Aspen1	16	18	20	3.43	8.92	5.56	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

Figure 2. 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).

Finally, qubit states are extremely fragile and difficult to control. On current systems, typical 2Q error rates are 1% –10%. Gate error rates also have large spatial and temporal variations depending on the qubit technology. Figure 3 shows these variations for an IBM system. If a program is executed on a subset of unreliable qubits, the success rate is greatly diminished. In addition, quantum state "decoheres" or loses reliability exponentially with time. That is, if a qubit is initialized, there is a short window of coherence time within which all gates in the program must be completed. On current superconducting systems, coherence time is typically less than 100 μs and varies across qubits. However, 2Q gates are relatively fast, requiring hundreds of nanoseconds. On trapped ion systems, coherence time is significantly longer (several seconds), but 2Q gates are slower, requiring several hundred microseconds.

Furthermore, QC devices have a qubit connection topology which determines the amount of communication required to perform 2Q gates. As shown in Figure 2, device topology varies across systems, with sparse nearest-neighbor connectivity in IBM and Rigetti, to full all-pairs connectivity in UMD. When full connectivity is not available, SWAP operations are used to enable 2Q gates between arbitrary pairs of qubits. These SWAPs increase program duration and more importantly, worsen the success rate. The choice of connectivity is not independent of the qubit type. Trapped ion qubits naturally support full connectivity, at least at small scales, while superconducting qubits typically use sparse connectivity because of difficulties in implementing dense physical interconnections.

Our work focuses on how these design tradeoffs influence QC computer architecture and software design. Toward this, we first develop a common compiler toolflow, TriQ, that maps programs onto diverse QC systems. Enabled by TriQ, we perform real-system experiments to explore the architectural design space.

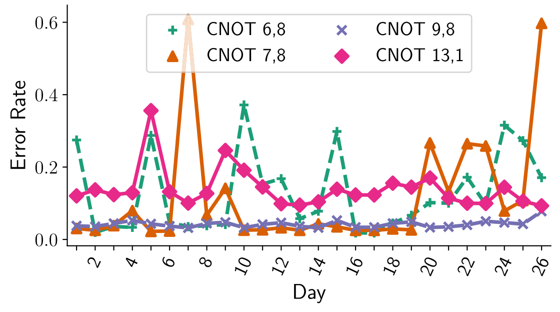


Figure 3. Daily variation of error rates of four hardware supported two-qubit controlled NOT gates in IBMQ14. The average error rate is approximately 8%, but there is up to 9x variation across qubits and days.

TRIQ: FULL-STACK MULTIVENDOR QC TOOLFLOW

Figure 4 illustrates the overall structure of TriQ. TriQ accepts Scaffold[5] programs as input. Scaffold is a C-like quantum language which has been used to develop large QC applications. Using ScaffCC, Scaffold's front-end compiler, TriQ generates an intermediate representation (IR) of the program and uses it as the input for the subsequent optimization passes. TriQ also takes hardware and system-specific features such as gate sets, connectivity, and noise information (from daily calibration logs for the systems) as configurable inputs. Hardware-dependent optimization using these inputs is a distinguishing feature of TriQ that allows it to obtain high success rates across platforms. As output, TriQ generates optimized code in the vendor-specified assembly code.

To compile the IR, the first step is to map program qubits onto distinct hardware qubits. For example, program qubits can be assigned to hardware qubits according to the order they are used in the program. This policy can result in high communication overhead and poor success rate when qubits participating in 2Q gates are not mapped close together. If program qubits are mapped onto unreliable hardware qubits, it can further worsen the success rate. Therefore, TriQ uses a *noise-adaptive mapping* strategy which optimizes both communication and reliability. TriQ chooses a set of qubits that match well with the communication requirements of the application and simultaneously, it ensures that this set of qubits has low error rates for the instruction mix of the application. TriQ implements this policy using a satisfiability modulo theory (SMT) optimization, solved using Microsoft's Z3 SMT solver.

