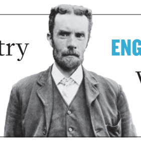
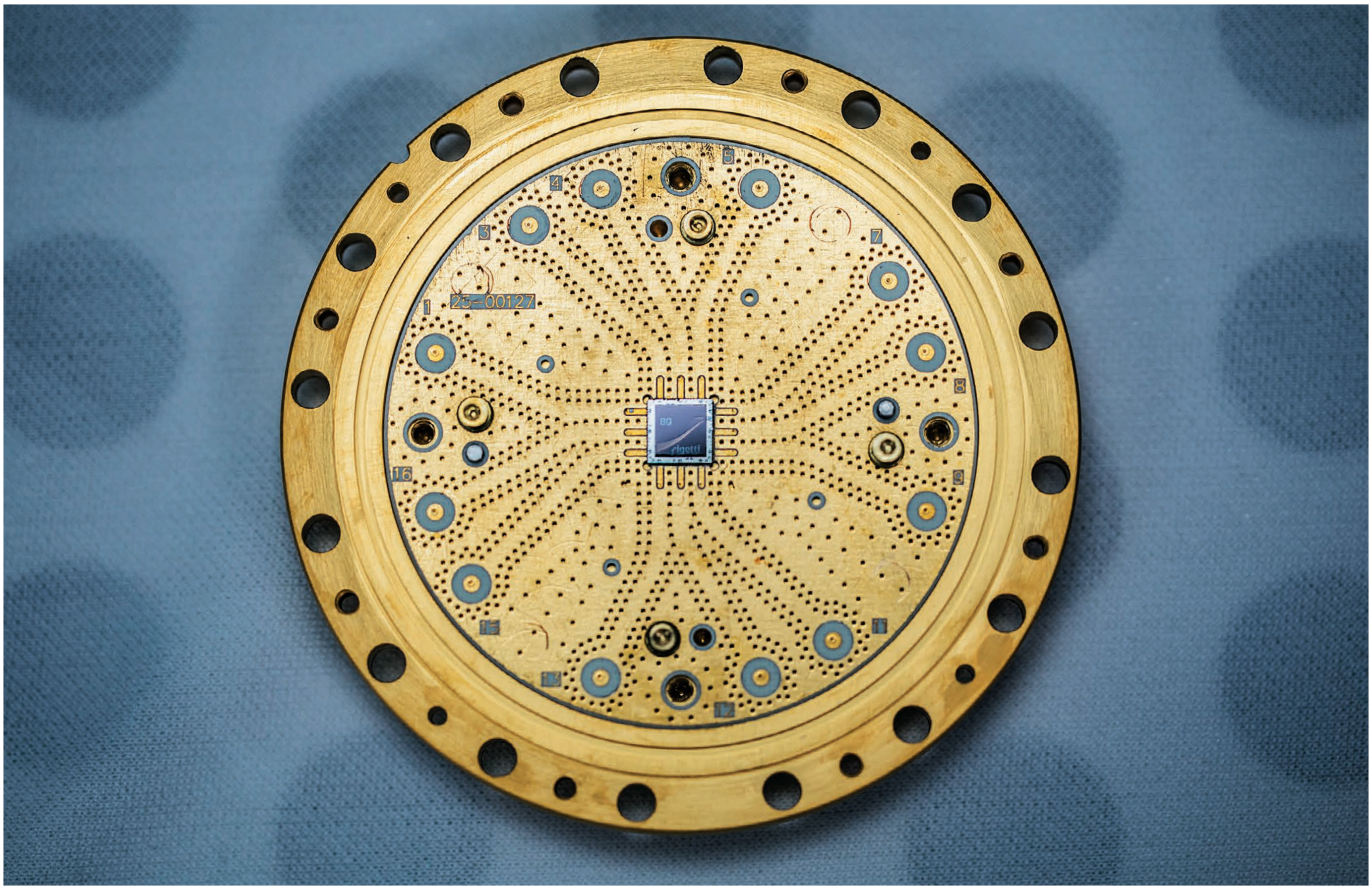


RIGETTI COMPUTING


An 8-qubit quantum processor built by Rigetti Computing.

First quantum computers need smart software

Early devices must solve real-world problems, urge **Will Zeng** and colleagues.

The world is about to have its first quantum computers. The complexity and power of quantum hardware, such as ion traps and superconducting qubits, are scaling up. Investment is flooding in: from governments, through the billion-dollar European Quantum Technology Flagship Program, for example; from companies, including Google, IBM, Intel and Microsoft; and from venture-capital firms, which have funded start-ups. One such is ours, Rigetti Computing, which in June opened the first dedicated facility for making quantum integrated circuits: Fab-1 in Fremont, California. The vision is that commercial quantum-computing services will one day solve problems that used to be unimaginably hard, in areas from molecular design and machine learning to cybersecurity and logistics.[1]

The problem is how best to program these devices. The stakes are high — get this wrong and we will have experiments that nobody can use instead of technology that can change the world.

