Where we are in the semester

Full stack quantum computer engineering

1. Algorithms: QAOA & VQE
2. Programming languages, assertions, stabilizers
3. Google Cirq, IBM Qiskit
4. Quantum circuit simulation and quantum supremacy
5. Extracting success: quantum computer architecture
6. Prototypes: quantum computer microarchitecture

► Programming assignments (2 × 25 points)
► Seminar presentations (2 × 25 points)
Compiling from a high-level program to hardware

Goals:
1. Correctness: maximizing probability of success!
2. Ease of programming?
3. Compatibility between hardware implementations?

Extreme device/resource constraints:
1. Native gate set
2. Device topology
3. Hardware noise
4. Parallelism constraints
Compiling from a high-level program to hardware

FIGURE 6.1 A generic tool flow for quantum programming. A quantum program is implemented in a domain-specific language (DSL) and then translated into hardware instructions after undergoing a series of compiler transformations and optimizations. A quantum intermediate representation (QIR) of the program can serve as a logical-level analog to conventional assembly code. For programs running on error-corrected qubits, the compiler would link in low-level QEC libraries into the code, transforming the logical qubit operations, to the physical operations on a number of qubits. The qubits of this “expanded” quantum program are then mapped onto a specific hardware implementation accounting for the specific gate operations and connectivity available. At the lowest level, the operations on physical qubits will be generated as instructions of the quantum control processor that orchestrate the specific control pulses (e.g., microwave or optical) required. For more detailed discussion of quantum computer software architectures see [Chong, Frederic T., Diana Franklin, and Margaret Martonosi. "Programming languages and compiler design for realistic quantum hardware," *Nature* 549, no. 7671 (2017): 180.] and [Häner, Thomas, Damian S. Steiger, Krysta Svore, and Matthias Troyer. "A software methodology for compiling quantum programs." *Quantum Science and Technology* 3, no. 2 (2018): 020501].
Compiling from a high-level program to hardware

Figure 4.1: Selective sharing of information allows algorithms to use limited resource in NISQ hardware most efficiently.

Figure: Credit: [Ding and Chong, 2020]
Native gate set

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

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

(a) $R(\theta, \phi)$ gate

(b) CNOT gate

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 $|\bar{x}\rangle$. (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 $|\bar{x}_C = x_C\rangle$ and $|\bar{x}_T = x_C \oplus x_T\rangle$, where $\oplus$ is addition modulo 2, or equivalently the XOR operation.

Figure: Credit: [Alexeev et al., 2020]

- Clifford + T ISA is sensible for an error-corrected machine
- But for NISQ machine, best two-qubit gate is dependent on native gate set
BOX 2. CHOOSING A CNOT GATE DECOMPOSITION

A logical controlled-NOT, or CNOT, gate is an entangling operation that flips the target qubit between 1 and 0 if the control qubit is in the 1 state. It must be decomposed into a sequence of native quantum gates for the qubit technology to perform the gate operation on the specific qubit system. Two possible decompositions are shown, where RX, RY, and RZ denote rotations around the x-, y- and z-axes, respectively. A system performance simulation could provide metrics to help choose which CNOT to incorporate into the specific design.

Depending on the fidelity of the single- and two-qubit gates in the circuits and on their speed, researchers may want to choose only one to implement the logical CNOT in the system. Depending on the performance of the qubits available at a particular point in the execution of the algorithm, it is also possible to choose a different logical CNOT sequence.

Figure: Credit: [Matsuura et al., 2019]
Native gate set

**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.

**Figure:** Credit: [Murali et al., 2019]

Two qubit gates remain dominant sources of errors.
Compiling from a high-level program to hardware

Goals:

1. Correctness: maximizing probability of success!
2. Ease of programming?
3. Compatibility between hardware implementations?

Extreme device/resource constraints:

1. Native gate set
2. Device topology
3. Hardware noise
4. Parallelism constraints
Device topology

- ion trap qubits: fully connected topology
- superconducting qubits: arbitrary qubits cannot directly interact; needs chain of swap gates

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].
Superconducting qubits: arbitrary qubits cannot directly interact; needs chain of swap gates
Device topology

Figure 3. (a) Layout of a 6-qubit quantum computer, (b)-(e) are possible routes from A to F. Note that options (b)(c)(d) have an identical number of swaps and (e) incur higher swaps. An intelligent policy would choose one from (b)(c)(d).

Figure: Credit: [Tannu and Qureshi, 2019]

Superconducting qubits: arbitrary qubits cannot directly interact; needs chain of swap gates
Compiling from a high-level program to hardware

Goals:

1. Correctness: maximizing probability of success!
2. Ease of programming?
3. Compatibility between hardware implementations?

Extreme device/resource constraints:

1. Native gate set
2. Device topology
3. Hardware noise
4. Parallelism constraints
Hardware noise

1. Decoherence error
2. Gate error (imprecise control of single qubit, two qubit gates)
3. Measurement error

<table>
<thead>
<tr>
<th>Technology</th>
<th>Coherence Time (s)</th>
<th>1-Qubit Gate Latency (s)</th>
<th>2-Qubit Gate Latency (s)</th>
<th>1-Qubit Gate Fidelity (%)</th>
<th>2-Qubit Gate Fidelity (%)</th>
<th>Mobile</th>
</tr>
</thead>
</table>

Table 1. Metrics for various quantum technologies. * Nuclear/Electron Hybrid

Figure: Credit: [Resch and Karpuzcu, 2019]
Hardware noise

1. Decoherence error
2. Gate error (imprecise control of single qubit, two qubit gates)
3. Measurement error
Hardware noise

