COVER FEATURE

A Layered Software Architecture for Quantum Computing Design Tools

Krysta M. Svore, Alfred V. Aho
Columbia University

Andrew W. Cross, Isaac Chuang
Massachusetts Institute of Technology

Igor L. Markov
University of Michigan

Compilers and computer-aided design tools are essential for fine-grained control of nanoscale quantum-mechanical systems. A proposed four-phase design flow assists with computations by transforming a quantum algorithm from a high-level language program into precisely scheduled physical actions.

Quantum computers have the potential to solve certain computational problems—for example, factoring composite numbers or comparing an unknown image against a large database—more efficiently than modern computers. They are also useful in controlling quantum-mechanical systems in emergent nanotechnology applications, such as secure optical communication, in which modern computers cannot natively operate on quantum data.

Despite convincing laboratory demonstrations of quantum information processing, as the "Ongoing Research in Quantum Computing" sidebar describes, it remains difficult to scale because it relies on inherently noisy components. Adequate use of quantum error correction and fault tolerance theoretically should enable much better scaling, but the sheer complexity of the techniques involved limits what is doable today. Large quantum computations must also achieve a high degree of parallelism to complete before quantum states decohere.

As candidate quantum technologies mature, the feasibility of quantum computation will increasingly depend on software tools, especially compilers, that translate quantum algorithms into low-level, technology-specific instructions and circuits with added fault tolerance and sufficient parallelism.

We propose a layered software architecture consisting of a four-phase computer-aided design flow that assists with such computations by mapping a high-level language source program representing a quantum algorithm onto a quantum device. By weighing different optimization and error-correction procedures at appropriate phases of the design flow, researchers, algorithm designers, and tool builders can trade off performance and accuracy.

QUANTUM COMPUTATION

The *quantum circuit*,[1] a commonly used computation model similar to a modern digital circuit, provides a representation of a quantum algorithm. Digital circuits capture both mathematical algorithms, such as for sorting and searching, and methods for real-world control and measurement, as in cellular phones and automobiles. Quantum circuits likewise describe methods for control of quantum systems, such as atomic clocks and optical communication links, that cannot be fully controlled with conventional binary digital circuits alone.

A quantum circuit consists of quantum bits (qubits), quantum gates, quantum wires, and qubit measurements. A *qubit* is analogous to a classical bit but can be in a wave-like *superposition* of the symbolic bit values 0 and 1, written $a|0\rangle + b|1\rangle$ where a and b are complex numbers. Mathematically, a qubit can be written as a vector of complex numbers. When measured, a qubit

Ongoing Research in Quantum Computing

Researchers in industry and government labs are exploring various aspects of quantum design and automation with a wide range of applications. In addition to the examples described below, universities in the US, Canada, Europe, Japan, and China are carrying out much broader efforts.

BBN Technologies

Based in Cambridge, Massachusetts, BBN Technologies (www.bbn.com) developed the world's first quantum key distribution (QKD) network with funding from the US Defense Advanced Research Projects Agency. The fiber-optical DARPA Quantum Network offers 24x7 quantum cryptography to secure standard Internet traffic such as Web browsing, e-commerce, and streaming video.

D-Wave Systems

Located in Vancouver, British Columbia, Canada, D-Wave Systems (www.dwavesys.com) builds superconductor-based software-programmable custom integrated circuits for quantum optimization algorithms and quantum-physical simulations. These ICs form the heart of a quantum computing system designed to deliver massively more powerful and faster performance for cryptanalysis, logistics, bioinformatics, and other applications.

Hewlett-Packard

The Quantum Science Research Group at HP Labs in Palo Alto, California, is exploring nanoscale quantum optics for information-processing applications (www.hpl.hp.com/research/qsr). In addition, the Quantum Information Processing Group at the company's research facility in Bristol, UK, is studying quantum computation, cryptography, and teleportation and communication (www.hpl.hp.com/research/qip).

Hypres

Located in Elmsford, New York, Hypres Inc. (www.hypres.com) is the leading developer of superconducting digital circuits for wireless and optical communication. Based on rapid single-flux quantum logic, these circuits have achieved gate speeds up to 770 GHz in the laboratory.

IBM Research

Scientists at IBM's Almaden Research Center in California and the T.J. Watson Research Center's Yorktown office in New York developed a nuclear magnetic resonance (NMR) quantum computer that factored 15 into 3 $\times$5 (http://archives.cnn.com/2000/TECH/computing/08/15/quantum.reut). Researchers at the Watson facility and the Zurich Research Lab are also developing Josephson junction quantum devices (www.research.ibm.com/ss_computing) as well as studying quantum information theory (www.research.ibm.com/quantuminfo).

Id Quantique

Based in Geneva, Switzerland, id Quantique (www.idquantique.com) is a leading provider of quantum cryptography solutions, including wire-speed link encryptors, QKD appliances, a turnkey service for securing communication transfers, and quantum random number generators. The company's optical instrumentation product portfolio includes single-photon counters and short-pulse laser sources.

Los Alamos National Lab

The Los Alamos National Lab (http://qso.lanl.gov/qc) in New Mexico is studying quantum-optical long-distance secure communications and QKD for satellite communications. It has also conducted groundbreaking work on quantum error correction, decoherence, quantum teleportation, and the adaptation of NMR technology to quantum information processing.

MagiQ Technologies

MagiQ Technologies (www.magiqtech.com), headquartered in New York City, launched the world's first commercial quantum cryptography device in 2003. MagiQ Quantum Private Network systems incorporate QKD over metro-area fiber-optic links to protect against both cryptographic deciphering and industrial espionage.

