A systems view of quantum computer engineering

Wednesday, September 16, 2020
Rutgers University
Yipeng Huang
Due Sunday, Sept. 20: read one of four articles. Share sketches / notes.

Due Wednesday, Sept. 23: choices for class debates / presentations.

• The schedule for in-class debates and seminar paper presentations is ready:


• Expectation: for course credit, each student will give two 30-minute debate or stand-alone presentations.

• Due one week before each presentation: an outline (for your first presentation) or slides (for your second presentation) for feedback.
Due Wednesday, Sept. 23: choices for class debates / presentations.

- Due Wednesday, September 23: your ranking of six most preferred presentation dates and papers. (1=most desired, 6=less desired)

- You have the option of suggesting one article outside of the papers I selected, if you have something in mind. I will check to make sure it fits with class topic.

- Consider these factors when picking: your interests, your background (e.g., HPC, microarchitecture, device physics, etc.), time commitments (other class projects, research deadlines).

- Please use Friday 11am office hours, or by appointment, or email for guidance on what papers to read and present.
Due Wednesday, Sept. 23: choices for class debates / presentations.

• I will do my best to honor everyone's top picks.

• But I will also aim to have good distribution of topics among the paper options and across class days.

• Hopefully, asking you to pick six options will lead to good distribution, but if that is not the case, I may ask for another round of selection.
Class grading rubric will be as follows:

• 2 × 25 points on the presentations

• 2 × 25 points on the programming exercises

• Week-to-week, I may have 5-point assignments to encourage engagement and to spur discussion about the papers.
These two lectures: Broad view of open challenges in quantum computer engineering

- A complete view of full-stack quantum computing.
- In short, challenges are in finding and building abstractions.
- In each layer, why we don’t or can’t have good abstractions right now.
- Recent and rapidly developing field of research.

Figure 1. Overview of the quantum computer system stack. A Microarchitecture for a Superconducting Quantum Processor. Fu et al.
All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits
Shor’s integer factoring algorithm

Factoring underpins cryptosystems.
For number represented as N bits—

Classical algorithm: needs $O(2^{3\sqrt{N}})$ operations
Factoring 512-bit integer: 8400 years. 1024-bit integer: $13 \times 10^{12}$ years.

Quantum algorithm: needs $O(N^2 \log(N))$ operations
Factoring 512-bit integer: 3.5 hours. 1024-bit integer: 31 hours.

A Practical Architecture for Reliable Quantum Computers. Oskin et al.
The horizontal axis is the length of the number to be factored. The steep curve is NFS, with the marked point at \( L = 768 \) requiring 3,300 CPU-years. The vertical line at \( L = 20,480 \) is NIST's 2007 recommendation for RSA key length for data intended to remain secure until 2030. The other lines are various combinations of quantum computer logical clock speed for a three-qubit operation known as a Toffoli gate (1 Hz and 1 MHz), method of implementing the arithmetic portion of Shor's algorithm (BCDP, D, and F), and quantum computer architecture (NTC and AC, with the primary difference being whether or not long-distance operations are supported). The assumed capacity of a machine in this graph is \( 2^L \) logical qubits. This figure illustrates the difficulty of making pronouncements about the speed of quantum computers.
What are the societal impacts?

• Is it good to threaten the cryptosystems that underpin secure and private communications? No!

• But, current crypto systems rely on assumptions about hardness of tasks, such as factoring and discrete logarithm.

• Developments in quantum algorithms pushes the boundary of what we know is easy vs. hard.

• Spurs the development of more secure cryptosystems (lattices, etc.).
All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. *Mature, high level quantum programming languages*
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits
Three abstraction challenges for quantum programming:

• *Truly high-level quantum algorithm programming abstractions.*

• Widely adopted, high performance, open source source language.

• More expressive quantum intermediate representations.
What do we mean by high-level language?

// Create a list of strings
ArrayList<String> al = new ArrayList<String>();
al.add("Quantum");
al.add("Computing");
al.add("Programs");
al.add("Systems");

/* Collections.sort method is sorting the
elements of ArrayList in ascending order. */
Collections.sort(al);
High-level language hides all these concerns:

- Choice of sorting algorithm implementation and comparator function on Strings
- Encoding of Strings as binary numbers
- Memory allocation and deallocation for String storage
- Execution of Java code on different ISAs
- Correctness of library implementation
- ...
• We want to get to that point with quantum programming

• Want to do things such as: initialize quantum data, perform search, without worrying about the detailed implementation
Grover’s database search

Figure 6.1. Schematic circuit for the quantum search algorithm. The oracle may employ work qubits for its implementation, but the analysis of the quantum search algorithm involves only the \( n \) qubit register.

