Statistical Assertions for Validating Patterns and Finding Bugs in Quantum Programs

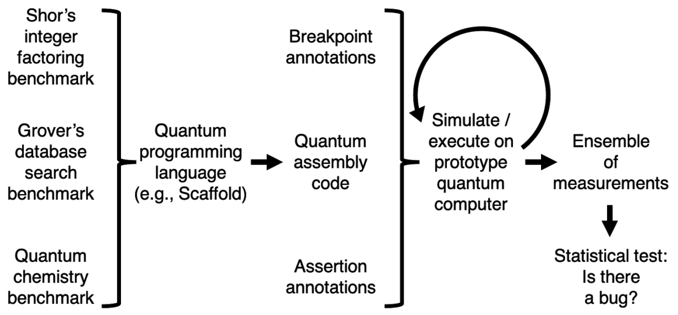
Yipeng Huang and Margaret Martonosi, Princeton University

In classical computing, debugging programs is one of the most time-consuming tasks in software development. Successful debugging relies on software development tools and also on the experience of the programmer. In quantum computing (QC), researchers foresee debugging to be an even greater challenge. In our ISCA '19 paper[1], we present approaches that represent some of the first-ever steps towards debugging tools for QC software and hardware systems. The debugging tools we propose are based on statistical tests, with a goal of aiding programmers in building correct quantum programs for near-term quantum computers. They can also be used by hardware designers to assess correct operation. Furthermore, we study a broad cross section of quantum algorithms, in order to identify recurring quantum program patterns and anti-patterns. These bugs and program patterns represent common and therefore important themes in QC debugging. They can guide the programmer to know where to put our proposed assertions for effective debugging.

Over roughly the past two decades, researchers have discovered hundreds of QC algorithms covering a range of applications in chemistry, optimization, search, and cryptography. Thus far, however, almost all of these algorithms only exist as abstract equations and specifications, and therefore may not have been fully checked for correctness. For some, lucky readers may find concrete quantum circuit diagrams among those specifications. A few dozen have actually been fleshed-out as program code in open-source repositories. Some of those implementations are likely correct, but almost none have been deeply debugged until now. Quantum programming is in its infancy.

The QC research landscape has changed rapidly. In the past handful of years, researchers have built the first prototype quantum computers capable of running small quantum programs. Notably, IBM has made simulators and small-scale quantum computers available for the public to run code and see results. *With this burgeoning interest in quantum computing experimentation, a new and urgent challenge lies in understanding what are quantum program bugs, and in helping experienced and novice quantum programmers translate those abstract algorithms into correctly functioning quantum program code. Our ISCA '19 work makes the community's first steps towards meeting this challenge.*

[1] Yipeng Huang and Margaret Martonosi. 2019. Statistical Assertions for Validating Patterns and Finding Bugs in Quantum Programs. In The 46th Annual International Symposium on Computer Architecture (ISCA '19), June 22–26, 2019, Phoenix, AZ, USA. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3307650.3322213


Shor's integer factoring benchmark
Grover's database search benchmark
Quantum chemistry benchmark
Quantum programming language (e.g., Scaffold)
Breakpoint annotations
Quantum assembly code
Assertion annotations
Simulate / execute on prototype quantum computer
Ensemble of measurements
Statistical test: Is there a bug?

We started this project by studying a wide array of quantum program benchmark implementations in several quantum programming languages. The ISCA '19 paper covered important exemplar algorithms such as quantum chemistry, Grover's database search, and Shor's integer factorization. We also analyzed several programs not treated in our paper such as optimization using a hybrid quantum-classical algorithm, and algorithms that rely on adiabatic evolution of quantum states. The goal was to capture a wide variety of quantum algorithm primitives—algorithm kernels that underlie the functionality of these algorithms and that are widely used across many QC algorithms. These algorithm primitives include variational optimization, amplitude amplification, and quantum Fourier transform. The implementations we studied span several open-source QC programming languages such as Scaffold, Microsoft's ProjectQ, IBM's Qiskit, and Google's Cirq. The goal was to have a broad understanding of how programmers might make mistakes in writing QC programs, and also to understand how different programming language features aid writing correct code. This is akin to prior papers on programming design patterns and parallel programming patterns.