NEC Labs

Scientists at NEC's Fundamental and Environmental Research Laboratories in Japan, in collaboration with the Riken Institute of Physical and Chemical Research, have demonstrated a basic quantum circuit in a solid-state quantum device (www.labs.nec.co.jp/Eng/innovative/E3/top.html). Recently, NEC researchers have also been involved in realizing the fastest fortnight-long, continuous quantum cryptography final-key generation.

NIST

The Quantum Information Program at the US National Institute of Standards and Technology (http://qubit.nist.gov) is building a prototype 10-qubit quantum processor as a proof-in-principle of quantum information processing. Potential applications include ultraprecise measurement (atomic clocks, optical metrology, and so on), control of dynamic processes, and nanotechnology. Researchers at the program's facilities in Boulder, Colorado, and Gaithersburg, Maryland, are also optimizing the speed of free-space quantum cryptography systems.

NTT Basic Research Labs

NTT's Superconducting Quantum Physics Research Group in Japan focuses on the development of quantum cryptography protocols (www.brl.ntt.co.jp/group/shitsuryo-g/qc). In particular, they have exhibited quantum cryptography using a single photon realized in a photonic network of optical fibers.

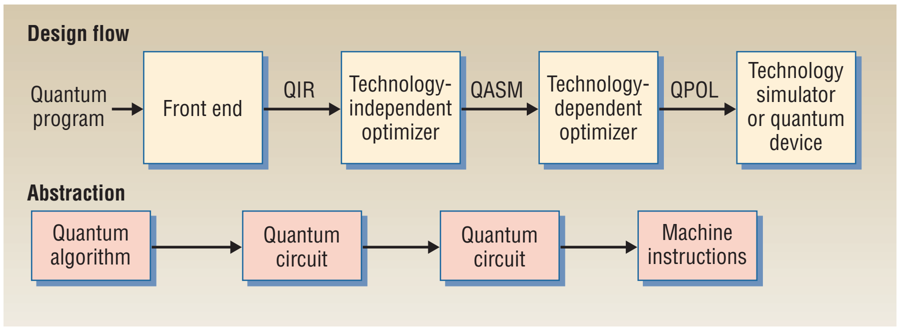


Figure 1. Proposed design flow. The first three phases are part of the quantum computer compiler, while the last phase implements the quantum algorithm on a quantum device or simulator.

assumes either the value 0 or the value 1, with probability $|a|^2$ and $|b|^2$, respectively.

An n-qubit quantum state is written as a vector representing a superposition of 2^n different bit strings. The state remains in a superposition for the computation's duration, and the final sequence of measurements collapses the state onto the bit string that gives the result of the computation. This result will not be affected if all bit strings in a given state are multiplied by a constant, called a *global phase*, before measurement. However, the ratios of coefficients of different bit strings are significant and determine *relative phases*.

A *quantum gate* is a reversible transformation of a quantum state that preserves total probability—for example, for a single qubit $|a|^2 + |b|^2 = 1$. Quantum gates are represented by unitary matrices that act on quantum state vectors by left multiplication. Gates are connected by *quantum wires* that transport qubits forward in time or space. Quantum wires cannot fan out—that is, qubits with unknown state cannot be duplicated. Matrix multiplication models composition of gates in series; the Kronecker, or tensor, product models composition of gates in parallel.

Inaccurate gates and uncontrolled environmental couplings introduce data errors. Uncontrolled coupling results in *decoherence,* which causes qubits to collapse to states that behave probabilistically, like (possibly biased) classical coins. Such states have no phase information and cannot perform quantum computation. These effects complicate quantum information processing, but researchers can address them using tools that perform optimizations and automatically add error correction.

FOUR-PHASE DESIGN FLOW

We envision a hierarchy of design tools with simple interfaces between layers that include programming languages, compilers, optimizers, simulators, and layout tools. Such an architecture appears necessary because no single entity can afford the huge investments required to develop all necessary tools. To this end, open source software encourages wider community participation.

A sufficiently transparent architecture facilitates tool interoperability, focused point-tool development, and incremental improvements. Quantum algorithm designers and those developing quantum circuit optimizations can explore new algorithms and error-correction procedures in more realistic settings involving actual noise and physical resource constraints. Researchers can also simulate important quantum algorithms on proposed new technologies before doing expensive lab experiments.

Our four-phase design flow, shown in Figure 1, maps a high-level program representing a quantum algorithm into a low-level set of machine instructions to be implemented on a physical device. The high-level quantum programming language encapsulates the mathematical abstractions of quantum mechanics and linear algebra.[1] The design flow's first three phases are part of the quantum computer compiler (QCC). The last phase implements the algorithm on a quantum device or simulator.

In addition to providing support for the abstractions used to specify quantum algorithms, the programming languages and compilers at the top level of our tool suite accommodate optimization improvements as our understanding of new quantum technologies matures. The simulation and layout tools at the bottom level incorporate details of the emerging quantum technologies that would ultimately implement the algorithms described in the high-level language. The tools balance tradeoffs involving performance, qubit minimization, and fault-tolerant implementations.

The representations of the quantum algorithm between the phases are the key to an interoperable tools hierarchy. In the first phase, the compiler front end maps a high-level specification of a quantum algorithm into a *quantum intermediate representation* (QIR)—a quantum circuit with gates drawn from some universal set. Compared to traditional logic circuits, quantum circuits are more structured and typically have intrinsic sequential semantics, wherein gates modify globally maintained state qubits in parallel.

