Quantum Codesign

Teague Tomesh and Margaret Martonosi, *Princeton University, Princeton, NJ, 08544-5263, USA*

Codesign has been an integral part of computer architecture since the very first systems were brought online. From the early days of the field until now, end-user applications inevitably shape the design and capabilities of subsequent generations of hardware. Likewise, the characteristics and capabilities of new computational hardware and systems often impact the algorithms and software that run on them. Quantum computing (QC) is similarly reliant on codesign approaches, particularly now in its resource constrained early days. This article discusses what codesign means in a QC setting, gives examples of its value to QC, and proposes key attributes of QC codesign approaches going forward.

The development of quantum computers has rapidly accelerated over the course of the last 5 years. The capability of quantum systems—i.e., the number of qubits and the accuracy of gate operations—has increased dramatically. However, despite the significant progress made in this short period of time, today's quantum computers are still highly resource-limited and error-prone. This has limited the domain of solvable problems for near-term computers compared to the envisioned fault-tolerant (FT) systems capable of running arbitrarily long programs. Hardware-software codesign approaches offer the potential to efficiently and effectively achieve the best mappings of challenging applications onto constrained hardware.

QC and Resource Constraints: The term noisy intermediate-scale quantum (NISQ) was coined to refer to today's QC implementations, which are growing quickly in capabilities, but are still severely resource constrained.[1] At one end of the stack, quantum hardware continues to scale to greater and greater numbers of physical qubits, but the depth (i.e., operation count) of successfully executable programs is still limited by qubit coherence time and high operation error rates. Similarly, current hardware typically presents only very limited communication between qubits within a small neighborhood since the required swapping operation is dominated by expensive and error-prone entangling gates. Despite these considerable challenges, current work has shown 10× or more benefits in program success rates if the compiler is designed to make use of information about its target's gate error rates and connection topologies.

At the other end of the stack, quantum algorithms are intrinsically written to assume certain capabilities of the quantum hardware. Success lies in good matches between what the algorithm assumes, and what a given implementation can actually offer. As an extreme example, for Shor's algorithm to run on "interesting" numbers, it will require thousands of high-quality error-corrected qubits; this is far beyond any current implementation. On the other hand, as we will show here, for QC applications whose resource requirements are closer to current-day systems, codesign techniques can "bridge the gap" and ensure a successful mapping.

What does Codesign for QC Mean? Codesign refers to the flow of information between different hardware and software stack layers, in order to (in individual runs or over entire design cycles) improve the overall application execution and hardware design. The information flow might include: key hardware parameters, design specifications, and resource requirements up and down the stack. Codesign for QC is about incorporating this information into the techniques and system designs at every layer of the stack to make optimal use of limited resources (see Figure 1). There is a natural tension between the designer's desire for abstraction (i.e., information hiding) and codesign's push for information flow. This article discusses good options for striking a strategic balance between abstraction and information flow, in order to support both short-term and long-term goals for QC systems.

The state-of-the-art in terms of both quantum algorithms and hardware is rapidly evolving. This creates a situation where successfully mapping algorithms to hardware first requires answering a set of questions:

0272-1732 © 2021 IEEE
Digital Object Identifier 10.1109/MM.2021.3094461
Date of publication 2 July 2021; date of current version 14 September 2021.

