Lecture 1:
Introduction to quantum computing, qubits, gates and superposition

Alexei Bazavov
Michigan State University

FRIB-TA Summer School
MSU Jun 20 — 22, 2022

Introduction

- What is quantum computing?
- Why do we need it?
- How can we make it work for the problems we want to solve?

Introduction

- What is quantum computing?

  A type of computing where the programming model follows principles of quantum mechanics

Introduction

- What is quantum computing?

  A type of computing where the programming model follows principles of quantum mechanics

Nielsen, Chuang, Ch. 2

Postulate 1: Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the *state space* of the system. The system is completely described by its *state vector*, which is a unit vector in the system's state space.

Introduction

- What is quantum computing?

  A type of computing where the programming model follows principles of quantum mechanics

Nielsen, Chuang, Ch. 2

Postulate 1: Associated to any isolated physical system is a complex vector space with

syste

vecto

Postulate 2: The evolution of a *closed* quantum system is described by a *unitary transformation*. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2,

$$|\psi'\rangle = U|\psi\rangle\,. \tag{2.84}$$

Introduction

- What is quantum computing?

  A type of computing where the programming model follows principles of quantum mechanics

Nielsen, Chuang, Ch. 2

Postulate 1: Associated to any isolated physical system is a complex vector space with

Postulate 2: The evolution of a *closed* quantum system is described by a *unitary trans*

Postulate 3: Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle\,, \tag{2.92}$$

and the state of the system after the measurement is

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}\,. \tag{2.93}$$

Introduction

- What is quantum computing?

  A type of computing where the programming model follows principles of quantum mechanics

Nielsen, Chuang, Ch. 2

Postulate 1: Associated to any isolated physical system is a complex vector space with

Postulate 2: The evolution of a *closed* quantum system is described by a *unitary trans*

Postulate 3: Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle} \tag{2.93}$$

Postulate 4: The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$. (2.92)

Classical computing

Typical problems we deal with:

- Combinatorial problems
- Evaluation of functions
- Root finding
- Numerical differentiation and integration
- Solving ODEs and PDEs
- Optimization
- Linear algebra (linear systems, eigenproblems)
- Stochastic (Monte Carlo) sampling

Classical computing

Why has classical computing been successfully applied to so many problems?

- Hardware improvements in 20th century
- Hundreds or even thousands of years of algorithm development

- Runge-Kutta method for ODEs — early 20th century
- Gauss elimination — 19th century (ideas around -3rd century!)
- Euler method for ODEs — 18th century
- Stochastic “integration” (Buffon’s needle) — 18th century
- Newton-Raphson method — 17th century (ideas around 1st century!)

$$x = \sqrt{a} \quad \Rightarrow \quad x^2 - a = 0 \quad \Rightarrow \quad x_0 = a, \quad x_{n+1} = (x_n + a/x_n)/2$$

Square root by iteration

```c
#include <stdio.h>
#include <math.h>

int main() {
  // parameters
  int depth = 100; float a = 2., eps = 1e-4;
  // storage
  int i; float x, xp;

  // initialization
  x = a; i = 0;

  // evolution
  do {
    xp = x; x = ( x + a/x ) / 2;
    i++;
  } while( fabs( x-xp ) > eps && i < depth );

  // read-out
  printf( "sqrt(%g) ~ %g\n", a, x );

  return 0;
}
```

Square root by iteration

```c
#include <stdio.h>
#include <math.h>

int main() {
  // parameters
  int depth = 4; float a = 2.;
  // storage
  int i; float x;

  // initialization
  x = a; i = 0;

  // evolution
  do {
    x = ( x + a/x ) / 2;
    i++;
  } while( i < depth );

  // read-out
  printf( "sqrt(%g) ~ %g\n", a, x );

  return 0;
}
```