In the process of building and analyzing the quantum algorithm benchmarks, our paper identifies and highlights three key difficulties in debugging quantum programs.

Challenge #1: Getting quantum debugging information is not straightforward as in classical debugging. The first difficulty is that programmers cannot easily examine the values of variables of a QC program, while the program is running. Inspecting quantum variables involves measuring and "collapsing" the delicate quantum states inside quantum computers. Once a quantum state is collapsed, any observations would not be a complete description of the state of the program. This limitation precludes the "printf" debugging approach commonly used by programmers with classical programs.

	Quantum algorithm primitives	**Benchmark algorithms / problems**	**Exhibited program patterns / assertions for debugging**
Algorithms for near-term, noisy, intermediate-scale quantum computing era	Entanglement protocols for quantum networking	Superdense coding; Quantum teleportation	Entanglement precondition assertions on initial conditions
	Hybrid quantum-classical variational algorithms	Quantum approximate optimization; Variational quantum eigensolver	Iterative algorithm progress checks
Algorithms for future, fault-tolerant, large-scale quantum computing era	Adiabatic evolution of quantum states	Ising spin chain model	Iterative algorithm progress checks
	Amplitude amplification	Grover's database search	Postcondition assertions that search results satisfy criteria
	Quantum Fourier transform	Phase estimation; Period finding; Shor's integer factoring	Postcondition assertions on correct deallocation of ancillary qubits; Modularity; Numerical data types

Our paper addresses Challenge #1 by finding ways to debug quantum programs using only the information about the collapsed quantum states (see figure on first page). We consider debugging programs in both simulated and real-system settings. In both settings, we introduce program instructions that tell the simulator or quantum computer to stop operation and measure quantum states early. The toolchain then uses multiple runs of the quantum program in order to find the probability distribution of the measurement outcomes from the quantum algorithm. In simulated settings, these probability distributions are tracked as internal data structures as the simulation progresses. In real-system QC runs, debugging occurs by running many "shots" (trials) on real QC prototypes, with measurements occurring at different assertion points on different runs in order to gather the data points for the distributions.

Challenge #2: Interpreting whether the measurements are correct is not an easy task for programmers. Even when observations or simulations are available, quantum states are in general high-dimensional and difficult to interpret. This limits their usefulness for programmers to debug misbehaving quantum programs; our tools use statistical analysis to guide debugging by interpreting the gathered data.

Our solution to the debugging/interpretation challenge is to use assertion annotations that invoke statistical tests on the measurement results, in order to help programmers decide if the results are consistent with three types of states (see figure on first page). Specifically, we use a chi-square statistical test to decide if the observed states belong to one of classical, superposition, or entangled states. These states correspond to tests for unimodal, uniform, and correlated sets of measurement outcomes. We focus our attention on these three types of states because they occur throughout a quantum program, and are easier for programmers to identify. These types of states are a subset of the quantum states that a quantum program can have, but cover many important cases.

Challenge #3: Programmers need guidelines for where to put assertions when debugging quantum programs. For the time being, the task of coding quantum programs entails translating quantum circuit diagrams into program code. The state-of-the-art in quantum programming is akin to programming classical computers 50 years ago: researchers are writing quantum programs operation-by-operation, on low-level bits of quantum information. This level of program abstraction is not conducive to debugging or for intelligently placing our proposed assertions.

One contribution of our paper is that it shows how the patterns and structures inside quantum algorithms guide programmers to know where to place assertions. This is where our broad survey of quantum algorithm primitives, problems, and language implementations pays off in terms of research insight (see table above). Program patterns common inside these algorithm primitives, such as looping operations, nesting operations, and mirroring operations, are higher-level programming abstractions that serve as guides for quantum programmers to know where to use the debugging tools. If the states don't match what the programmer expects, the statistical tests help the programmer zoom in and find mistakes in the program code.

