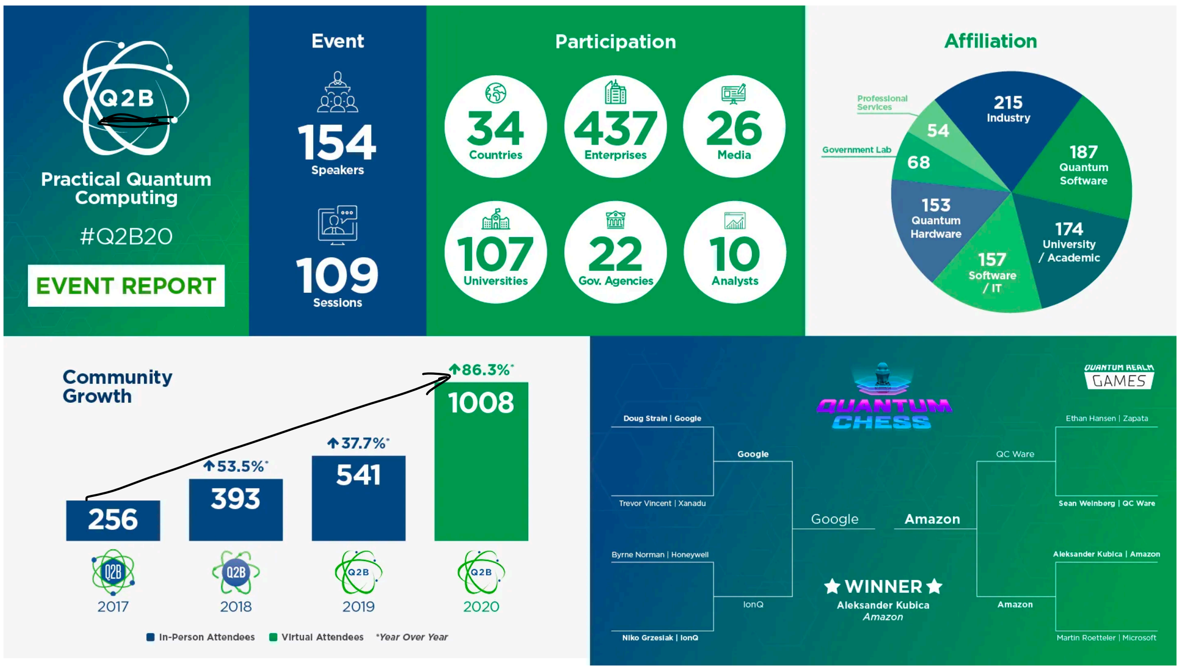
1. Basics of Quantum Computing

Isaac H. Kim (UC Davis)

Goal

- Review of quantum simulation algorithms, with a focus on state-of-the-art methods and key ideas.
- I will assume that you are comfortable with the standard quantum mechanics, e.g., braket notation, Born rule, etc.
- Today: Basics of Quantum Computation

Quantum Computing



Speed of existing quantum computers

- Using your laptop, you can perform a 64-bit integer addition in less than a nanosecond.
- Quantum computers available today need at least 10ns~100ns, even microseconds, to apply a single *elementary gate*.
- One would need 10s of layers of such gates to perform the integer addition, leading to at least 100~10,000-fold slowdown compared to your laptop.
- Everybody is saying that quantum computer is more efficient than a classical computer. What’s happening here?

Asymptotic Scaling

- While the existing quantum computers are small and slow, technology will eventually advance, making them larger and faster.
- In that regime, it is important to understand the asymptotic scaling of the time needed to do the computation.

Example: Shor’s algorithm

$N = 190 \cdots 7 = \underbrace{1 \cdots 01}_{n}$

Shor's also runs in $O(n^3)$ time

- Peter Shor famously came up with the factoring algorithm in 1994.
- This algorithm uses at most cn^3 *quantum* gates, where c is a numerical constant and n is the number of bits in the number you want to factorize.
- On the other hand, the best known method using a classical computer requires a number of gates that scales at least $c'' \exp(c' n^{1/3})$.
- To compare their speed in real time, we can multiply by the time to execute the gates. But this only changes the constant.
- Eventually, as n grows, the time needed using quantum gates will be much smaller than the time needed using only classical gates.