Figure 6.2. Circuit for the Grover iteration, \( G \).

Operations based on classical problem information

Nielsen and Chuang, p251
**Table 7** Grover’s amplitude amplification subroutine in two languages, showcasing QC-specific language syntax for reversible computation (rows 2 & 6) and controlled operations (rows 3 & 5).

<table>
<thead>
<tr>
<th><strong>Scaffold (C syntax) [13]</strong></th>
<th><strong>ProjectQ (Python syntax) [36]</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 int j;</td>
<td># reflection across</td>
</tr>
<tr>
<td>qbit ancilla[n-1]; // scratch register</td>
<td># uniform superposition</td>
</tr>
<tr>
<td>for(j=0; j&lt;n-1; j++) PrepZ(ancilla[j], 0);</td>
<td></td>
</tr>
<tr>
<td>2 // Hadamard on q</td>
<td>with Compute(eng):</td>
</tr>
<tr>
<td>for(j=0; j&lt;n; j++) H(q[j]);</td>
<td>All(H)</td>
</tr>
<tr>
<td>// Phase flip on q = 0...0 so invert q</td>
<td>All(X)</td>
</tr>
<tr>
<td>for(j=0; j&lt;n; j++) X(q[j]);</td>
<td>with Control(eng, q[0:-1]):</td>
</tr>
<tr>
<td>3 // Compute x[n-2] = q[0] and ... and q[n-1]</td>
<td></td>
</tr>
<tr>
<td>CCNOT(q[1], q[0], ancilla[0]);</td>
<td></td>
</tr>
<tr>
<td>for(j=1; j&lt;n-1; j++)</td>
<td></td>
</tr>
<tr>
<td>CCNOT(ancilla[j-1], q[j+1], ancilla[j]);</td>
<td></td>
</tr>
<tr>
<td>4 // Phase flip Z if q=00...0</td>
<td></td>
</tr>
<tr>
<td>cZ(ancilla[n-2], q[n-1]);</td>
<td></td>
</tr>
<tr>
<td>5 // Undo the local registers</td>
<td>Z</td>
</tr>
<tr>
<td>for(j=n-2; j&gt;0; j--)</td>
<td>with Control(eng, q[0:-1]):</td>
</tr>
<tr>
<td>CCNOT(ancilla[j-1], q[j+1], ancilla[j]);</td>
<td></td>
</tr>
<tr>
<td>CCNOT(q[1], q[0], ancilla[0]);</td>
<td></td>
</tr>
<tr>
<td>6 // Restore q</td>
<td># ProjectQ automatically</td>
</tr>
<tr>
<td>for(j=0; j&lt;n; j++) X(q[j]);</td>
<td># uncomputes control</td>
</tr>
<tr>
<td>for(j=0; j&lt;n; j++) H(q[j]);</td>
<td>Uncompute(eng)</td>
</tr>
</tbody>
</table>
Three abstraction challenges for quantum programming:

• Truly high-level quantum algorithm programming abstractions.

• *Widely adopted, high performance, open source source language.*

• More expressive quantum intermediate representations.
FIG. 1: A schematic diagram showing the paths to connecting a personal computer to a usable gate-level quantum computer. Starting from the personal computer (bottom center), nodes in green show software that can be installed on the user's personal computer. Grey nodes show simulators run locally (i.e., on the user’s computer). Dashed lines show API/cloud connections to company resources shown in yellow clouds. Quantum simulators and usable quantum computers provided by these cloud resources are shown in blue and gold, respectively. Red boxes show requirements along the way. For example, to connect to Rigetti Forest and use the Agave 8 qubit quantum computer, one must download and install pyQuil (available on macOS, Windows, and Linux), register on Rigetti’s website to get an API key, then request access to the device via an online form. Notes: (i) Rigetti’s Quantum Virtual Machine requires an upgrade for more than 30 qubits, (ii) local simulators depend on the user’s computer so numbers given are approximate, and (iii) the grey box shows quantum computers that have been announced but are not currently available to general users.
Three abstraction challenges for quantum programming:

