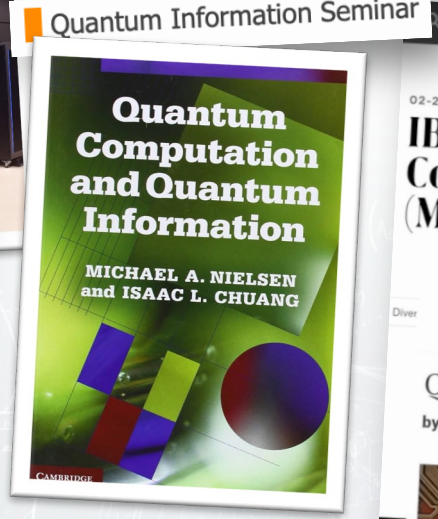
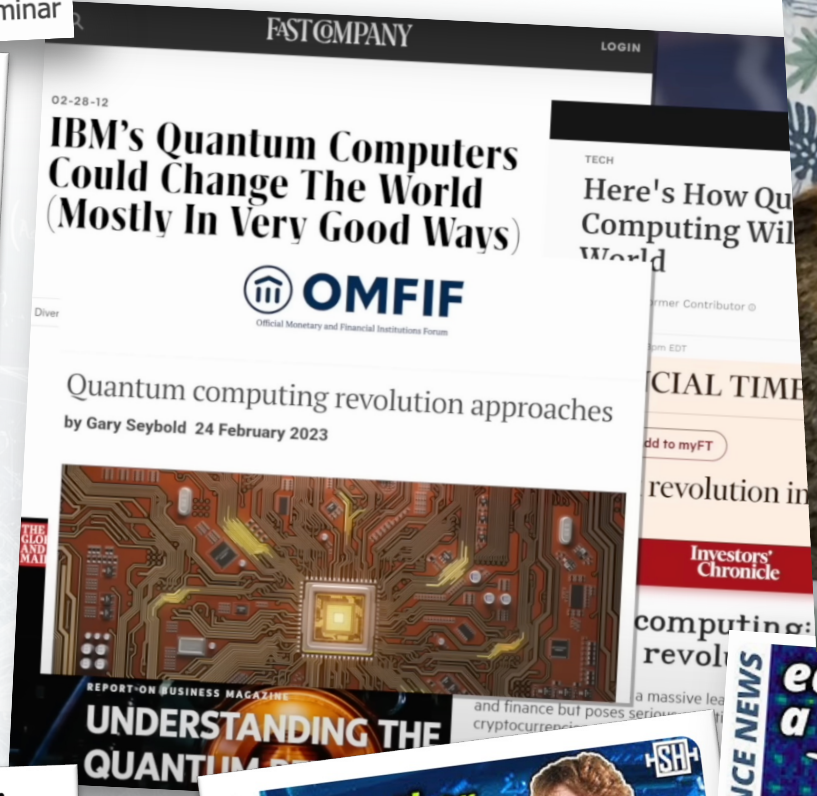
I ain't no physicist but I know what matters.
– Popeye the Sailor

Quantum mechanics: Real Black Magic Calculus
– Albert Einstein

`lecture: 8 – quantum circuits (part two)`

Scott Aaronson: Quantum Computing | Lex Fridman Podcast #72





it's complicated

Quantum error correction

Threshold theorem

Physical and logical qubits

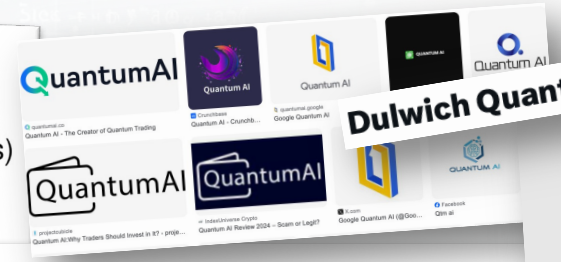
New Quantum Computing Approach Makes Rapid Progress