We outline three developments that are needed over the next five years to ensure that the first quantum computers can be programmed to perform useful tasks. First, developers must think in terms of 'hybrid' approaches that combine classical and ▶

▶ quantum processors. For example, at Rigetti we have developed an interface called Quil[2], which includes a set of basic instructions for managing quantum gates and classical processors and for reading and writing to and from shared memory. Second, researchers and engineers must build and use open-source software for quantum-computing applications. Third, scientists need to establish a quantum-programming community to nurture an ecosystem of software. This community must be interdisciplinary, inclusive and focused on applications.

HYBRID SYSTEMS

Today's quantum programming differs from much previous theoretical work on algorithms; it is becoming more and more practical.

Theoretical computer scientists have been developing potential algorithms for imagined quantum computers since the 1990s. Mathematician Peter Shor's famous code for breaking encryptions was one of the first; many more are listed in the Quantum Algorithm Zoo from the US National Institute of Standards and Technology (see go.nature.com/2inmtco). These algorithms are generally designed for big, noiseless quantum computers, which are unlike the devices that will be available within the next five years. These will have tens to thousands, not millions, of qubits, with little redundancy to correct for internal errors. They will calculate a limited range of things in a noisy way. For example, they will not be able to use Shor's algorithm to find the prime factors of large numbers. So their use must be targeted: they will not always beat conventional computers.

These limitations can be overcome by building quantum processors as 'accelerators' to boost the performance of conventional computers. A classical computer might, for example, optimize operations to compensate for noise in the quantum processor, or aggregate answers from sequences of short quantum programs. Such hybrid programming has been demonstrated in quantum chemistry[3] and in optimization[4]. Algorithms that run on small, superconducting quantum processors have performed steps in calculating the ground states of materials and molecular systems, for example[5,6]. Another algorithm has solved constrained optimization problems, which are common in areas such as machine learning, logistics and scheduling[4].

We've found, however, that it can be hard to predict the performance of hybrid algorithms. For example, the quality of the quantum subroutine in hybrid algorithms for chemistry can vary greatly depending on the system that is being simulated and the mathematical tricks used. So hybrid quantum-computing algorithms need to be studied empirically, as they are for machine learning. The way to find out how a system works is to build it, see what it does and back up any rules of thumb with mathematics later. This work will begin in earnest once the first quantum computers are available, and it will accelerate fast.

To reach this stage, researchers must change their mindsets, and this could be hard. We will find that some past work has little utility. We've all seen talks on quantum algorithms whose complexities are peppered with huge exponents, meaning that they could take millions of years to complete. For the coming devices, such codes are so impractical as to be useless.

Quantum programmers must care about practical details such as noise models and exact counts of logic gates. They will have to decide which qubits in the computer to use and how to deal with ranges of operational fidelities and low-level precisions that are foreign to most modern programmers. But the gain will be worth the pain.

"We will find that some past work has little utility."

In turn, hardware designers need to be responsive to the choices and preferences of quantum programmers, so that their technology can become more useful.

OPEN SOFTWARE

Different classical computers behave similarly enough to enable software written for one to run on others. Early quantum computers will have their own nuances, and software for them will need to be bespoke. When each operation and instruction matters, generalized solutions need to be optimized, and software and hardware designed concurrently. Algorithms must be discovered numerically rather than algebraically, and developed using simulators and software rather than pens and paper.

Innovative digital tools are needed for developing and testing algorithms, writing software and programming the devices. Quantum programmers should keep an eye on the underlying physics, so that they are aware of different types of noise in sequences of pulses, for example. Performance benchmarks, such as a suite of standard molecules to simulate, are also necessary.

Differences between quantum and classical programming begin at the instruction level. Classical computers use Boolean logic — with basic operations such as AND, NOT, OR. Operations in quantum computations, such as multiplying tensors and matrices, are much more complex and result in unusual behaviour. For example, quantum information cannot be cloned exactly between processor registers; and reading the state of a quantum register alters the information stored in it.[7] Hybrid software needs to handle all these behaviours simply enough for programmers to be able to code easily. The result will be a new programming paradigm, as object-oriented, probabilistic and distributed programming once were.

Quantum programmers must decide which aspects of the system are essential for them to consider and which they can skim over in practice. For example, executing a program on superconducting quantum processors requires instructions to be translated many times. Control and readout instructions are converted from digital to analog to quantum to analog to digital as they go from the control hardware to the qubits and back. Programmers don't want to have to deal with all the microwave engineering and physics, but they need to be aware of how these processes affect noise or the time it takes to run the code. They need tools to work directly with the devices, so that they can understand and exploit the trade-offs.

Easy programming interfaces are crucial to making quantum computers widely usable; examples include Quil and OpenQASM[8] from IBM. More sophisticated options still need to be added, such as optimizations for specific types of processors. Higher-level languages for writing and compiling quantum programs also need to be developed.