- Truly high-level quantum algorithm programming abstractions.
- Widely adopted, high performance, open source source language.
- More expressive quantum intermediate representations.
Qubits take on binary values; supply initial qubit values

| q0m0 = 0 \lor q0m0 = 1 | q1m0 = 0 \lor q1m0 = 1 |
| q0m0 = 0 | q1m0 = 0 |
| q0m1 = 0 \lor q0m1 = 1 | q1m3 = 0 \lor q1m3 = 1 |

Hadamard gate

| q0m0 = 0 \land q0m1 = 0 \implies +\frac{1}{\sqrt 2} | q0m0 = 0 \land q0m1 = 1 \implies +\frac{1}{\sqrt 2} |
| q0m0 = 1 \land q0m1 = 0 \implies +\frac{1}{\sqrt 2} | q0m0 = 1 \land q0m1 = 1 \implies -\frac{1}{\sqrt 2} |

Phase damping noise channel

| q0m2rv = 0 \land q0m2rv = 1 | q0m1 = 1 \land q0m2rv = 0 \implies +0.8 |
| q0m1 = 0 \implies q0m2rv = 0 | q0m1 = 1 \land q0m2rv = 1 \implies -0.6 |

CNOT gate

| q0m1 = 0 \land q1m0 = 0 \implies q1m3 = 0 | q0m1 = 1 \land q1m0 = 0 \implies q1m3 = 1 |
| q0m1 = 0 \land q1m0 = 1 \implies q1m3 = 1 | q0m1 = 1 \land q1m0 = 1 \implies q1m3 = 0 |
Benchmarking ZX Calculus Circuit Optimization Against Qiskit Transpilation. Yeh et al.
All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. **Universally accepted quantum ISAs**
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits
A small set of quantum gates are universal...

\[ |0\rangle \xrightarrow{R(\theta, \phi)} \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \]

\[ |1\rangle \xrightarrow{R(\theta, \phi)} \cos\left(\frac{\theta}{2}\right)|1\rangle - e^{-i\phi}\sin\left(\frac{\theta}{2}\right)|0\rangle \]

FIG. 1. The rotation and controlled-NOT (CNOT) gates are an example of a universal quantum gate family when available on all qubits, with explicit evolution (above) and quantum circuit block schematics (below). (a) The single-qubit rotation gate \( R(\theta, \phi) \), with two continuous parameters \( \theta \) and \( \phi \), evolves input qubit state \( |x\rangle \) to output state \( |\tilde{x}\rangle \). (b) The CNOT (or reversible XOR) gate on two qubits evolves two (control and target) input qubit states \( |x_C\rangle \) and \( |x_T\rangle \) to output states \( |\tilde{x}_C = x_C\rangle \) and \( |\tilde{x}_T = x_C \oplus x_T\rangle \), where \( \oplus \) is addition modulo 2, or equivalently the XOR operation.

Quantum Computer Systems for Scientific Discovery. Alexeev et al.
...but those universal gates decompose various ways...

A systems perspective of quantum computing. Matsuura et al.
...and different quantum hardware support different gates.

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

All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. **Uniform, fully connected quantum device architectures**
5. Reliable quantum gates and qubits
Fig. 1. Examples of several IBM cloud accessible devices. The top left 5-qubit device was the first one made available via the IBM Quantum Experience [40]. The one to the right of it was made available after including additional entangling gates between two pairs of qubits. A 16-qubit device was made available approximately a year after the first device. The devices in the bottom row show three variations of 20-qubit devices available to members of the IBM Q Network [41].
All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. **Reliable quantum gates and qubits**
Fig. 2. Controlled-NOT (CNOT) gate error distributions for a variety of IBM devices. Beginning with the earlier devices (top rows), the average error rates remained quite large but have improved with continuing research. The bottom row represents the device shown in Fig. 4. The error reductions are the result of improved gate fidelities and increasing coherence times [44], [45], as well as a better understanding of spectator qubit errors.

Challenges and Opportunities of Near-Term Quantum Computing Systems. Corcoles et al.
All the quantum computer abstractions we don’t yet have right now

0. Quantum computer support for quantum computer engineering
1. Fault-tolerant, error-corrected quantum algorithms
2. Mature, high level quantum programming languages
3. Universally accepted quantum ISAs
4. Uniform, fully connected quantum device architectures
5. Reliable quantum gates and qubits
Position statement for this graduate seminar

• Quantum computer engineering has become important.

• Requires computer systems expertise beyond quantum algorithms and quantum device physics.