@SabineHossenfelder

The Problem With Science Communication

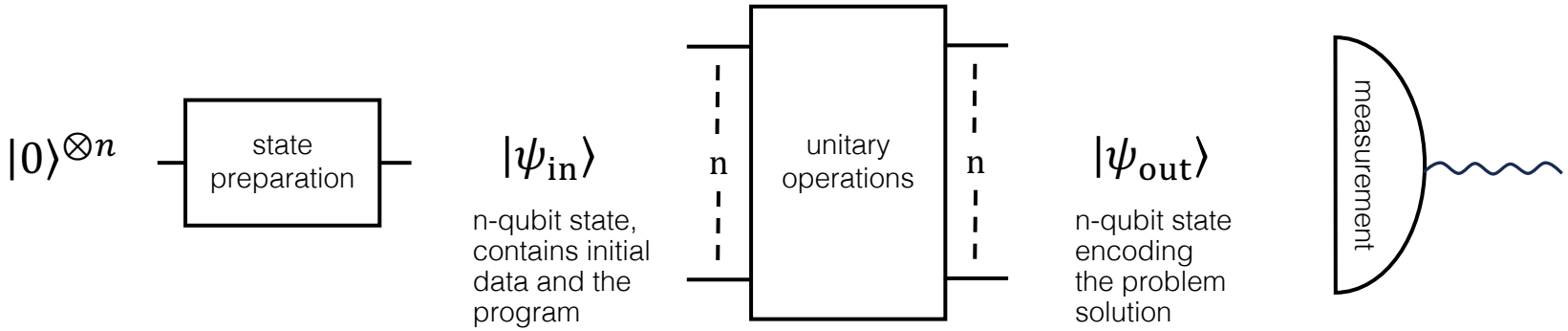
@Veritasium

- molecule spins manipulated with NMR (old...)
- ion traps (3-Qubit Grover search in 2017)
- NV centers, quantum dots (integrable with semiconductor devices)
- optical cluster states
- superconducting qubits (low-temp. Josephson tunnel junctions)



drashk hor online Quantum Espresso and QuantumATK courses - Dear prof I suggest the two courses, one course is about qua

A *general quantum computer* should be able to read, process, and save quantum states, preserving *superposition* and *entanglement*.



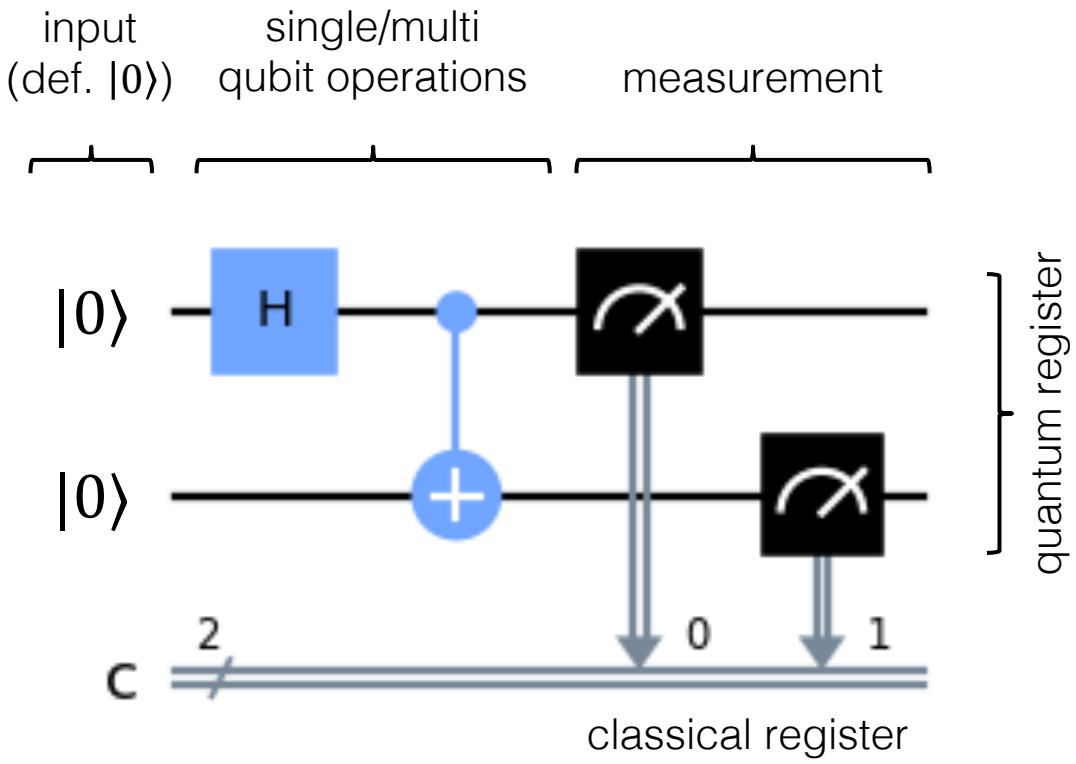
Can it compute anything that a classical computer cannot?