Broader context of quantum program correctness: Our research in quantum debugging tools is an important and pragmatic approach to the problem of writing correct quantum programs. Prior proof-based approaches to quantum program correctness typically rely on functional programming language syntax, in languages such as Quipper, LIQUi|>, Q#, and

Qwire. While formal approaches are useful, they are not complete on their own, and they require stronger assumptions about the correct and noise-free operation of the underlying quantum hardware. In contrast, this work's statistical assertions are applicable to debugging both ideal and noisy quantum program runs. Furthermore, the assertions can be used in the procedural quantum programming languages (Qiskit, Cirq, Scaffold) that support much of the experimental work today. Just as in classical programming, quantum programmers will rely on a mix of pragmatic and formal techniques.

Long-term impact: Quantum computing is at a critical juncture. After decades of research into both quantum algorithms and underlying quantum physical devices, quantum computers are now at a size and reliability where programmers can actually run quantum programs. But quantum algorithms are unintuitive, and quantum programming languages are as of now too low-level to provide many automatic correctness guarantees (such as libraries of validated modules, data types representing numbers, type checking, and garbage collection) that programmers have come to expect in classical languages. Due to these challenges, researchers have identified quantum program debugging to be one of the urgent challenges hindering experimental progress with quantum computers[2].

Our work is among the first papers to study quantum programming bugs. Among those, this paper stands out in its detailed examples of quantum program code snippets, examples of how programmers may make mistakes, results on what symptoms those mistakes will cause, and then how a programmer may use the observable symptoms to diagnose the underlying bug. Our work has sparked discussion among quantum computing researchers about what bugs may appear when reproducing common quantum algorithms[3].

Our paper is also notable for its broad scope. We studied a comprehensive set of quantum algorithm primitives, concrete algorithms, and implementation languages, in order to create a taxonomy of bugs, programming patterns, and assertions that are general enough to cover those quantum programs. Our efforts led to a paper that we believe serves as a panoramic introduction to quantum algorithms, suitable for architecture researchers and curious undergrads.

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

[3] Robert Rand, Kesha Hietala, and Michael Hicks. 2019. Formal Verification vs. Quantum Uncertainty. Summit on Advances in Programming Languages, SNAPL.

Code release: Our research has been adopted into two open-source quantum software packages.

First, our quantum algorithm benchmarks for quantum chemistry and Shor's algorithm for factoring integers are now part of the benchmarks for the Scaffold programming language. This algorithm benchmark suite includes other algorithms that we have now debugged and validated, such as Grover's algorithm for database search, and an Ising spin chain model based on adiabatic quantum computing.

Second, our toolchain for using statistical tests to check on assertions is now a pull-request ready for acceptance into IBM's Qiskit framework, pending review from the code owners. Qiskit is the world's most widely-used framework for quantum computing tutorials and outreach to curious students. As such, once the integration is approved, our assertions toolchain will likely be the first QC debugger that many new quantum programmers will encounter, on a scale of thousands of QC programmers per year.

Follow-on work: Our paper has already been cited by a paper[4] discussing how a subset of our programming patterns and assertions can be checked dynamically, without stopping quantum program execution or destroying the quantum state.

The techniques in that paper may serve as a new form of lightweight error correction for quantum programs, where some error correction is feasible because knowing where to assert (classical, superposition, and entangled) states tells the quantum computer to selectively do measurement collapse, thereby creating an attractor state needed for error correction. That contrasts with full-blown quantum error correction without any knowledge of expected states, where such full quantum error correction is prohibitively costly in the foreseeable future due to the limited size and reliability of prototype quantum computers.

Our work supports that follow-on work by providing the insight about the useful types of assertions, and by providing the quantum algorithm programming patterns that would guide the placement of those dynamic assertions.

Citation for Test of Time award: This paper addresses important challenges in debugging quantum systems by identifying key algorithm and program patterns, and by proposing and evaluating assertion-based debugging methods.

[4] Huiyang Zhou and Gregory Byrd. 2019. Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation. In IEEE Computer Architecture Letters, vol. 18, no. 2, pp. 111-114, 1 July-Dec. 2019. doi: 10.1109/LCA.2019.2935049