Q: $T_Q \, c n^3$

C: $T_C \, c'' \exp(c' n^{1/3})$

$T_Q \gg T_C$

Asymptotics

O(10^2)   O(10^5) : "Physics" Big-O notation

- In computer science, it is very common to use Big-O notations. This is different from the physics big-O notation.
- $f(n) = O(g(n))$: There is a constant $c > 0$ such that for a sufficiently large $n \geq n_0$, for some constant n_0, $f(n) \leq cg(n)$. : Upper bound
  $f(n) = 3n^3 - 2n^2 + n + 1$
  $\leq c' n^3 \Rightarrow f(n) = O(n^3)$
- $f(n) = \Omega(g(n))$: $f(n) \geq cg(n)$: Lower bound
- $f(n) = \Theta(g(n))$: $c'g(n) \leq f(n) \leq cg(n)$: Close to physic big-O notation
- $f(n) = o(g(n))$: $\displaystyle\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. : "Subleading"

$f(n) = n^{1/2}$ $f(n) = o(n)$

Runtime is $3n + n^{1/2} = 3n + o(n)$

Asymptotics: Short summary

- $f(n) = O(g(n))$: $f(n) \leq cg(n)$.
- $f(n) = \Omega(g(n))$: $f(n) \geq cg(n)$.
- $f(n) = \Theta(g(n))$: $c'g(n) \leq f(n) \leq cg(n)$
- $f(n) = o(g(n))$: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.

Time complexity

- The time complexity of an algorithm quantifies the amount of time needed to run the algorithm.
- Obviously this would be a function of the input size n, and in general will be a complicated function.
- The big-O notation will be useful to understand the asymptotic scaling of the time complexity.

Shor's also has a (Quantum) time complexity of $O(n^3)$

Efficiency

- An algorithm is *efficient* if its time complexity (and space complexity) is $O(n^k)$ for some $k < \infty$.

  ex) An algorithm with the time complexity of $10^{10^{10}} n^{10^{23}}$ is efficient, even though this is obviously not practical.

- An algorithm A is more efficient than algorithm B if A has a smaller time complexity than B.

  ex) $10^{100}n$ is more efficient than $(1 + 10^{-100})^n$.

$O(n^{\ln(n)})$

Quantum vs. Classical computing: Similarities

- Bits: $0, 1$
- Elementary gates: AND, NOT, NAND, …

Universal

x	y	NAND(x,y)
0	0	1
0	1	1
1	0	1
1	1	0

Quantum vs. Classical computing: Similarities

- Qubits: $|0\rangle, |1\rangle$
- Elementary gates: 1- and 2-qubit gates $\rightarrow$ Unitary transformations

Hilbert space : $\mathcal{H} = \bigotimes_{i=1}^{n} \mathcal{H}_i$ $\quad \dim(\mathcal{H}_i) = 2$

Canonical basis set : $\{ |x\rangle : x$ is a n-bit string. $\}$

ex) $x = \underbrace{11\cdots 010\cdots 11}_{n}$ $\quad x = x_1 \cdots x_k \cdots x_n \quad x_1, \cdots x_n \in \{0,1\}$

1-qubit gate U acting on qubit k : $U\,|x_1 \cdots x_k \cdots x_n\rangle$
$= |x_1 \cdots x_{k-1}\rangle \left(U|x_k\rangle\right) |x_{k+1} \cdots x_n\rangle$

Quantum vs. Classical computing: Differences

$$U: \quad UU^\dagger = U^\dagger U = I \qquad U^\dagger = U^{-1}$$

- Every quantum gate is unitary, hence reversible.
- Not every classical gate is unitary.
- **Q1: Can quantum computers do everything that classical computers can do?**
- Q2: Can quantum computers provide speedups?

Input		Output
x	y	NAND(x,y)
0	0	1
0	1	1
1	0	1
1	1	0

Reversible computation

- It turns out that reversible computation is possible. (Bennett, 1973)
- Basic idea: Use Toffoli gates : Acts on 3 bits

x	y	z
0	0	0
0	1	0
1	0	0
1	1	0
0	0	1
0	1	1
1	0	1
1	1	1

$\xrightarrow{\text{Toffoli}}$

x'	y'	z'
0	0	0
0	1	0
1	0	0
1	1	1
0	0	1
0	1	1
1	0	1
1	1	0

$x, y, z \rightarrow x, y, z \oplus \mathrm{AND}(x,y)$

↳ Addition mod 2.