No! It's just matrix multiplication after all.

But maybe it can do it faster? **To simulate n qubits classically you need $\mathcal{O}(2^n)$ memory which grows rapidly with n.**

To talk about computational complexity, we define basic operations as *gates*. Instead of being general, we'll use them to construct circuits that solve specific problems.

a basic quantum circuit:



qiskit intro

```
from qiskit import (ClassicalRegister,
                    QuantumRegister,
                    QuantumCircuit,
                    transpile)

q = QuantumRegister(2, name="qubit")
c = ClassicalRegister(2, name="bit")
circuit = QuantumCircuit(q, c)

from qiskit_aer import Aer

S_simulator = (
    Aer.backends(name='statevector_simulator'))[0]

# circuit simulation
job = S_simulator.run(
    transpile(
        circuit, S_simulator
    )
)

print(job.result().get_statevector())

# Statevector([1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
#             dims=(2, 2))
```

one-qubit gates:

identity $\quad \mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Hadamard $\quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad |0/1\rangle \rightarrow \frac{|0\rangle \pm |1\rangle}{\sqrt{2}}$

X (NOT) $\quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad |0\rangle \rightarrow |1\rangle, |1\rangle \rightarrow |0\rangle$

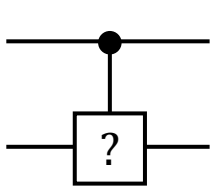
Z $\quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad |0\rangle \rightarrow |0\rangle, |1\rangle \rightarrow -|1\rangle$

PHASE $\quad R_\varphi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} \quad |0\rangle \rightarrow |0\rangle, |1\rangle \rightarrow e^{i\varphi}|1\rangle$

Rotation on the Bloch sphere around the x/y/z axis:

$$R_x(\vartheta) = \begin{pmatrix} \cos\frac{\vartheta}{2} & -i\sin\frac{\vartheta}{2} \\ -i\sin\frac{\vartheta}{2} & \cos\frac{\vartheta}{2} \end{pmatrix} \quad R_y(\vartheta) = \begin{pmatrix} \cos\frac{\vartheta}{2} & -\sin\frac{\vartheta}{2} \\ \sin\frac{\vartheta}{2} & \cos\frac{\vartheta}{2} \end{pmatrix} \quad R_z(\vartheta) = \begin{pmatrix} e^{-\frac{i\vartheta}{2}} & 0 \\ 0 & e^{\frac{i\vartheta}{2}} \end{pmatrix}$$

two-qubit control gates:



control qubit

target qubit

control-NOT $\quad CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

$|00\rangle \rightarrow |00\rangle$
$|01\rangle \rightarrow |01\rangle$
$|10\rangle \rightarrow |11\rangle$
$|11\rangle \rightarrow |10\rangle$

control-Z $\quad CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$|00\rangle \rightarrow |00\rangle$
$|01\rangle \rightarrow |01\rangle$
$|10\rangle \rightarrow |10\rangle$
$|11\rangle \rightarrow -|11\rangle$

control-PHASE $\quad CU_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix}$

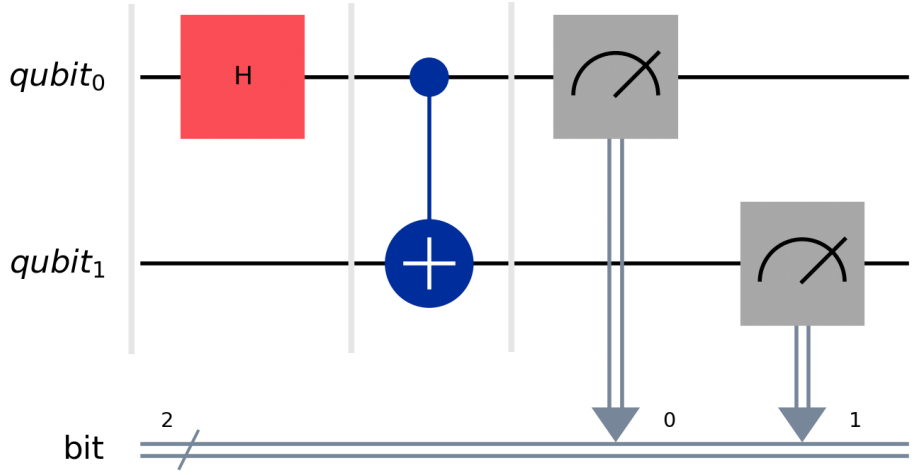
$|00\rangle \rightarrow |00\rangle$
$|01\rangle \rightarrow |01\rangle$
$|10\rangle \rightarrow |10\rangle$
$|11\rangle \rightarrow e^{i\varphi}|11\rangle$