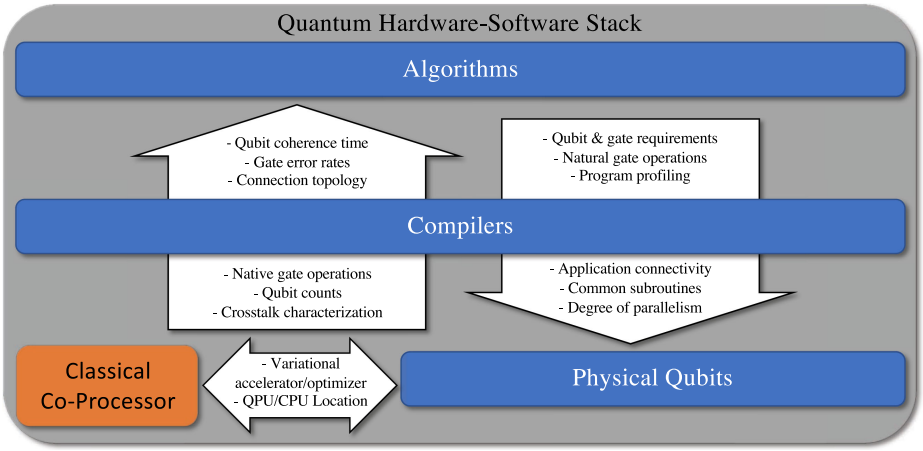

FIGURE 1. The free flow of information up and down the hardware-software stack is an integral part of codesign. The systems at each layer (algorithms, compilers, and hardware) face an inevitable tension between abstraction and complexity. Given the constrained nature of quantum resources, codesign is necessary to successfully map applications to hardware.

1) What resources does the application require?
2) What is the capacity and capability of the hardware?
3) How will the computation be divided between classical and quantum resources?

The answers to these questions constrain the space of solvable problems. They present challenges to the design and application of quantum systems, and taking a narrow-minded approach may hinder the effectiveness of those systems. Codesign is about considering these constraints at every layer of the stack and using the details available in the layers above and below to develop informed techniques to reach efficient and effective solutions.

WE SEE THE CODESIGN AND INTEGRATION OF THE ENTIRE STACK AS A VIABLE ROUTE TOWARD USEFUL QUANTUM COMPUTATION IN THE NISQ ERA AND BEYOND.

Every layer of the hardware-software stack contains certain parameters and information, which can be relevant to designers at higher and lower layers. Developing techniques and designs which exploit these parameters is the essence of codesign. Figure 1 shows the flow of information relevant to near-term NISQ systems. By mapping out the parameters which are relevant now, we can begin to understand what will be important to focus on in the near term, what is important in the long term, and what will remain relevant throughout. Some details, such as the connection topology between qubits, may be abstracted away once gate error rates become low enough. Other parameters, such as qubit count, may continue to be relevant throughout the NISQ and FT eras.

Concurrent with the incredible growth and activity in quantum computing, codesign techniques are already being developed that exploit these parameters. In this article, we provide an overview of the applications of codesign to the quantum computing stack. From the algorithms through the compiler down to the hardware, there are ample opportunities for significant efficiency gains by exploiting application and hardware specific knowledge. A general trend in classical computing is the growing levels of abstraction between layers of the stack. While this has benefited classical computing in myriad ways, the current state of QC requires the opposite approach. Exposing more detail, more parameters, between layers will allow system designers to better meet the constraints imposed by the NISQ technology. We see the codesign and integration of the entire stack as a viable route toward useful quantum computation in the NISQ era and beyond.

QC BACKGROUND

Quantum programs are expressed as quantum circuits, where time runs from left to right and each horizontal line designates a different qubit. A quantum circuit implements a given program by acting on the qubits with a set of different operations, also known

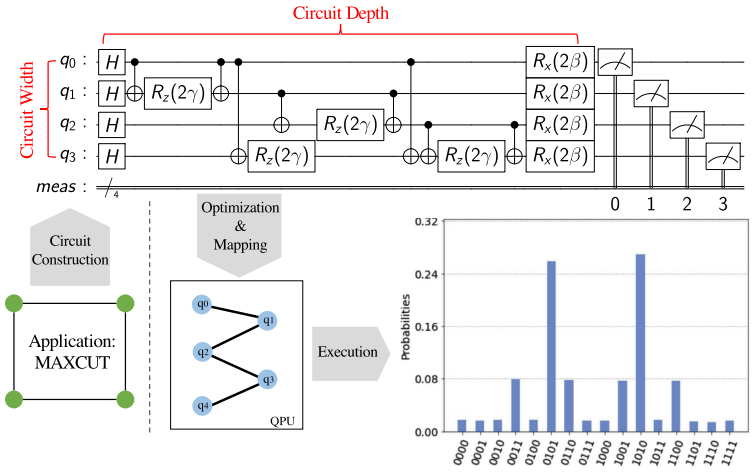

FIGURE 2. Example of QC execution flow.

as quantum gates. Figure 2 shows an example of a 4-qubit circuit containing *H*, R_z, R_x, and *CNOT* gates and measurements of all four qubits at the very end.