Stochastic, uncorrelated noise

<table>
<thead>
<tr>
<th></th>
<th>Quantum noise mixtures (Pauli errors)</th>
<th>Quantum noise channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pauli-X type</td>
<td>Bit flip noise</td>
<td>Amplitude damping noise (related to T1 time)</td>
</tr>
<tr>
<td>Pauli-Z type</td>
<td>Phase flip noise</td>
<td>Phase damping noise (related to T2 time)</td>
</tr>
<tr>
<td>Combinations</td>
<td>Symmetric / asymmetric depolarizing noise</td>
<td>Generalized amplitude damping</td>
</tr>
<tr>
<td>Simulation technique</td>
<td>Can model as probabilistic ensembles of state vectors</td>
<td>Requires density matrix representation</td>
</tr>
</tbody>
</table>

**Table:** Summary of canonical quantum noise models.
Bit flip noise channel

$|0\rangle \rightarrow \text{BitFlip}(0.64) \rightarrow \begin{cases} 
P(|0\rangle) = 0.64 \\
P(|1\rangle) = 0.36 
\end{cases}$

We represent such a mixture of quantum states as a density matrix:

$0.64 |0\rangle \langle 0| + 0.36 |1\rangle \langle 1|
= 0.64 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} + 0.36 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix}
= 0.64 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + 0.36 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}
= \begin{bmatrix} 0.64 & 0 \\ 0 & 0.36 \end{bmatrix}$

(Conventions from [Nielsen and Chuang, 2011, Chapter 8.3])
Density matrix representation

\[
0.64 |0\rangle \langle 0| + 0.36 |1\rangle \langle 1| = \begin{bmatrix} 0.64 & 0 \\ 0 & 0.36 \end{bmatrix}
\]

More general representation:
\[
\rho = \sum_j p_j |\psi_j\rangle \langle \psi_j|
\]
Quantum (noise) channel

A quantum channel $\mathcal{E}(\rho)$ acts on mixed state $\rho$:

$$\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$$
The bit flip channel flips the state of a qubit with probability $1 - p$. It has two elements:

\[ E_0 = \sqrt{p}I = \sqrt{p} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]

\[ E_1 = \sqrt{1 - p}X = \sqrt{p} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \]
The bit flip noise channel $\mathcal{E}_{\text{bitflip}}(0.64)$ acts on the $|0\rangle$ state like so:

$$\mathcal{E}_{\text{bitflip}}\left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\right) = \sum_k E_k \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} E_k^\dagger$$

$$= 0.8 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} 0.8 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + 0.6 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} 0.6 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.64 & 0 \\ 0 & 0.36 \end{bmatrix}$$
Phase flip noise channel

The phase flip channel flips the phase of a qubit with probability $1 - p$. It has two elements:

$$E_0 = \sqrt{p} I = \sqrt{p} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$E_1 = \sqrt{1 - p} Z = \sqrt{p} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
## Hardware noise

Table: Summary of canonical quantum noise models.

<table>
<thead>
<tr>
<th>Quantum noise mixtures (Pauli errors)</th>
<th>Quantum noise channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pauli-X type Bit flip noise</td>
<td>Amplitude damping noise (related to T1 time)</td>
</tr>
<tr>
<td>Pauli-Z type Phase flip noise</td>
<td>Phase damping noise (related to T2 time)</td>
</tr>
<tr>
<td>Combinations Symmetric / asymmetric depolarizing noise</td>
<td>Generalized amplitude damping</td>
</tr>
</tbody>
</table>

Simulation technique

- Can model as probabilistic ensembles of state vectors
- Requires density matrix representation
The amplitude damping channel leaves $|0\rangle$ alone while probabilistically flipping $|1\rangle$. It has two elements:

\[
E_0 = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{1 - \gamma} \end{bmatrix}, \quad E_1 = \begin{bmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{bmatrix}
\]
Hardware noise

1. Decoherence error
2. Gate error (imprecise control of single qubit, two qubit gates)
3. Measurement error

<table>
<thead>
<tr>
<th>Technology</th>
<th>Coherence Time (s)</th>
<th>1-Qubit Gate Latency (s)</th>
<th>2-Qubit Gate Latency (s)</th>
<th>1-Qubit Gate Fidelity (%)</th>
<th>2-Qubit Gate Fidelity (%)</th>
<th>Mobile</th>
</tr>
</thead>
</table>

Table 1. Metrics for various quantum technologies. * Nuclear/Electron Hybrid

Figure: Credit: [Resch and Karpuzcu, 2019]

But what about correlated noise events?
Compiling from a high-level program to hardware

Extreme device/resource constraints:

1. Native gate set
2. Device topology
3. Hardware noise
4. Parallelism constraints

Generic strategies:

▶ Minimize two-qubit gates while respecting native gate set and topology
▶ Minimize quantum circuit depth while completing whole circuit
▶ Maximize parallelism while avoiding crosstalk noise
Parallelism constraints

Some types of gates commute, so we can move earlier or later.
Parallelism constraints

1. Amount of parallelism available in the instruction stream
2. Achievable parallelism in the control microarchitecture ("each student gets one coaxial input")
3. Safe parallelism despite crosstalk due to spatiotemporal and spectral overlap
Compiling from a high-level program to hardware

Goals:

1. Correctness: maximizing probability of success!
2. Ease of programming?
3. Compatibility between hardware implementations?

Extreme device/resource constraints:

1. Native gate set
2. Device topology
3. Hardware noise
4. Parallelism constraints
Primary sources

- [Ding and Chong, 2020, Chapters 4,6,7]
- [Córcoles et al., 2020, Section III.B]
- [National Academies of Sciences, Engineering, and Medicine, 2019, Chapter 6.5]
- [Martonosi and Roetteler, 2019, Chapter 6]