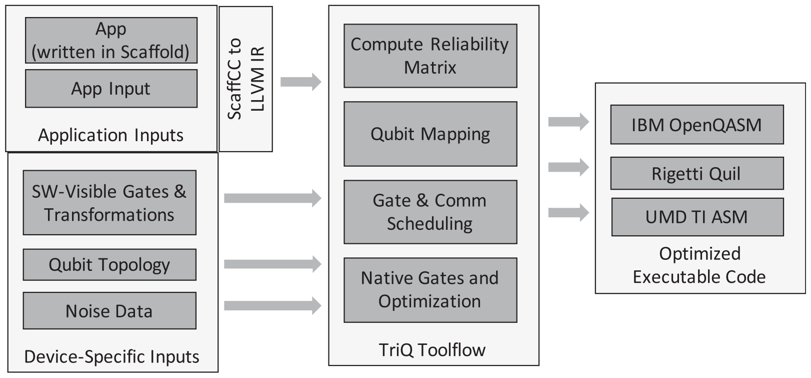


Figure 4. Overview of the TriQ toolflow. Inputs are high-level Scaffold programs and their inputs, as well as device-specific QC system properties. Output is optimized code in one of three vendor-specific executable formats.

To flexibly target different devices, we designed the SMT optimization to work with an abstract representation of the hardware. TriQ preprocesses the target device's connectivity graph and gate error rate data and converts them to a *reliability matrix* representation. For each pair of qubits, the matrix specifies the reliability of the lowest error rate path for a 2Q gate between the qubits. When two hardware qubits are far away in the communication topology, the reliability of the best path will be low. It will also be low if all paths between the two qubits have high error rate edges. Therefore, using the matrix, TriQ can pick communication- and reliability-optimized mappings. Since the core functionality of the pass operates using this matrix abstraction, we can flexibly compute good mappings for any device topology and noise profile simply by changing compile-time inputs.

Second, TriQ schedules gates in the program in a topologically-sorted order using the IR. This allows maximum operations to be executed in parallel, reducing the errors due to qubit decoherence. For devices which do not support full connectivity, TriQ automatically inserts the necessary communication operations to bring qubits into adjacent positions before executing 2Q gate. To improve success rates, TriQ incorporates noise-awareness in this step by selecting the lowest error rate paths for moving qubits, rather than any shortest distance path.

Third, TriQ translates high-level IR gates into device-specific IR. Using a set of legal code transformations that are provided as input, TriQ replaces IR gates with equivalent device-specific gates, e.g., OpenQASM code for IBM systems. During this pass, TriQ also applies a 1Q gate optimization where continuous sequences of 1Q gates are compressed into shorter sequences. TriQ exploits knowledge of hardware error rates in this step as well. On all three vendors, single qubit rotations gates along the *Z*-axis of the qubit have no error.[6] While compressing gate sequences, TriQ maximizes the use of these Z rotations, further increasing success rates.

REAL-SYSTEM ARCHITECTURAL STUDIES USING TRIQ

We performed real-system measurements for a set of 12 benchmarks on 7 QC systems. These benchmarks include important QC kernels such as the Toffoli gate and quantum Fourier transform

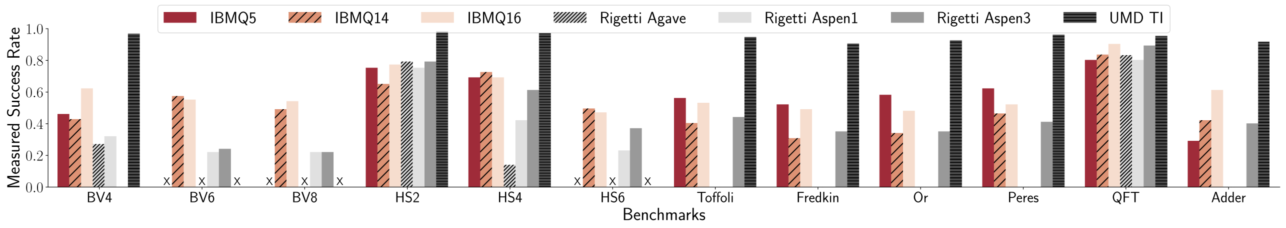


Figure 5. Success rate for 12 benchmarks on 7 systems. Success rates varies drastically across systems and is influenced by error rates, qubit connectivity, and application-machine topology match. Benchmarks that are too large to be mapped onto a machine are marked "X." This comparison is intended to 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.

operation. To understand architectural choices, we performed multiple experiments with each benchmark and system, varying the level of optimization and the inputs used for compilation. We used three main variants of the compiler with increasing levels of optimization for gate sequences, communication and for noise-adaptivity and a fourth baseline version with no optimization. We compared different executables in terms of instruction count and success rate. Figure 5 shows the measured success rates using TriQ's full optimizations. The key insights from our study are summarized next.