A typical model for QC execution begins with the programmer using a high-level programming language to construct a quantum circuit. In Figure 2 an example application, solving the MAXCUT problem on a 4-node graph, is used to illustrate the QC execution model. The compiler will then take this circuit and compile it down to a set of native gates supported by the hardware. In addition to the transpilation from high level to native gates, the quantum compiler will also perform optimizations to reduce the circuit depth and provide a mapping from the logical qubits to the physical qubits on the hardware. The mapping process may also require inserting swap operations to satisfy the hardware's connection topology. It is also important to note that this compilation process is usually performed prior to *every* program execution. This is critical for NISQ computing because the latest hardware calibration data must be taken into account to map the logical circuit to the physical qubits, which currently have the lowest noise and highest fidelity operations.

Once the program is compiled into an executable form, it is run *N* times on the hardware and the result of each run, called a shot, is recorded. The *N* shots form a distribution over the set of measured answers and the quality of the execution can be characterized by the success rate.

HOW CODESIGN APPROACHES HAVE INFLUENCED QC HARDWARE DESIGN

The theory of quantum computing, manifesting as algorithms and applications, has had a long head start over the hardware. As a result, the design of QC hardware has been shaped by applications that were designed even before the first, fully integrated systems were built. The feedback loop between hardware and software was finally completed when the first quantum systems were made available to application designers. The NISQ algorithms designed for these machines have opened the door to further development of application-specific hardware tailored to near-term applications.

The impact of early quantum algorithms and applications can be seen in the design of current hardware. Error-correcting codes are a critical class of quantum applications that have shaped the qubit connectivity of many of today's quantum computers. Nearest-neighbor 2-D lattices are a natural connection topology for superconducting hardware targeting surface-codes whereas sparser connectivities are preferable for heavy-hexagon or heavy-square codes.[2] Having a range of codes to choose from allows hardware designers to make tradeoffs in design complexity between sparser and more densely connected qubits. Additionally, the operations that are used to express algorithms have influenced the gate sets supported by hardware. A recent example is the mid-circuit measurement operation, used in textbook quantum algorithms like teleportation and error-correction to provide control-flow within programs, which is now available in commercial systems.[3] Supporting new operations in hardware can also affect the way algorithms are implemented in software. IBM recently demonstrated the possible resource tradeoffs by implementing an algorithm using either 12-qubits or only 2-qubit plus mid-circuit measurement.[4]

The development of near-term applications and algorithms has also shaped the design of quantum

hardware. Hybrid quantum-classical variational algorithms were developed in response to the limited capabilities of NISQ devices. This new class of applications raises interesting architectural questions such as: where should the classical coprocessor be located, and how will it interface with the quantum processor? Ideally the computational resources should be placed as close together as possible to minimize the time that the data are in flight since the variational algorithms contain inner loops, which may execute many thousands of times. However, in the case of superconducting quantum computers, this would require that the classical coprocessor function at temperatures near absolute zero while dissipating minimal heat to minimize the chance of qubit decoherence. Research efforts to solve these issues are already under way, including a new cryogenic control chip "Horseridge" announced by Intel.[5] Variational algorithms are also distinguished by their use of parameterized quantum gates, which are the knobs that the classical optimizer uses to traverse the cost function landscape. These parameters are angles from $[0, 2\pi)$, which are input to different rotation gates within the quantum circuit. This introduces a dilemma since quantum gates are usually calibrated only once or twice per day and every value in the interval cannot be tuned as an individual gate in the hardware. Therefore, architectures which natively support continuously tunable gates or are able to provide high fidelity approximations using a small subset of pretuned gates will be better suited to executing these types of algorithms. These examples show the advantage that can be gained by designing the capabilities of the hardware to support specific applications like variational algorithms.

APPLICATION-SPECIFIC ARCHITECTURES ARE A PROMISING STRATEGY FOR DEALING WITH THE LIMITED QUANTUM RESOURCES OF THE NISQ ERA.