In the second phase, a technology-independent optimizer maps the QIR into an equivalent lower-level circuit representation of single-qubit and controlled-NOT (CNOT) gates. The compiler optimizes this *Quantum Assembly Language* (QASM) according to a cost function such as circuit size, circuit depth, or accuracy. Since limiting quantum computing to a fixed set of registers and fixed word size would significantly restrict its power, QASM does not have such limitations, unlike traditional assembly languages. Therefore, parallelism has a greater impact and must be extracted by the compiler.

The third phase consists of optimizations suited to the quantum computing technology and outputs *Quantum Physical Operations Language* (QPOL), a physical-language representation with technology-specific parameters. QPOL includes two subphases: The first maps the representation of single-qubit and CNOT gates into a QASM representation using a fault-tolerant discrete universal set of gates; the second maps these gates into a QPOL representation containing the physical instructions for the fault-tolerant operations scheduled in parallel, including the required movements of physical particles. Knowledge of the physical layout and architectural limitations enters no later than at this step.

The final phase utilizes technology-dependent tools such as layout modules, circuit and physical simulators, or interfaces to actual quantum devices. If at this point certain technology constraints or objectives have not been met, algorithm and device designers can repeat some earlier phases. In addition, it is possible to add fault tolerance and error correction at multiple phases of the design process.

The "Sample Design Flow: EPR Pair Creation" sidebar provides a concrete example of how our proposed design flow automates the process of transforming mathematical models into software for controlling a live quantum-mechanical system.

PROGRAMMING ENVIRONMENT AND LANGUAGE

Designing a quantum programming environment is difficult given the currently limited repertoire of quantum algorithms. However, this situation is likely to improve as the demand for nanoscale control increases. The programming model is also uncertain because researchers can design a quantum computer as either an application-specific integrated circuit or a general-purpose processor. However, it is safe to assume that classical computers will monitor quantum devices through a bidirectional communication link.[2]

A quantum programming environment should possess several key characteristics.[2] First, it needs a high-level quantum programming language that offers the necessary abstractions to perform useful quantum operations. It should support complex numbers, quantum unitary transforms (quantum gates), and measurements as well as classical pre- and postprocessing. Support for reusable subroutines and gate libraries is also required. However, the exact modularization of a quantum programming environment remains an open question.

In addition, the environment as well as the programming language should be based on familiar concepts and constructs. This would make learning how to write, debug, and run a quantum program easier than using a totally new environment.

The quantum programming environment also should allow easy separation of classical and quantum computations. Because a quantum computer has noise and limited coherence time, this separation can limit computation time on the quantum device. The compiler for a quantum programming language should be able to translate a source program into an efficient and robust quantum circuit or physical implementation; it should be easy to translate into different gate sets or optimize with respect to a desired cost function.

The quantum programming environment should allow easy separation of classical and quantum computations.

Further, the high-level programming language should be hardware-independent and compile onto different quantum technologies. However, the language and environment should allow the inclusion of technology-specific modules.

A language that supports high-level abstractions would facilitate development of new quantum algorithms and applications. Researchers have proposed many quantum programming languages based on the quantum circuit model,[2,3] but a language that provides further insights on quantum information processing is needed. We also seek a language that simplifies creation of robust, optimized target programs.

QUANTUM COMPUTER COMPILER

A generic compiler for a classical language on a classical machine consists of a sequence of phases that transform the source program from one representation into another.[4] This partitioning of the compilation process has led to the development of efficient algorithms and tools for each phase. Because the front-end processes for QCCs are similar to those of classical compilers, researchers can use the algorithms and tools to build lexical, syntactic, and semantic analyzers for QCCs. However, the intermediate representations, the optimization phase, and the code-generation phase of QCCs differ greatly from classical compilers and require novel approaches, such as a way to insert error-correction operations into the target language program.

Quantum intermediate representation

Other popular quantum computation models, such as adiabatic quantum computing, can be converted to quantum circuits. Therefore, in our design flow's first phase, the QCC's front end maps a high-level specification of a quantum algorithm into a QIR based on the quantum circuit model.[1]

Provisions must be made in the QIR for classical and quantum control flows as well as data flows. In particular, quantum-to-classical conversions are accomplished via quantum measurements, while quantum conditionals and entangled switch statements are implemented using quantum multiplexer gates.[5] High-level optimizations may involve simultaneous changes to quantum and classical control flows and to data flows. We also consider fault-tolerant constructions at various phases in the design flow and incorporate circuit synthesis and optimization techniques in both the technology-independent and technology-dependent phases.

Circuit synthesis and optimization

During the second and third phases, the QCC synthesizes and optimizes a QASM representation of a quantum circuit using procedures similar to those currently used for digital circuits. Algorithms for classical logic circuit synthesis map a Boolean function into a circuit using gates from a given gate library. Similarly, quantum circuit synthesis creates a circuit that performs a given unitary transform up to an irrelevant global phase or a prescribed quantum measurement.

A digital logic designer can immediately construct a two-level circuit of a Boolean function, linear in the size of the function's truth table, and then use various techniques to optimize it. In contrast, finding a good quantum circuit to implement a $2^n \times 2^n$ unitary matrix is difficult. Only very recently have constructive algorithms become available that yield an asymptotically optimal circuit with $O(4^n)$ gates. Because CNOT gates are typically most expensive, their counts have been pushed down to only a factor of two away from lower bounds.[5] Remaining gates operate on single qubits at a time, but unlike CNOT gates their functionality can be tuned using continuous parameters.