$x, y, z \rightarrow x, y, z \oplus \mathrm{AND}(x,y)$

$\rightarrow x, y, z \oplus \mathrm{AND}(x,y) \oplus \mathrm{AND}(x,y) = x, y, z$

$x, y, 0 \rightarrow x, y, \mathrm{AND}(x,y)$

NAND is AND followed by NOT

- Conclusion: Any efficient classical algorithm can be made reversible whilst maintaining its efficiency.

1 NAND ← 1 Toffoli, 1 NOT

Quantum computation

- Both Toffoli gate and NOT gate can be implemented using 1- and 2-qubit gates.
- Therefore, any efficient classical computation can be done efficiently on a quantum computer!

- But, quantum computer can do more…

Reversible computation, in superposition

$$\boxed{\sum_x \alpha_x |x\rangle |0\rangle \to \sum_x \alpha_x |x\rangle |f(x)\rangle}$$: Possible

$$\sum_x \alpha_x |x\rangle \nrightarrow \sum_x \alpha_x |f(x)\rangle$$: Generally impossible

$x \quad y \quad NAND(x,y)$

$x, y, 0 \xrightarrow{\text{Toffoli}} x, y, AND(x,y)$

Extra

$|x\rangle|y\rangle|0\rangle \xrightarrow{\text{Toffoli }(Q)} |x\rangle|y\rangle|AND(x,y)\rangle$

$x,y \in \{0,1\}$

$|x\rangle|y\rangle|z\rangle|0\rangle|0\rangle \to |x\rangle|y\rangle|z\rangle|AND(x,y)\rangle|0\rangle \to |x\rangle|y\rangle|z\rangle|AND(x,y)\rangle|AND(z, AND(x,y))\rangle$

Q: What about the intermediate results in the computation?

$\sum_{x,y,z} \alpha_{x,y,z} |x\rangle|y\rangle|z\rangle|0\rangle|0\rangle \to \sum_{x,y,z} \alpha_{x,y,z} |x\rangle|y\rangle|z\rangle|AND(x,y)\rangle|AND(z, AND(x,y))\rangle$ $\Rightarrow |0\rangle$

Trick: Uncomputation

- Goal: Implement $|x\rangle|0\rangle \to |x\rangle|f(g(x))\rangle$ using $U_f(|x\rangle|y\rangle) = |x\rangle|y+f(x)\rangle$ and $U_g(|x\rangle|y\rangle) = |x\rangle|y+g(x)\rangle$.

$$|x\rangle|0\rangle|0\rangle \xrightarrow{U_g} |x\rangle|g(x)\rangle|0\rangle \xrightarrow{U_f} |x\rangle|g(x)\rangle|f(g(x))\rangle \xrightarrow{U_g^{-1}} |x\rangle|0\rangle|f(g(x))\rangle$$

Reversible computation, in superposition

$$\sum_x \alpha_x |x\rangle |0\rangle \to \sum_x \alpha_x |x\rangle |f(x)\rangle$$

Fact: Computing $f(x)$ in superposition can be done efficiently on a quantum computer if $f(x)$ is efficiently computable on a classical computer.

Quantum vs. Classical computing: Differences

- Every quantum gate is unitary, hence reversible.
- Not every classical gate is unitary.
- Q1: Can quantum computers do everything that classical computers can do?
- **Q2: Can quantum computers provide speedups?**

Quantum speedups

- Exponential speedups: Factoring (Shor), **Quantum simulation**, …?
- Polynomial speedups: Database search (Grover), Optimization, Monte Carlo simulation, …

→ Modest

Classical: $O(2^n)$

Quantum: $O(2^{n/2})$

Summary

- Anything you can do classically efficiently, you can do quantumly efficiently as well.
- There are quantum algorithms which are exponentially faster than classical algorithms.
- Next lecture: I will be more explicit about the elementary gates.