Application-specific architectures are a promising strategy for dealing with the limited quantum resources of the NISQ era. Similar to the use of GPUs in classical computing, quantum chips may be designed as dedicated accelerators, which provide additional computational resources to process running on a classical CPU. This is already the role played by the QPU in today's variational algorithms, and could provide a path to quantum advantage. Such application-specific designs are already being studied in the case of superconducting architectures.[6] The design of accelerator based quantum computers can take place alongside efforts toward reaching fault-tolerance, and simultaneously exploring both design spaces would also be beneficial for both.

HOW CODESIGN HAS GUIDED QC ALGORITHM DESIGN

The design of quantum algorithms has already been deeply influenced by the capabilities of NISQ hardware.

The rise in popularity of variational algorithms is a direct result of the limited coherence times and low gate fidelities of current quantum computers. These algorithms make a tradeoff between qubit coherence requirements and the number of circuit executions. Instead of running a single, deep circuit to find a solution for a target application, the same problem can be solved by running many shallow circuits. Variational algorithms also utilize classical computational resources running an optimization algorithm in an outer loop. In this paradigm, the quantum processor functions more like an accelerator are used to compute the classically difficult objective function. The structure of variational algorithms introduces interesting questions concerning circuit structure, compilation optimizations, and data communication between computational resources.

The circuit or "ansatz" structure of variational algorithms is a clear example of codesign. Some of the first proposed ansatzes were labelled as hardware-efficient because they were structured to perfectly match the connectivity of the underlying hardware. This type of design is advantageous because it does not require the insertion of any swap operations, however, the expressivity and trainability of hardware-efficient ansatzes can be limited. These issues can be alleviated by taking the codesign process one step further and incorporating application-specific information into the design of the variational ansatz to better match the structure of the problem. These parameterized circuits are easier to train at the cost of requiring a few more swap gates. Designing optimal variational ansatzes with respect to the application and hardware topology is currently an open and important area of future research.

Hamiltonian simulation is an example of an important quantum application, which has benefited from the codesign of algorithms with respect to the capabilities of NISQ hardware. Hamiltonian simulation was

one of the first known applications for quantum computers with an exponential speedup. However, until very recently, the best quantum algorithms for this task required a prohibitive number of quantum gates such that any implementation would require error correction. New variational algorithms for Hamiltonian simulation have been developed, which lower these gate requirements by orders of magnitude.[7] These new algorithms reduce circuit depth by changing the relationship between the quantum and classical computational resources. Rather than carrying out the Hamiltonian simulation with a single run of a deep, fixed quantum circuit, we can allow the classical computer to shoulder some of the computation and repeatedly execute many different shallow quantum circuits.

HOW CODESIGN HAS INFLUENCED QC COMPILER DESIGN

Compilers, situated between the application and hardware layers are in a prime position to take full advantage of codesign techniques. Quantum compilers have focused their attention on developing techniques to mitigate the limitations of hardware while accommodating the demands of algorithms. Because resources such as qubit and gate counts are so limited, quantum compilers must exploit information from the algorithms above and the hardware below to ensure good performance. In fact, this necessity is the biggest difference between classical and quantum compilation. The effectiveness of classical compilers is usually measured by the speed up they provide, but for quantum compilers performance can be measured by the probability of whether a program runs at all. The critical reliance of quantum programs on good compilers points to the necessity of application and hardware specific optimizations.

Application level information can provide the compiler with context so it can be more aggressive and strategic in its optimizations. As an example, the structure of variational algorithms is quite different from other typical algorithms. Variational algorithms utilize parameterized quantum circuits, which are executed many times with different parameter values but the overall layout of the circuits remains fixed between runs. The compiler can exploit this information to reduce overall runtime by storing the fixed parts of the circuit across executions and only updating the portions, which contain the parameterized gates.[8] Application-specific information can also reveal symmetries and opportunities for gate cancellation that would otherwise be invisible to optimizations performed at lower levels of the stack. Applications like Hamiltonian simulation contain a large amount contextual information because they are concerned with simulating some physical system. This knowledge can be used to optimize the program's order of execution to mitigate certain types of errors—analogous to how the order of operations for floating point computations can impact accuracy.[9]

In the current NISQ era, mitigating the effects of noise has become a major focus for compiler development. Noise levels vary not only between QPUs, but also across qubits and time within the same device. Noise-aware compiler optimizations tailored to specific hardware and utilizing the most up-to-date calibration data have increased program success rates

TABLE 1. Summary of the codesign parameters and examples covered in this article. (Top) Codesign parameters. (Bottom) Codesign examples (the numbers in the parenthesis show which parameters are relevant to that example).