It is important that all these tools are open source. Such a model was not available at the dawn of digital computing, but its power to speed innovation, as with Linux in the early days of the web, is essential for the quantum-programming community to grow quickly. We have made a start with our quantum-programming toolkit, Forest, which is written in Python, open source and accessible to anyone. It joins an exciting early ecosystem — much of it open source — developed by different academic and industrial research groups. Other examples are LIQUi|> (embedded in F#), Scaffold (C++), Quipper (Haskell), QGL (Python), ProjectQ (Python), QCL, QuIDDPro and Chisel-Q (Scala). Researchers must resist pressure to standardize tools prematurely or keep the high-level, exploratory parts of the programming stack proprietary.

BUILD A COMMUNITY

A new breed of quantum programmer is needed to study and implement quantum software — with a skillset between that of a quantum information theorist and a software engineer. Such programmers will understand how quantum devices operate well enough to instruct them and minimize problems. They will be able to build usable software and will have a deep knowledge of the mathematics of quantum algorithms and computation. Experts from fields in which the software will be applied must be closely involved if the code is to be

Inside the clean room at Rigetti Computing's Fab-1 facility in Fremont, California.

truly useful. For example, chemists such as Alán Aspuru-Guzik at Harvard University in Cambridge, Massachusetts, drove interest in using hybrid algorithms in quantum-chemistry calculations. Researchers in other fields, especially in machine learning and optimization, should get on board.

Advanced kinds of education are needed to train this new breed. Several centres are well positioned to draw together the interdisciplinary skills and tools needed to offer degrees in quantum-computer engineering: the Institute for Quantum Computing at the University of Waterloo in Canada, the Institute for Quantum Information and Matter at the California Institute of Technology in Pasadena, the quantum-engineering doctoral training centres in the United Kingdom, and QuSoft, the Dutch research centre for quantum software in Amsterdam. At Rigetti we have started a Junior Quantum Engineer programme for bachelor's degree students, which includes training in quantum programming. We have partnered with the Quantum Machine Learning accelerator at the Creative Destruction Lab (a technology-transfer centre that fosters start-ups) at the University of Toronto, Canada, to provide access to and support for Forest and other programming tools.

Early-career quantum programmers have tremendous opportunities to become leaders of a transformational field. But they need support. Their supervisors must recognize that work on an open-source software project might delay their next pure research paper. They need industrial internships to gain a broader practical perspective. And they need institutional backing to work between the fields of software engineering and quantum physics.

NEXT STEPS

It is crucial that research on quantum-computing algorithms is tied more closely to research on the software that's used to implement them.

First, funders should insist that theoretical work is implemented in software and, as much as possible, tested on hardware. Second, algorithm researchers must be explicit about the architecture they are targeting. They must show evidence of how algorithms will be practically implemented on different noisy systems. Third, funders and journal editors must establish standard ways to assess algorithm performance and resource requirements. This will enable hardware and software to improve together, and will sift out the most viable algorithms more quickly. Open-source tools should be used wherever possible, and publications should encourage the publication of code alongside results.

Finally, the quantum-computing community must prioritize engagement with experts in areas such as simulation and machine learning. Quantum and classical programmers must collaborate more. We call on every current and aspiring quantum-algorithm researcher to present their work at a classical conference at least once in the next year. It falls to us to expand the community that will realize the incredible potential of quantum computing. ■ SEE INSIGHT P. 171

Will Zeng *is director of quantum cloud services,* **Blake Johnson** *is director of quantum engineering,* **Robert Smith** *is senior software engineer,* **Nick Rubin** *is senior applications researcher,* **Matt Reagor** *is director of quantum engineering,* **Colm Ryan** *is principal quantum engineer and* **Chad Rigetti** *is chief executive at Rigetti Computing in Berkeley, California, USA.*
email: will@rigetti.com

1. Mohseni, M. *et al. Nature* **543,** 171–174 (2017).
2. Smith, R. S., Curtis, M. J. & Zeng, W. J. Preprint at https://arxiv.org/abs/1608.03355 (2016).
3. Rubin, N. Preprint at https://arxiv.org/abs/1610.06910 (2016).
4. Farhi, E., Goldstone, J. & Gutmann, S. Preprint at https://arxiv.org/abs/1411.4028 (2014).
5. Kandala, A. Preprint at https://arxiv.org/abs/1704.05018 (2017).
6. O'Malley, P. J. J. *et al. Phys. Rev.* X, **6,** 031007 (2016).
7. Hayes, B. *Am. Sci.* **102,** 22–25 (2014).
8. Cross, A. W., Bishop, L. S., Smolin, J. A. & Gambetta, J. M. Preprint at https://arxiv.org/abs/1707.03429 (2017).

The authors declare competing financial interests; see go.nature.com/2f7lrpk