qubit swap $\quad SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$|00\rangle \rightarrow |00\rangle$
$|01\rangle \rightarrow |10\rangle$
$|10\rangle \rightarrow |01\rangle$
$|11\rangle \rightarrow |11\rangle$

```python
q = QuantumRegister(2, name="qubit")
c = ClassicalRegister(2, name="bit")
circuit = QuantumCircuit(q, c)

circuit.h(q[0])
circuit.cx(q[0], q[1])

# adding the instruction to measure
circuit.measure(q, c)

# actual measurement simulation
M_simulator = (
    Aer.backends(name='qasm_simulator'))[0]

job = M_simulator.run(
    transpile(circuit, M_simulator)
)

print(job.result().get_counts())

# {'11': 496, '00': 528}

# visualising the circuit
print(circuit)
# or
import matplotlib.pyplot as plt
circuit.draw('mpl')
plt.show()
```

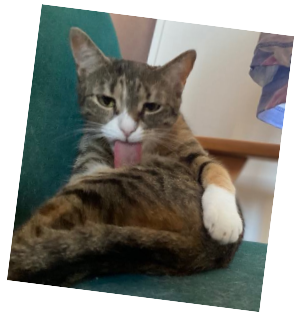
$$\begin{pmatrix}1\\0\\0\\0\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\1\\0\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$$



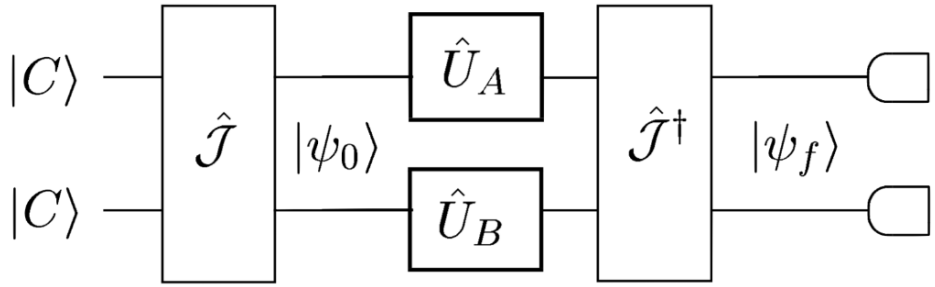

```python
# helpful custom functions (download on the website)
import Our_Qiskit_Functions as oq

# wraps measurement simulation and get_counts()
print(oq.Measurement(circuit, shots=100))
# 52|11>   48|00>

# let's delete the final two measurement instructions
circuit.data = circuit.data[:-2]

# wraps state simulation and get_statevector()
print(oq.Wavefunction(circuit))
# 0.70711  |00>   0.70711  |11>
```

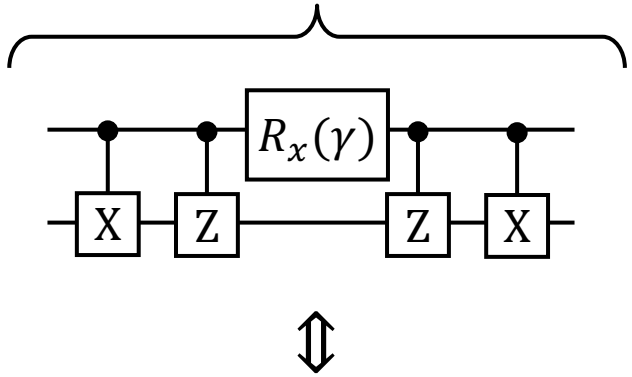
QPD circuit

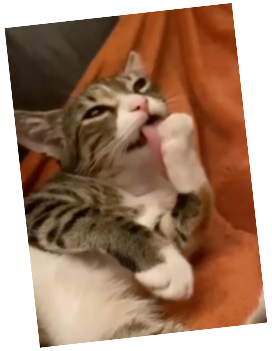
Last week: vectors, matrices, normalization, missing minus signs, tensor products, mathematical rigour (actually useful though)