Hardware Characteristics	1. Coherence times	2. Gate fidelities	3. Native operations	4. Qubit connectivity
Program Profiling	5. Resource requirements	6. Application symmetries	7. Hybrid computation	8. Circuit connectivity

Variational Algorithms: 100x reduction in gate counts [7] (1, 2)	Ansatz design (1, 3, 4, 6)	Noise-aware compilation: 28x increase in success rate [10] (1, 2, 3, 4)
Program Ordering [9] (5, 6)	Connectivity Design [2], [16] (6, 8)	Mid-circuit Measurement: 6x reduction in qubit cost [3], [4] (5)
Pulse-level optimization [14], [15] (1, 3, 6)	Estimating correct program output [12], [13] (1, 2, 6)	Readout classifiers [11] (1, 2, 3)
Partial compilation: 3x runtime speedup [8] (5, 6, 7)	Co-locating CPU and QPU [5] (5, 7)	

and are now a standard part of IBM's Qiskit quantum software package.[10] Exposing pulse-level access to higher layers in the stack has led to the development of readout classifiers, trained on low-level hardware data, which are better able to distinguish between $|0\rangle$ and $|1\rangle$ states.[11] Noise-aware approaches have also been adapted to methodologies, which estimate the true output of a quantum circuit by incorporating the error characteristics and noisy outputs of multiple quantum devices.[12] Other works exploit the reversibility of quantum circuits for the purposes of output verification and debugging.[13] Furthermore, compilers which operate at the pulse level, converting applications (specified at the circuit level) directly into the native pulse operations of the hardware, have been shown to reduce program runtime and increase fidelity.[14,15]

Even in the FT era, the compiler will still play a critical role in mapping particular applications to specific architectures. Recent work from Intel studied the simultaneous impact of qubit encoding (an algorithmic level consideration) and hardware connection topology on qubit and gate counts for the specific application of Hamiltonian simulation.[16] They found that different encodings can result in different communication requirements between qubits which suggests that, given a particular qubit layout, one encoding may incur fewer swaps than another. Looking at different hardware connectivities, they found that the performance gap between square grid and ladder topologies was not as great as the gap between linear and ladder connectivities. This sort of insight may be beneficial to hardware designers since it is a more difficult task to fabricate full, 2-D square grids versus a ladder layout.

CODESIGN APPROACHES AND BEST PRACTICES

The examples in the preceding sections have demonstrated how hardware-software information flow is already influencing QC. Codesign relies on such information flow, but on the other hand, abstraction remains valuable too. So it is essential to identify the limited and specific information that offers the most leverage up and down the stack. These parameters and characteristics will form the key for QC codesign, while also allowing other details to be abstracted away.

Table 1 summarizes attributes that were key to the codesign examples previously covered. We saw how many of the design choices made at the algorithmic level were influenced by low-level hardware characteristics. Applications can make more efficient use of quantum resources by incorporating knowledge of the underlying connection topology and native gate set. Taking the opposite view, high-level program profiling can lead to more effective and efficient hardware designs. Between these layers, the compiler can take full advantage of both the low- and high-level information to map applications to hardware in the most efficient way possible.

Finally, we note how the set of key codesign parameters convey information that is portable across quantum architectures. QC implementations come in a variety of forms between superconducting circuits, photonic chips, and ion traps. Although specific details may vary across architectures, certain characteristics remain relevant: how noisy are the qubits? Which operations are supported? How error-prone are they? What is the connection topology? Different QC implementations will have different answers to these questions and will therefore be better suited for certain applications over others. Codesign combining hardware characterization and program profiling provides a principled approach to quantum computation across architectures.