When developing reusable software for automating quantum circuit design, reducing technological dependence is desirable. Today, the NAND gate is easier to implement than the AND gate in CMOS-based integrated circuits. Commercial circuit synthesis tools address this by decoupling libraryless logic synthesis from technology mapping. The former step uses an abstract gate library, such as AND-OR-NOT, and emphasizes the scalability of synthesis algorithms that capture the given

Sample Design Flow: EPR Pair Creation

Accurately capturing quantum-mechanical systems using traditional 0s and 1s is inherently difficult. Quantum information must therefore be processed directly—without converting it to bits—during state transformation and teleportation, communication, measurements, and other common tasks.

Figure A illustrates how our proposed four-phase design flow automates the transformation of mathematical models into software for controlling a live physical system.

An algorithm designer, researcher, or engineer initially expresses a mathematical specification of a quantum algorithm in a high-level quantum programming language, automatically creating a quantum circuit that encapsulates the mathematical abstractions of quantum mechanics and linear algebra.

In the design flow's first phase, the quantum computer compiler abstracts the quantum circuit as a quantum intermediate representation (QIR). Next, the QCC translates the circuit into Quantum Assembly Language (QASM) that captures a universal set of quantum gates. In the third phase, the QCC translates QASM instructions into Quantum Physical Operations Language using software tools. QPOL has knowledge of particulars of the quantum device, including layout and a technology-specific gate library. Finally, technology-dependent software tools translate QPOL into machine instructions.

In this example, we demonstrate how to produce Einstein-Podolsky-Rosen (EPR) pairs[1] for implementation on a trapped-ion computer. Trapped-ion systems have shown considerable potential as a future quantum computing technology.[2] These computers use charged, electromagnetically trapped atoms as qubit carriers and the internal state of single ionized atoms as qubits. Ions can be shuttled in and out of ion traps to increase the quantum computer's effective size.

An important physical resource for quantum computing and communication, EPR pairs are entangled quantum states that cannot be decomposed into (tensor products of) single-qubit states. They represent *quantum nonlocality* and have applications in quantum state teleportation, ultraprecise measurement, and lithography as well as in a number of quantum computing algorithms.

For EPR pair creation, we abstract the mathematical representation in a quantum circuit composed of a Hadamard (H) and CNOT gate. The figure shows sample QASM and QPOL representations. Determining the phase in which to insert fault tolerance and error correction is an open research question; here we show how to replace a CNOT gate with a circuit for a fault-tolerant encoded CNOT operation limited to local interactions.

QPOL instructions for creating an EPR pair can be translated into a sequence of laser pulses—in this case, for performing a CNOT gate on an ion-trap device. The machine instructions are as follows:

1. Alternately raise and lower the potentials of electrodes A, 1, 2, and 3 to move ions from trap A to trap B.
2. Apply a laser to the "green" ion to cool the ion chain that may have heated during movement.
3. Apply π pulse on the first red sideband of the *x* ion.
4. Apply pulse on carrier of the *y* ion.
5. Apply π pulse on the first red sideband of the *x* ion.
6. Split the "green" ion and the *x* ion away from the *y* ion and move them back to trap A.

The six-step process could take around 10-100 μs computation's global structure. The latter step converts all gates of a logic circuit to gates from a technology-specific gate library, often supplied by a chip manufacturer, and is based on local optimizations.

We expect the distinction between technology-independent circuit synthesis and technology mapping to carry over to quantum circuits.[6] This is precisely why the QCC maps the quantum algorithm into a QASM representation consisting of single-qubit and CNOT gates in the second phase of our design flow.

In addition, temporary decompositions into elementary gates could help optimize pulse sequences and reduce systematic inaccuracies in physical implementations. For example, a CNOT gate can be mapped onto a specific technology by appropriately timing pulses that couple two qubits, with pre- and postprocessing by less sophisticated pulses that affect single qubits.[6]

Technology-mapped circuits could potentially be optimized further via automatic instantiation of error correction, efficient handling of universal gate libraries without tunable gates, and identification of reusable quantum logic blocks and their efficient implementation.

Quantum Assembly Language

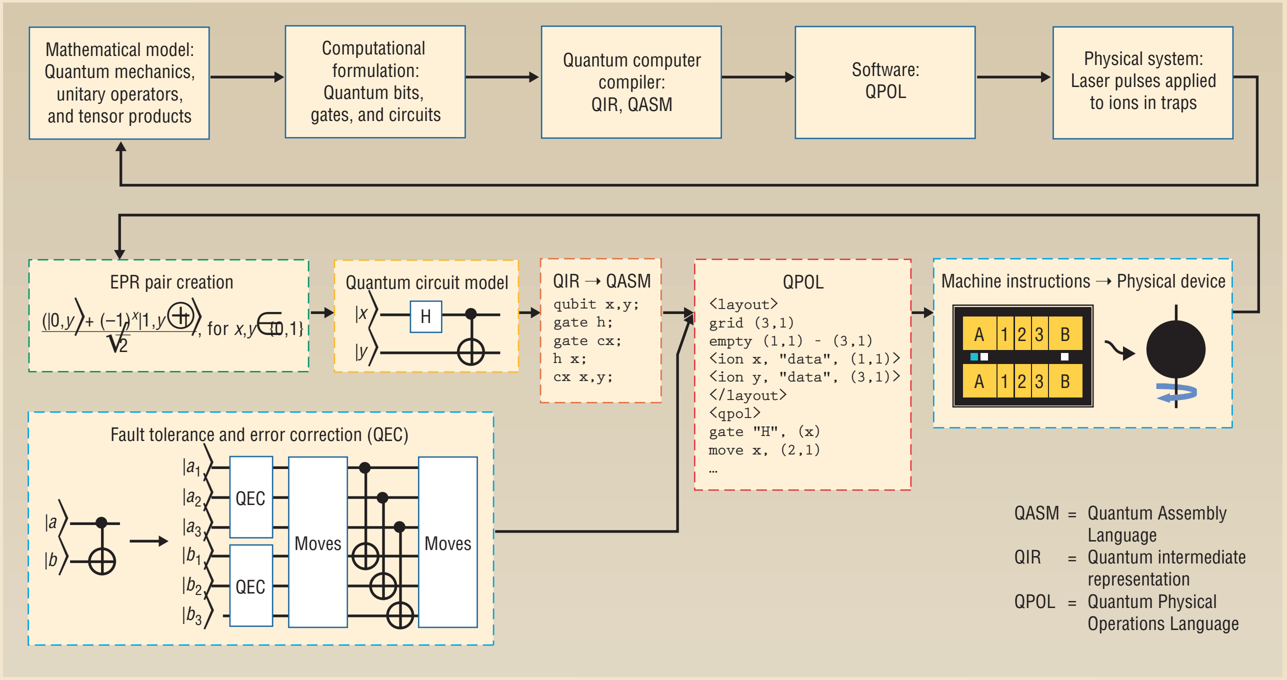
During the technology-independent phase of our design flow, the QCC maps a representation of the quantum algorithm into an equivalent set of Quantum Assembly Language instructions. QASM is a classical reduced-instruction-set computing assembly language extended by a set of quantum instructions based on the quantum circuit model. It uses qubits and registers of classical bits (cbits) as static units of information that must be declared at the program's beginning. Quantum instructions in QASM consist solely of single-qubit unitary gates, CNOT gates, and measurements. Any quantum circuit can be constructed using these instructions.