Importance of Gate Set Specificity: We studied whether it is beneficial to expose native gates to software, instead of abstracting them in a device-independent gate set. When TriQ has information about the native gate set, the gate optimization passes offers significant benefits. TriQ expresses several program instructions using a small number of native gates, leading to an average 50% reduction in the instruction count and up to 26% increase in success rate. Therefore, unlike prior proposals for device-independent ISAs for QC systems,[7,8] our results show that such abstractions are detrimental to high success rates. We recommend that vendors make the most low-level native gates in their devices software visible. As an analogy to classical microprocessors, this is similar to making microoperations software visible.[9]

Importance of Qubit Connectivity: Our work demonstrates that the match between application communication requirements and device topology significantly crucially impacts success rates. Comparing near-neighbor versus fully-connected systems (like IBM and UMD systems) shows that machines with dense qubit connectivity are less sensitive to application characteristics and allow a wider variety of programs to execute successfully. Compared to a baseline, TriQ's communication optimizations offer up to 22X reduction in 2Q gate counts. For certain programs, this means the difference between a failed execution where noise corrupts the output and a successful execution where the correct answers dominate. When the architecture does not have full connectivity, compilers like ours can allow applications to take maximum advantage of the available hardware resources.

Importance of Noise Adaptivity: Our work shows that the noise variability in QC hardware can be effectively mitigated by software techniques. By mapping programs onto reliable regions of the hardware and orchestrating communication along reliable hardware paths, TriQ effectively shields applications from spatiotemporal noise variations. These optimizations provide further average success rate gains of 2.8X over gate and communication optimizations, and allows more applications to execute successfully. Put together, TriQ's optimizations offer up to 1.5-28X higher success rates than IBM's Qiskit, Rigetti's Quil compiler, and hand optimized code from UMD. Our work is the first to show that such optimizations are important even on trapped ion systems which have less variability. Noise variations are likely in all near-term QC systems in the next 5 to 10 years. Therefore, compilers like TriQ will be crucial for reliable program executions.

TriQ's functionality is portable across diverse platforms while still performing full top-to-bottom optimizations for device and application characteristics provided as compile-time inputs. Leveraging microarchitecture details such as native gate sets and noise rates was the key to our improvements. Therefore, QC systems are not yet ready for device-independent abstraction layers that hide and obstruct information flow between hardware and software.

IMPACT OF OUR WORK

Recently, tech news was dominated by discussions of Google's so-called "quantum supremacy" announcement and reactions from other scientists and QC vendors.[1] While QC systems offering high revenue streams (e.g., as cloud accelerators) are still in the future, clearly QC is increasing in importance and has reached an inflection point in terms of engineering achievements in real implementations. This makes our work extremely timely, with high potential for impact. Just this year, several academic and industry vendors have already adjusted their compiler toolflows and aspects of their exposed gate sets in response to our work. Our optimizations are already part of IBM's Qiskit Terra compiler as of version 0.8 and Rigetti's Quil compiler version 1.16. TriQ, open sourced at https://github.com/prakashmurali/TriQ is also the first compiler for trapped ion systems.

Our study features systems with different qubit, noise, and architectural attributes and provides important insights for designing better architecture and hardware. These insights will likely influence future QC ISA design. Although QC applications work with any universal gate set, we demonstrate that shielding the natural gates for a qubit technology by abstracting them into more common gates imposes severe reliability and performance overheads on NISQ systems. Future QC ISAs need to work in tandem with the underlying qubit technology. Our work also underscores the importance of matching the application's communication requirements and hardware topology by codesigning them. When they are not well-matched, successful executions are unlikely.

Our work also breaks new ground in QC benchmarking by being distinct from the existing practices of measuring isolated hardware characteristics or benchmarking custom-designed applications. On one hand, vendors characterize systems in terms of metrics such as gate error rates and qubit coherence times. These metrics are isolated measurements for each hardware component, and not direct measurements of program behavior. TriQ enables direct and accurate measurements of program behavior across widely divergent QC platforms. In classical computing, this is akin to the difference between knowing characteristics like core counts and clock rate, versus knowing actual benchmark performance. On the other hand, vendors have developed benchmarking applications such as quantum volume. These methods use a family of custom generated circuits to measure hardware quality. Our work does not field a preferred benchmark, but instead relies on a suite of diverse applications to understand the impact of hardware on applications. This is similar to the difference between benchmarking supercomputers with LINPACK or other dedicated algorithms, and measuring the performance of real applications. We believe that this approach of application-based benchmarking will become common practice in QC, much like how benchmark suites such as SPEC are used for classical benchmarking.