AS TIME PROGRESSES AND THE QUALITY OF QUANTUM HARDWARE IMPROVES, WE MAY FIND THAT THE DESIGN COMPLEXITY INTRODUCED BY EXPOSING CERTAIN LOW-LEVEL PARAMETERS IS NO LONGER BENEFICIAL. THESE DETAILS CAN BE ABSTRACTED AWAY TO SIMPLIFY DESIGN, BUT IT IS LIKELY THAT SOME PARAMETERS WILL MAINTAIN THEIR RELEVANCE.

HOW WILL CODESIGN EVOLVE AS QC EVOLVES?

In today's world of classical computing, only a small subset of people need to actively consider the function of individual transistors when designing new systems. This is in contrast to the current state of quantum computing where successful program execution can rely on the effective utilization of each qubit. In the near-term, abstraction is a luxury that quantum systems cannot afford. Utilizing low-level details will be critical for the efficient usage of limited quantum resources, and as the quality of quantum hardware improves the benefits of

abstraction will increase. This creates tension between system designs tailored to the near term and those built around the capabilities of long-term FT architectures. One may ask: is there any point to considering near-term applications and techniques if the hardware they are built on will be irrelevant in the future? Classical architects caught in the mid of Moore's Law faced similar questions such as these, but even then it was clear that only designing systems for hardware which would be built decades in the future was a nonviable strategy. The key issue for QC is understanding and mapping out the timeline to fault-tolerance. No one can say for certain when, or if, fault-tolerant error-corrected quantum computers will arrive, but the work done today on the codesign of current applications and hardware will help to shed light on this question and provide a path forward.

Without interesting near-term applications, there are no examples between small toy problems and large FT applications to serve as a guide for hardware designers. The QC systems which are being built today are nearing the boundaries of classical computation for certain problems, but they only possess a fraction of the resources required to run Shor's algorithm on 2048 bit numbers. Harnessing the capabilities, limited though they may be, of near-term devices and applying them to useful applications requires a full-stack, codesign approach. This will help to inform the design of subsequent generations of QC hardware and provide insights into the types of problems which are suited to quantum computation.

FRAMING THE DESIGN OF NEAR-TERM QUANTUM COMPUTERS AROUND THE ARCHITECTURE OF ACCELERATORS IS A PROMISING PATH TOWARD NEAR-TERM USEFULNESS. THIS DOES NOT REPLACE THE QUEST FOR FAULT-TOLERANCE. INSTEAD, THESE TWO PATHS ARE MUTUALLY BENEFICIAL TO ONE ANOTHER.

As time progresses and the quality of quantum hardware improves, we may find that the design complexity introduced by exposing certain low-level parameters is no longer beneficial. These details can be abstracted away to simplify design, but it is likely that some parameters will maintain their relevance. For example, although the abstraction between physical and logical qubits will become useful after error-correction is achieved, qubits are likely to remain expensive resources and this must be considered in application and compiler design. Therefore, techniques developed today which optimize qubit usage will remain relevant far into the future. Qubit count is just one example of a low-level design parameter, which may be relevant across timescales. Quantum computer architecture is still a relatively new field and there is much work to be done in mapping out the codesign parameter space and incorporating that knowledge into near- and long-term architectures.

CONCLUSION

Looking forward there are plentiful opportunities for codesign of the quantum computing stack. Furthermore, the performance and efficiency gained through such codesign will be a prerequisite for achieving quantum advantage in the NISQ era. The key to effective codesign is the open flow of information throughout the layers of the hardware and software stacks. In the context of QC, this will require breaking abstractions to allow for application- and hardware-specific optimizations.

Framing the design of near-term quantum computers around the architecture of accelerators is a promising path toward near-term usefulness. This does not replace the quest for fault-tolerance. Instead, these two paths are mutually beneficial to one another. Application-specific accelerators are much more likely to solve useful problems before their more general counterparts, which will allow the quantum ecosystem to grow sooner and faster. A growing ecosystem will benefit the error correction research track and the subsequent reductions in gate errors and increases in qubit counts will also lead to better accelerators. Codesign is crucial to this task and there is still much interesting work left to be done.