Quantum Physical Operations Language

QPOL precisely describes the execution of a given quantum algorithm expressed as a QASM program on a particular technology, like trapped-ion systems. QPOL includes physical operations as well as technology-

References

1. A. Einstein, B. Podolsky, and N. Rosen, "Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?" *Physical Rev.,* vol. 47, no. 10, 1935, pp. 777-780.
2. D.J. Wineland et al., "Experimental Issues in Coherent Quantum-State Manipulation of Trapped Atomic Ions," *J. Research of NIST,* vol. 103, no. 3, 1998, pp. 259-328.



Mathematical model: Quantum mechanics, unitary operators, and tensor products
Computational formulation: Quantum bits, gates, and circuits
Quantum computer compiler: QIR, QASM
Software: QPOL
Physical system: Laser pulses applied to ions in traps
EPR pair creation
$\frac{|0,y\rangle + (-1)^x|1,y \oplus 1\rangle}{\sqrt{2}}$, for $x,y \in \{0,1\}$
Quantum circuit model
$|x\rangle$
$|y\rangle$
H
QIR → QASM
QPOL
Machine instructions → Physical device
A 1 2 3 B
A 1 2 3 B
Fault tolerance and error correction (QEC)
$|a\rangle$
$|b\rangle$
$|a_1\rangle$
$|a_2\rangle$
$|a_3\rangle$
$|b_1\rangle$
$|b_2\rangle$
$|b_3\rangle$
QEC
QEC
Moves
Moves
QASM = Quantum Assembly Language
QIR = Quantum intermediate representation
QPOL = Quantum Physical Operations Language

```
qubit x,y;
gate h;
gate cx;
h x;
cx x,y;
```

```
<layout>
grid (3,1)
empty (1,1) - (3,1)
<ion x, "data", (1,1)>
<ion y, "data", (3,1)>
</layout>
<qpol>
gate "H", (x)
move x, (2,1)
…
```

Figure A. Using quantum information processing to control live physical systems. Proposed four-phase design flow, detailed for EPR pair creation on a trapped-ion computer with machine instructions translated into a sequence of laser pulses that perform a CNOT gate. A feedback loop allows for repetition of earlier phases.

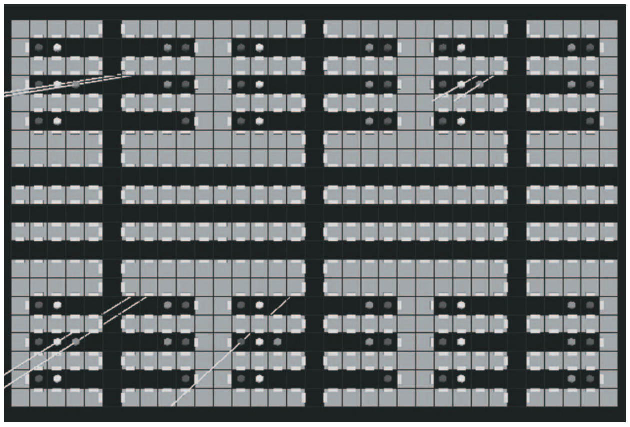


Figure 2. Trapped-ion simulator. Graphical display shows an H-tree layout. Qubits are electronic states of ions, represented by spheres, and gates are laser pulses, represented by lines. The qubits can move within the black regions but not into the substrate, drawn using light squares.

specific modules. In particular, it organizes physical operations into five instruction types:

- *Initialization* instructions specify how to prepare the initial system state. This can include loading qubits into the quantum computer, initializing auxiliary physical states used in computations, and setting qubits to $|0\rangle$.
- *Computation* instructions include quantum gates and measurements.
- *Movement* instructions control the relative distance between qubits to bring them together to undergo simultaneous operations or move them apart.
- *Classical control* instructions provide simple logic operations and allow quantum gates to be applied based on classical bit values stored in classical memory.
- *System-specific* instructions control physical parameters of the system that do not explicitly fall into the other categories.

The final QPOL distributes these instructions to the available instruction processing units—highly parallel quantum computers will have many—and by inserting appropriate waiting times.