Our study features systems with different qubit, noise, and architectural attributes and provides important insights for designing better architecture and hardware. These insights will likely influence future QC ISA design.

Most importantly, our work represents a significant advance on the way to practically viable QC, which requires us to close a five to six order of magnitude gap between algorithm needs and device capabilities. Our work demonstrates methods for achieving up to two orders of magnitude improvements in program success rates and our approaches work well across vendor implementations. In a world where increasing qubit count comes only with great engineering effort, our work offers substantial and orthogonal advances over underlying hardware progress alone.

ACKNOWLEDGMENTS

This work was supported in part by EPiQC, an NSF Expedition in Computing, under Grant CCF-1730082. The work of Cinthia Huerta Alderete was supported by CONACYT under Doctoral Grant 455378.

REFERENCES

1. Frank Arute *et al.* "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
2. P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: Architectural comparisons and design insights," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 527–540.
3. IBM, "IBM Qiskit," 2018, Accessed on: Jan. 1, 2020. [Online]. Available: https://qiskit.org/
4. Rigetti, "QuilC compiler," 2020, Accessed on: Jan. 1, 2020. [Online]. Available: https://github.com/rigetti/quilc
5. A. J. Abhari *et al.*, "Scaffold: Quantum programming language," Princeton University, Princeton, NJ, USA, Tech. Rep. TR-934-12, 2012.
6. D. C. McKay, C. J. Wood, S. Sheldon, J. M. Chow, and J. M. Gambetta, "Efficient z gates for quantum computing," *Phys. Rev. A*, vol. 96, Aug. 2017, Art. no. 022330.
7. X. Fu *et al.*, "A microarchitecture for a superconducting quantum processor," *IEEE Micro*, vol. 38, no. 3, pp. 40–47, May 2018.
8. A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," 2017, *arXiv:1707.03429*. [Online]. Available: https://arxiv.org/abs/1707.03429
9. M. V. Wilkes, "The best way to design an automatic calculating machine," in *The Early British Computer Conferences*. Cambridge, MA, USA: MIT Press, 1989, pp. 182–184.

Prakash Murali is currently a Ph.D. student in the Computer Science Department, Princeton University. His research focuses on accelerating the progress toward practical quantum computation using computer architecture and compilation techniques. Contact him at pmurali@cs.princeton.edu.

Norbert M. Linke is currently an Assistant Professor with the University of Maryland, College Park, and a Fellow of the Joint Quantum Institute (JQI) working on quantum algorithm implementations, quantum simulations, and quantum networking with trapped ions. He was a Postdoctoral Researcher and Research Scientist with the Group of Chris Monroe, University of Maryland. Linke received the graduate degree from the University of Ulm, and the doctorate degree from the University of Oxford. Contact him at linke@umd.edu.

Margaret Martonosi is currently the Hugh Trumbull Adams '35 Professor of Computer Science with Princeton University. Her research focuses on computer architecture and hardware–software interface issues in both classical and quantum systems. Martonosi received the Ph.D. degree in electrical engineering from Stanford University. She is a Fellow of IEEE and the Association for Computing Machinery (ACM). Contact her at mrm@princeton.edu.

Ali Javadi Abhari is currently a Research Staff Member with IBM, Armonk, NY, USA, and a Manager of the Quantum Compiler Group. His research interests include quantum computing software, compilation, and architecture. Javadi Abhari received the Ph.D. degree in electrical engineering from Princeton University. Contact him at ali.javadi@ibm.com.

Nhung Hong Nguyen is currently a Ph.D. student with Linke Lab, University of Maryland. She was a Research Assistant with the Center for Quantum Technology, Singapore, working in satellite quantum key distribution. Her research focuses on digital quantum simulation, algorithms implementation, and error encoding on trapped ions. Nguyen received the B.S. degree in physics from Nanyang Technological University, Singapore, working on surface spectroscopy with neutral atoms. Contact her at nhunghng@umd.edu.

Cinthia Huerta Alderete is currently a Ph.D. student with the National Institute of Astrophysics, Optics and Electronics (INAOE), San Andrés Cholula, Mexico, currently on a research stay at the Joint Quantum Institute, University of Maryland. Her research is focused on, but not limited to, the simulation of paraparticle oscillators in a trapped-ion system. Aside from this topic, she had collaborated on a few projects based on the circuit implementation of different phenomena in quantum physics. Contact her at aldehuer@umd.edu.