Now:







$$\hat{\mathcal{J}} = \exp(-i\,\gamma\,\hat{D}\otimes\hat{D}\,/\,2) \qquad \hat{D} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

```python
# doing it by hand
def J_gate(gamma, adj=False):
    J_circuit = QuantumCircuit(QuantumRegister(2),
                               name=('J' if not adj
                                     else 'J+'))
    J_circuit.cx(0, 1)
    J_circuit.cz(0, 1)
    J_circuit.rx((gamma if not adj else -gamma), 0)
    J_circuit.cz(0, 1)
    J_circuit.cx(0, 1)
    return J_circuit.to_gate()

# or simply
from qiskit.circuit.library import UnitaryGate
def J_gate(gamma, adj=False):
    return UnitaryGate(
        sp.linalg.expm(
            (-1)**adj * (-1j) * gamma / 2 * np.kron(D, D)
        ), label=('J' if not adj else 'J+')
    )

gamma = np.pi/3
circuit.append(J_gate(gamma, adj=False), q)
circuit.y(q[0]) # U_A = D = 1j * Y
circuit.z(q[1]) # U_B = Q = 1j * Z
circuit.append(J_gate(gamma, adj=True), q)

print(circuit)
print(oq.Wavefunction(circuit))
```

Quantum parallelism

Let $f: \{0,1\}^n \to \{0,1,\ldots,d-1\}$ be a function from n-digit bitstrings to integers from 0 to some d-1.

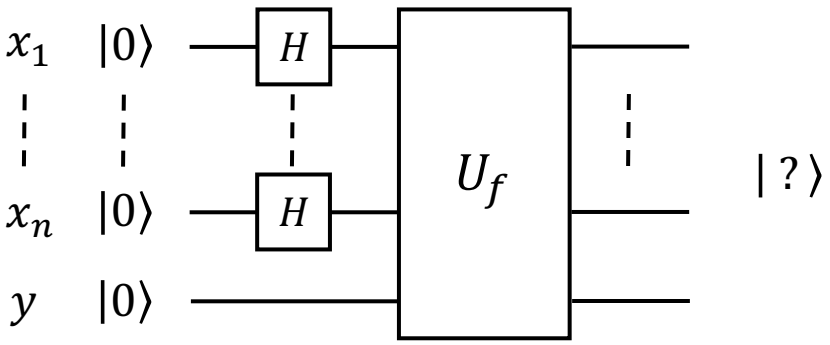
How to encode f into a unitary operation? One idea: construct a unitary U_f such that

$$U_f|x_1\ x_2\ \ldots x_n\rangle \otimes |y\rangle = |x_1\ x_2\ \ldots x_n\rangle \otimes |y \oplus f(x_1, x_2, \ldots, x_n)\rangle$$

- $|x_1\ x_2\ \ldots x_n\rangle$: n-qubit state
- $|y\rangle$: qudit (d-dim. state)
- $\oplus$: mod d
- $f(x_1, x_2, \ldots, x_n)$: function value for $x = x_1, \ldots, x_n$

One can check that this is a valid unitary (preserves scalar products). What to do with it?

Quantum parallelism



$$|0\rangle^{\otimes n} \otimes |0\rangle \xrightarrow{H^{\otimes n} \otimes \mathbb{I}} \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right]^{\otimes n} \otimes |0\rangle$$

"the most democratic state" $|s\rangle$ - superposition of all possible input states

$$= \left(\frac{1}{\sqrt{2}}\right)^n \left(\sum_{x_1,\ldots,x_n=0}^{1} |x_1, \ldots, x_n\rangle\right) \otimes |0\rangle$$

$$\xrightarrow{U_f} \left(\frac{1}{\sqrt{2}}\right)^n \sum_{x_1,\ldots,x_n=0}^{1} |x_1, \ldots, x_n\rangle \otimes |f(x_1, \ldots, x_n)\rangle$$

superposition of function values for all possible inputs, with a single evaluation of U_f!

But what now? If we just measure the resulting state,

the output will be a randomly drawn input and its function value.

We need to somehow process the state in a clever way

in order to benefit from parallelism.

Quantum algorithms

- *Deutsch's, Deutsch-Jozsa*

  Determining whether a function is constant or balanced. Simple but rather useless.

- *Shor's*

  Factoring large numbers. Useful but rather complicated.

-
-
-

topics for final presentations?