In the case of trapped-ion computers, initialization has three stages: loading of multiple ions into a loading region, laser cooling to reduce ion temperatures, and optical pumping to put all qubits into a known state.

Computation is naturally described in terms of single-qubit rotations and a controlled-phase gate between ions in the same trap, both achieved using a laser pulse sequence. Measurement uses another laser pulse that causes ions in the $|0\rangle$ state to fluoresce. Electrostatic fields can move ions between multiplexed traps, and they can move multiple ions in and out of the same trap.

An external classical processor controls the execution of QPOL instructions, stores measurement results, and performs conditional instructions based on stored cbits.

System-specific instructions recool ions when they heat due to movement operations. Certain laser pulses also accomplish recooling, but the lasers are applied differently for cooling than for gates, requiring different programming and pulse-sequence optimization.

HIGH-PERFORMANCE SIMULATION OF QUANTUM CIRCUITS

Quantum-mechanical effects are useful for accelerating certain classical computations, as Lov Grover[7] and Peter Shor[8] have shown; however, numerical simulation of quantum computers on classical computers remains important for engineering reasons.

In classical electronic design automation, chip designers always test independent modules and complete systems by simulating them on test vectors before costly manufacturing. Numerical simulations can also help to evaluate quantum heuristics that defy formal worst-case analysis or only work well for a fraction of inputs.

For the numerical simulation phase of our design flow, we again use the quantum circuit formalism. Because mathematical models of quantum states, quantum gates, and measurement involve linear algebra, a key aspect of efficient simulation is exploiting the structure in the matrices and vectors derived from quantum circuits. To this end, researchers have proposed polynomial-time simulation techniques for circuits arising in error correction[9] and for "slightly entangled" quantum computation.

QuIDDPro: A generic graph-based simulator

George Viamontes and colleagues[10] have proposed a generic simulation technique based on data compression using the *quantum information decision diagram* (QuIDD) data structure. Its worst-case performance is no better than what can be achieved with basic linear algebra, but it can dramatically compress structured vectors and matrices, including all basis states, small gates, and some tensor products.

A QuIDD is a directed acyclic graph with one source and multiple sinks, each labeled with a complex number. The graph models matrix and vector elements as directed paths; any given vector or matrix can be encoded as a QuIDD and vice versa. Graph algorithms working on QuIDDs, supplied as a software library, implement all linear-algebraic operations in terms of compressed data representations.

Time and memory used by these algorithms to simulate a useful class of quantum circuits scale polynomially with the number of qubits. All components of Grover's algorithm, except for some application-dependent oracles, fall into this class. QuIDD-based simulation of the algorithm requires time and memory resources that are polynomial in the oracle function's size. If a compact

QuIDD can represent a particular oracle function for some search problem, then classical simulation of the algorithm runs nearly as fast as an ideal quantum circuit.

QuIDDs can also simulate density matrices by implementing several additional operations, such as trace-overs, in terms of graph traversals.[10] Straightforward modeling of any 16-qubit density matrix would require 64 Tbytes of memory. In contrast, for a reversible 16-qubit adder circuit using CNOT and Toffoli gates, the QuIDDPro package (http://vlsicad.eecs.umich.edu/quantum/qp) requires less than 5 Mbytes.

Trapped-ion simulator

Numerical simulations of quantum systems are also useful when studying the feasibility or performance of specific physical implementations.[9] We have carried out such a simulation for trapped-ion systems with up to 1,000 qubits; this applies to quantum *stabilizer circuits*, which are central to quantum error correction.

The keys to such realistic simulations are the layout of qubits in physical space and the scheduling of operations. Our layout tool maps circuits onto an *H-tree,* a recursively constructed fractal layout. This reduces movement operations required per gate by keeping qubits in inner codes near one another within concatenated quantum codes, which also have a self-similar structure. Our scheduler tool uses implicitly specified paths to optimize for minimal distances, expanding QASM instructions to include movements.

The simulator output includes the final quantum state (for circuit verification), measurement and failure histories, total execution time, and, in the case of a fault-tolerant circuit, validity of the final output. As Figure 2 shows, output also is a graphical display of QPOL instructions as they are simulated.

DESIGN FLOW FOR FAULT-TOLERANT ARCHITECTURES

The inherently noisy nature of quantum computers requires inserting error-correction routines and replacing gates with their fault-tolerant implementations to achieve scalability. A system architect can apply this process manually, synthesizing and laying out each fault-tolerant gate (*architecture-driven design*), or a compiler can apply it algorithmically (*software-driven design*).

We are currently considering both processes for trapped-ion computing systems, but the principles extend to other physical systems. The central goal of both designs is to guarantee that the final sequence of physical operations will execute fault-tolerantly on the target system—if failures occur infrequently enough, then the resulting errors cannot cause the system to fail.

Fault-tolerant classical components

In special applications of modern digital computers, the canonical method for fault-tolerant computation is triple modular redundancy.[11] TMR involves feeding gate inputs copied three times into three gates that fail with probability $O(p)$. The output lines of these faulty gates fan out into three majority voting gates. The majority gates essentially amplify the correct value of the computation so that the fault-tolerant gate fails only if two or more failures occur. Mathematically, the fault-tolerant gate fails with probability $O(p^2)$.