Much of the work done now will still be relevant in the fault-tolerant era and shape the path toward that goal. A suite of quantum benchmarks including near-term as well as the smaller subroutines for far-term algorithms would be useful for analyzing the performance of current NISQ hardware and informing the design of subsequent generations. As the development of both quantum hardware and algorithms continues, coherence times will increase, gate costs will decrease, and the number of qubits will multiply, but the same architectural questions will remain. What is the connection topology between qubits and how do they match the communication requirements of

the algorithm? What are the natural operations used to express the algorithm and how close are they to the native gates supported by the hardware? Current research is pursuing answers to these questions, which will help to shape the design of next generation quantum systems. In fact, it is likely they will need to be revisited time and time again as we seek to build better ever more powerful quantum computers.

REFERENCES

1. J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, pp. 79, 2018.
2. C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, "Topological and subsystem codes on low-degree graphs with flag qubits," *Phys. Rev. X.*, vol. 10, no. 1, 2020, Art. no. 011022.
3. S. K. Moore, "Honeywell's ion trap quantum computer makes big leap," *IEEE Spectr. Technol., Eng., Sci. News*, Mar. 3, 2020. [Online]. Available: spectrum.ieee.org/tech-talk/computing/hardware/honeywells-ion-trap-quantum-computer-makes-big-leap
4. P. Nation and B. Johnson, "How to measure and reset a qubit in the middle of a circuit execution," *IBM Res. Blog*, Feb. 11, 2021.
5. Intel PR, "Intel debuts 2nd-Gen horse ridge cryogenic quantum control chip," *Intel Newsroom*, Dec. 3, 2020.
6. G. Li, Y. Ding, and Y. Xie, "Towards efficient superconducting quantum processor architecture design," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 1031–1045.
7. Y.-X. Yao *et al.*, "Adaptive variational quantum dynamics simulations," *PRX Quantum*, Amer. Physi. Soc., vol. 2, no. 3, p. 030307, Jul. 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PRXQuantum.2.030307
8. P. Gokhale *et al.*, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines." In *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 266–278.
9. K. Gui *et al.*, "Circuit optimization for simulations of quantum systems," *Bulletin Amer. Physi. Soc.*, APS, 2021.
10. P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Architecting noisy intermediate-scale quantum computers: A real-system study," *IEEE Micro*, vol. 40, no. 3, pp. 73–80, May/Jun. 2020.
11. T. Patel and D. Tiwari, "DisQ: A novel quantum output state classification method on IBM quantum computers using openpulse," in *Proc. 39th Int. Conf. Comput.-Aided Des.*, 2020, pp. 1–9.
12. T. Patel and D. Tiwari, "Veritas: Accurately estimating the correct output on noisy intermediate-scale quantum computers," in *Proc. SC20: Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 188–203.
13. T. Patel and D. Tiwari, "Qraft: Reverse your quantum circuit and know the correct program output," In *Proc. 26th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2021, pp. 443–455.
14. Y. Shi *et al.*, "Optimized compilation of aggregated instructions for realistic quantum computers," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 1031–1044, Harvard
15. J. Cheng, H. Deng, and X. Qia, "Accqoc: Accelerating quantum optimal control based pulse generation," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 543–555.
16. N. P. D. Sawaya, G. G. Guerreschi, and A. Holmes, "On connectivity-dependent resource requirements for digital quantum simulation of d-level particles," in *Proc. IEEE Int. Conf. Quantum Comput. Eng.*, 2020, pp. 180–190.

TEAGUE TOMESH is currently working toward a Ph.D. degree in the Computer Science Department, Princeton University. His research interests include closing the gap between quantum hardware and practical applications by applying computer architecture and compilation techniques. He is a member of IEEE and the Association for Computing Machinery (ACM). Contact him at ttomesh@princeton.edu.

MARGARET MARTONOSI is the Hugh Trumbull Adams 35 Professor of Computer Science with Princeton University, Princeton, NJ, USA. Her research interests include computer architecture and hardware-software interface issues in both classical and quantum systems. Martonosi received a Ph.D. degree in electrical engineering from Stanford University. She is a Fellow of IEEE and the Association for Computing Machinery. Contact her at mrm@princeton.edu.