Today: **Grover's algorithm — Unstructured Database Search**

Let $f: \{0, 1, \ldots, N-1\} \to \{0, 1\}$ with $N = 2^n$. We are told that this function evaluates to 0 for all inputs except for one, denoted ω for "winner". How to find ω?

Interpretation: searching in a database for an entry that fulfills a specific condition.

Classically you need to check the function value for $N/2$ inputs on average: complexity $\mathcal{O}(N)$.

Grover's algorithm — Unstructured Database Search

Let $f: \{0, 1, \ldots, N-1\} \to \{0, 1\}$, with $N = 2^n$, such that only one $x \equiv w$ satisfies $f(x) = 1$:

$$\exists!\ w \mid f(w) = 1 \quad \wedge \quad \forall\, x \neq w \mid f(x) = 0.$$

We are given access to a subroutine — the „oracle" $U_w = \mathbb{I} - 2|\omega\rangle\langle\omega|$ — that implements f in the following way (a bit differently than U_f from three slides ago but also valid):

$$U_w|x\rangle = -|x\rangle \quad \text{for} \quad x = w$$

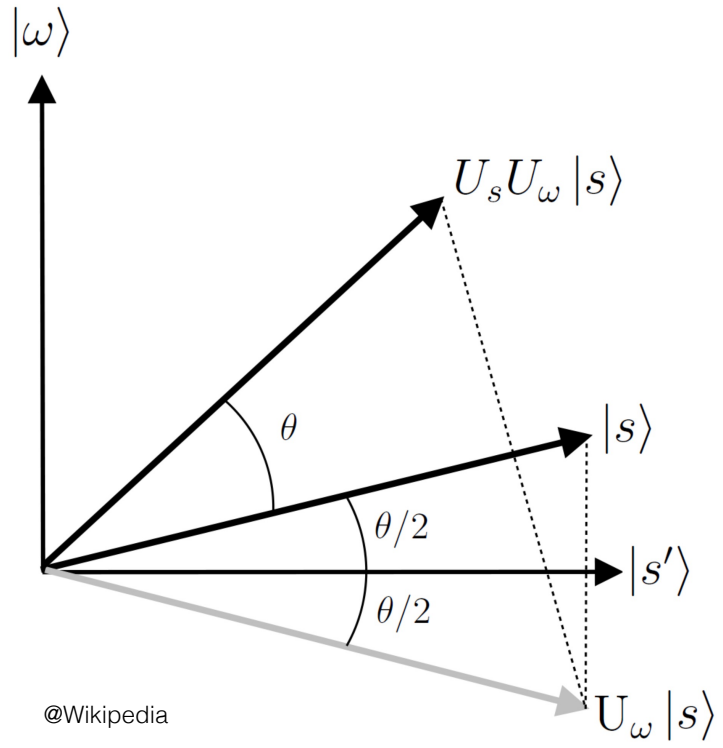
$$U_w|x\rangle = |x\rangle \quad \text{for} \quad x \neq w$$

Here $|x\rangle$ is shorthand for $|x_1, \ldots, x_n\rangle$, i.e., a n-qubit state where x_i are the digits of the binary representation of $x \in \{0, \ldots, N-1\}$.

Algorithm complexity depends on how many times we need to apply the oracle to determine w.

Grover's algorithm — Unstructured Database Search

Geometric intuition for how it works:



@Wikipedia

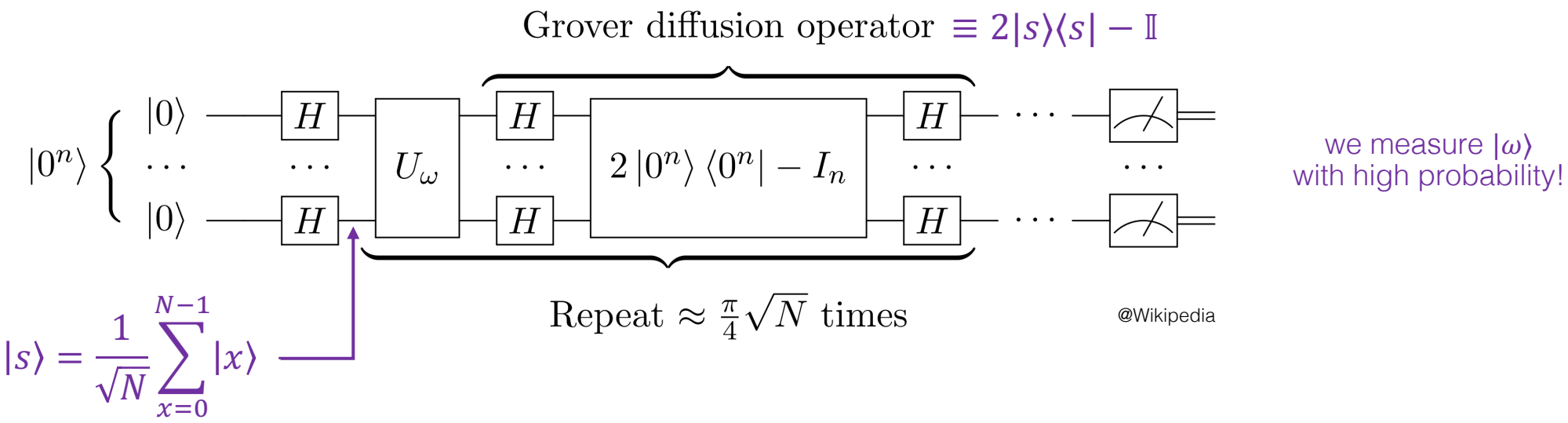
The n-qubit system starts in state $|s\rangle$. It lives in a subspace spanned by $|\omega\rangle$ and $|s'\rangle = \frac{1}{\sqrt{N-1}} \sum_{x \neq w} |x\rangle$: $|s\rangle = \frac{1}{\sqrt{N}} |\omega\rangle + \frac{\sqrt{N-1}}{\sqrt{N}} |s'\rangle$.

It can be verified that the application of the oracle U_ω and the diffusion operator $U_s = 2|s\rangle\langle s| - \mathbb{I}$ rotates the n-qubit state within the subspace by an angle $\theta = 2 \arcsin \frac{1}{\sqrt{N}}$.

It follows that after approximately $\pi\sqrt{N}/4$ iterations of $U_s U_\omega$, the state is rotated *close* to $|\omega\rangle$, which is then likely to be measured and thus determined.

Therefore, the algorithm has complexity $\mathcal{O}(\sqrt{N})$, quadratically improving the classical algorithm. It has, however, a non-unit probability of success, with the error probability approaching 0 with growing N.

Grover's algorithm — Unstructured Database Search



we measure $|\omega\rangle$
with high probability!

Task 1 [0.3 pts]**:** Implement the oracle for $n = 3$ qubits and $\omega = 010$. Show that it works.

Task 2 [0.7 pts]**:** Implement two steps of Grover's algorithm. Plot a histogram of the measurement outcomes.

Grover's algorithm — Unstructured Database Search

Extra task [0.2 pts]**:** Run your code on a real quantum computer. You can build one yourself (harder) or use the IBM Quantum Platform service (easier). For the latter, you will need to create an IBM Quantum account and start a subscription – the Open Plan is free and gives you 10 minutes of quantum hardware runtime per month.

Links that may be useful (working as of 26.04.2024):

https://quantum.ibm.com
https://docs.quantum.ibm.com/start/install
https://www.ibm.com/quantum/pricing
https://docs.quantum.ibm.com/run/primitives-get-started#get-started-with-sampler

Grover's algorithm — hints

Oracle:

- we need $V_\omega = X^{1-\omega_0} \otimes \cdots \otimes X^{1-w_{N-1}}$ which flips a qubit only when $\omega_i = 0$; also $V_\omega^2 = \mathbb{I}$
- the $[\mathrm{CCZ}]$ gate flips the sign for $\omega = 111$ (all qubits are 1); it's available in Qiskit
- the oracle is then $U_\omega = V_\omega\, [\mathrm{CCZ}]\, V_\omega$ (check!)

Grover diffusion operator:

- $X^{\otimes n}\, [\mathrm{CCZ}]\, X^{\otimes n}$ flips the sign only for $|0\rangle \implies X^{\otimes n}\, [\mathrm{CCZ}]\, X^{\otimes n} = \mathbb{I} - 2|0\rangle\langle 0|$
- therefore $H^{\otimes n} X^{\otimes n}\, [\mathrm{CCZ}]\, X^{\otimes n} H^{\otimes n} = \mathbb{I} - 2|s\rangle\langle s| = -U_s$

Final diagram for one iteration and $\omega = 001$:

- O - oracle, D - diffusion operator