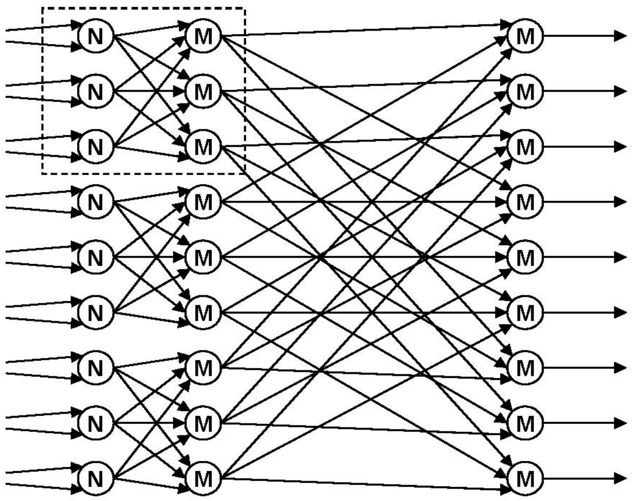


Figure 3. TMR fault-tolerant NAND gate at the second level of recursion, constructed from three fault-tolerant NAND (N) gates and three majority (M) gates.

Figure 3 shows a TMR fault-tolerant NAND gate at the second level of recursion, constructed from three fault-tolerant NAND gates and three majority gates. All gates are assumed to fail with probability p, such that the highlighted TMR NAND gate fails with probability $< 6p^2$, ignoring input errors. The entire circuit shown fails with probability $< 6^3p^4$. If $p < 1/6$, then this circuit is more reliable than a basic gate.

Applying TMR recursively k times, as illustrated in Figure 3 for $k = 2$, fault-tolerant components can be made to fail with probability bounded above by $p_f(k) = (cp)^{2k}/c$. The constant c is determined by the maximum number of fault paths through the highlighted circuit that lead the circuit to fail. In this case, $c = 6$ because at least two gates or two majority voters must fail. If each basic gate fails with probability $p < 1/c$, then $p_f(k) \to 0$ as $k \to \infty$. This construction exhibits a *fault-tolerance threshold* $p_{th} = 1/c$.

Fault-tolerant quantum components

We construct fault-tolerant quantum components using procedures similar to classical fault-tolerance techniques. They can encode quantum information using *quantum computation codes*[12] that allow fault-tolerant computation via a discrete universal set of gates. Calderbank-Shor-Steane codes are one family of quantum codes that allow a transversal implementation of an encoded CNOT gate. Transversal gates are always fault tolerant because they

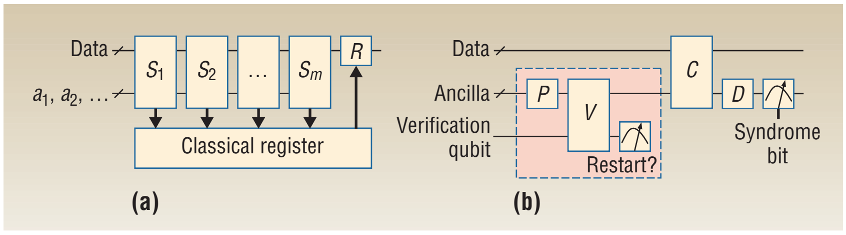


Figure 4. Fault-tolerant quantum computation. (a) Recovery operation. (b) Single syndrome bit extraction.

are implemented in a bitwise fashion—a gate between a pair of encoded qubits is implemented by applying the gate from bit 1 of the first encoded qubit to bit 1 of the second encoded qubit, and so on.

However, there are no known computation codes for which a universal set of encoded operations can be implemented transversally. In practice, performing quantum gates requires fault-tolerant preparation of several kinds of *ancillas,* or scratch qubits. After each gate, we insert on each qubit a recovery operation that consumes a *syndrome extraction ancilla* to acquire syndrome bits. Syndrome extraction ancillas must be available in great supply and may need to be checked for critical errors using *verification ancillas*. All of these operations must remain fault tolerant when qubits can only interact locally.[13]

Figure 4 illustrates key aspects of this process. A recovery operation, shown in Figure 4a, interacts fault-tolerantly with the data via syndrome bit extraction networks S_1,S_2, … S_m. This involves using a syndrome extraction ancilla (a_1,a_2, …) to measure each syndrome bit, possibly several times, and storing the results to a classical register. A classical computer processes the register and applies the appropriate error correction R to the data. Recovery operations follow every fault-tolerant gate to correct errors potentially introduced by that gate.

As Figure 4b shows, extracting a single syndrome bit fault-tolerantly first requires an ancilla state. The highlighted network prepares (P)and verifies (V) the ancilla; a verification qubit indicates if the ancilla failed the verification network V. Upon successful preparation of an ancilla, the C network interacts with the data fault-tolerantly to collect a syndrome bit. The quantum network D then decodes and measures the bit. Some classical postprocessing may take the place of D.

Fault-tolerant architectures

A quantum computation code conceptually separates the logical and physical machine. Both architecture-driven and software-driven designs exploit this fact to yield two different processes within the framework of our design flow.

An architecture-driven design process inserts fault-tolerant gates from a predesigned library during technology-dependent code generation. A design team creates the library of universal, fault-tolerant, technology-specific components using a combination of replacement rules, heuristic methods, and device models, then publishes the library together with design rules for connecting the composite components.

A software-driven design process inserts fault-tolerant gates during technology-independent code generation using replacement rules based on quantum circuits. Sophisticated schedulers and layout tools insert QPOL instructions to preserve fault tolerance. Algorithmic optimizations make fine-grained replacements, and compilers can use feedback from simulators to focus the optimizers on the circuit's critical regions. Our software architecture allows such insertion and testing of error-correction and fault-tolerance techniques at multiple stages in the design flow.

Our work has thus far focused on the languages, transformations, and fault-tolerance procedures needed along the design flow to produce robust implementations. However, many important challenges remain to be solved before researchers can build or even realistically design a scalable quantum computer.

To effectively use available quantum resources, we must be able to schedule and synchronize parallel quantum computations. We also need efficient technology-independent optimization algorithms for realistic classes of quantum circuits as well as strategies for adapting generic circuits to specific architectural constraints and implementation technologies.

Identifying and evaluating meaningful architectural design blocks will necessitate further development of simulation techniques for quantum circuits and high-level programs.

Achieving robust, scalable quantum computation will require both fault-tolerant architectural strategies compatible with emerging quantum device technologies and optimization algorithms that minimize the number of fault paths, code size, or number of gates in fault-tolerant circuits.

It will also be necessary to match tools to experimental implementations as well as develop methodologies for design verification and test such as quantum state tomography, circuit-equivalence checking, and test-vector generation.

The grandest challenge of all is to design a high-level programming language that encapsulates the principles of quantum mechanics in a natural way so that physicists and programmers can develop and evaluate more quantum algorithms.

Design and verification tools for robust quantum circuits are vital to the future of quantum information processing systems, and their development will be a natural evolutionary step as such machines graduate from the laboratory to engineering design.

Acknowledgments

We are grateful to Stephen Edwards for many helpful comments on computer-aided design flows. Krysta Svore acknowledges support from an NPSC fellowship, Andrew Cross acknowledges support from an NDSEG fellowship, and Igor Markov acknowledges funding from the DARPA QuIST program and the NSF.

References

1. M.A. Nielsen and I.L. Chuang, *Quantum Computation and Quantum Information,* Cambridge Univ. Press, 2000.
2. S. Bettelli, T. Calarco, and L. Serafini, "Toward an Architecture for Quantum Programming," *The European Physics J. D,* vol. 25, no. 2, 2003, pp. 181-200.
3. B. Ömer, "A Procedural Formalism for Quantum Computing," doctoral dissertation, Dept. Theoretical Physics, Technical Univ. of Vienna, 1998.
4. A.V. Aho, R. Sethi, and J.D. Ullman, *Compilers: Principles, Techniques, and Tools,* Addison-Wesley, 1986.
5. V.V. Shende, S.S. Bullock, and I.L. Markov, "Synthesis of Quantum Logic Circuits," to appear in *IEEE Trans. Computer-Aided Design of Integrated Circuits,* 2006; http://arxiv.org/abs/quant-ph/0406176.
6. V.V. Shende, I.L. Markov, and S.S. Bullock, "Finding Small Two-Qubit Circuits," *Proc. SPIE,* vol. 5436, Apr. 2004, pp. 348-359.
7. L.K. Grover, "A Fast Quantum Mechanical Algorithm for Database Search," *Proc. 28th Ann. ACM Symp. Theory of Computing,* ACM Press, 1996, pp. 212-219.
8. P.W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Computing,* vol. 26, no. 5, 1997, pp. 1484-1509.
9. S. Aaronson and D. Gottesman, "Improved Simulation of Stabilizer Circuits," *Physical Rev. A,* vol. 70, no. 5, 2004; www.scottaaronson.com/papers/chp5.pdf.
10. G.F. Viamontes, I.L. Markov, and J.P. Hayes, "Graph-Based Simulation of Quantum Computation in the State-Vector and Density-Matrix Representation," *Quantum Information and Computation*, vol. 5, no. 2, 2005, pp. 113-130.
11. J. von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies,* C.E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956, pp. 329-378.
12. D. Aharonov and M. Ben-Or, "Fault-Tolerant Computation with Constant Error," *Proc. 29th Ann. ACM Symp. Theory of Computing,* ACM Press, 1997, pp. 176-188.
13. K.M. Svore, B.M. Terhal, and D.P. DiVincenzo, "Local Fault-Tolerant Quantum Computation," *Physical Rev. A*, vol. 72, no. 5, 2005; http://arxiv.org/abs/quant-ph/0410047.

Krysta M. Svore *is a PhD student in the Department of Computer Science at Columbia University. Her research interests include quantum computation, particularly quantum fault tolerance and error correction, as well as data mining and intrusion detection. Svore received an MS in computer science from Columbia University. Contact her at kmsvore@cs.columbia.edu.*

Alfred V. Aho *is Lawrence Gussman Professor of Computer Science and vice chair for undergraduate education in the Department of Computer Science at Columbia University. His research interests include quantum computing, programming languages, compilers, and algorithms. Aho received a PhD in electrical engineering and computer science from Princeton University. He is a Fellow of the American Association for the Advancement of Science, the ACM, and the IEEE. Contact him at aho@cs.columbia.edu.*

Andrew W. Cross *is a PhD candidate in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology, and a research assistant in the Quanta Group at MIT Media Lab's Center for Bits and Atoms. His research focuses on fault-tolerant quantum computing. Cross received an MS in electrical engineering and computer science from MIT. Contact him at awcross@mit.edu.*

Isaac Chuang *is an associate professor with joint appointments in the Department of Electrical Engineering and Computer Science and the Department of Physics at MIT, where he also leads the Quanta Group at MIT Media Lab's Center for Bits and Atoms. His research interests include quantum information science, AMO implementations of quantum computers, quantum algorithms, and architectures for quantum information systems. Chuang received a PhD in electrical engineering from Stanford University. Contact him at ichuang@mit.edu.*

Igor L. Markov *is an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Michigan. His research interests include physical design and physical synthesis for VLSI, synthesis and simulation of quantum circuits, and artificial intelligence. Markov received a PhD in computer science from the University of California, Los Angeles. He is a member of the ACM and the IEEE Computer Society, and a senior member of the IEEE. Contact him at imarkov@eecs.umich.edu.*