Square root by iteration

```c
#include <stdio.h>
#include <math.h>

int main() {
  // parameters
  float a = 2.;
  // storage
  float x;

  // initialization
  x = a;

  // evolution
  x = ( x + a/x ) / 2;
  x = ( x + a/x ) / 2;
  x = ( x + a/x ) / 2;
  x = ( x + a/x ) / 2;

  // read-out
  printf( "sqrt(%g) ~ %g\n", a, x );

  return 0;
}
```

Square root by iteration

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5    // parameters
6    float a = 2.;
7    // storage
8    float x;
9
10   // initialization
11   x = a;
12
13   // evolution
14   x = ( x + a/x ) / 2;
15   x = ( x + a/x ) / 2;
16   x = ( x + a/x ) / 2;
17   x = ( x + a/x ) / 2;
18
19   // read-out
20   printf( "sqrt(%g) ~ %g\n", a, x );
21
22   return 0;
23 }
```

```
01000000000000000000000000000000
00111111100000000000000000000000
00111111101101010101010101010110
00111111101101010000010100000101
00111111101101010000010011110011
```

degree(s) of freedom

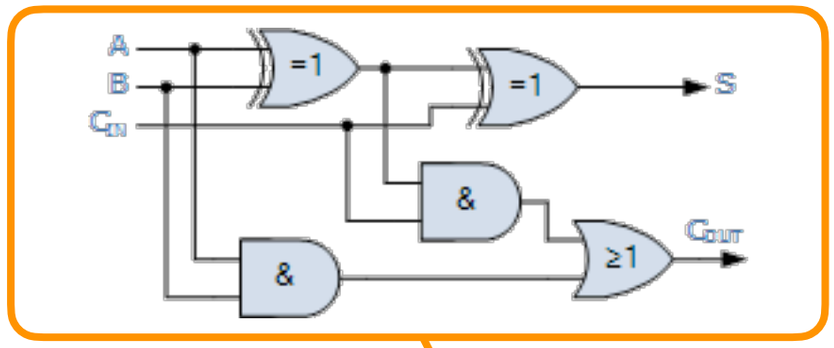
initial state preparation

evolution

measurement

Square root by iteration

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5    // parameters
6    float a = 2.;
7    // storage
8    float x;
9
10   // initialization
11   x = a;
12
13   // evolution
14   x = ( x + a/x ) / 2;
15   x = ( x + a/x ) / 2;
16   x = ( x + a/x ) / 2;
17   x = ( x + a/x ) / 2;
18
19   // read-out
20   printf( "sqrt(%g) ~ %g\n", a, x );
21
22   return 0;
23 }
```




```
7f: f3 0f 10 45 fc  movss -4(%rbp), %xmm0
84: f3 0f 5e 45 f8  divss -8(%rbp), %xmm0
89: f3 0f 58 45 f8  addss -8(%rbp), %xmm0
8e: f3 0f 10 0d 00 00 00 00   movss (%rip), %xmm1
96: f3 0f 5e c1   divss %xmm1, %xmm0
9a: f3 0f 11 45 f8  movss %xmm0, -8(%rbp)
```

- Why do we need quantum computing?

Scaling

- There are known examples such as factorization into primes, traveling salesman and other problems that require non-polynomial computational effort
- As a physics example, consider *classical* Ising model:

$$E(\{s\}) = -J \sum_{\langle ij \rangle} s_i s_j, \quad s_i \in \{-1,1\}$$

$$Z = \sum_{\{s\}} \exp\left(-\frac{E(\{s\})}{k_B T}\right)$$

- The average energy at a given temperature T:

$$\langle E \rangle(T) = \frac{1}{Z} \sum_{\{s\}} E(\{s\}) \exp\left(-\frac{E(\{s\})}{k_B T}\right)$$

Scaling

- To calculate $\langle E \rangle$ on a N^3 lattice requires summing over 2^{N^3} spin configurations
- Assume that $N = 10$ (a very modest lattice) and it takes n floating point operations to evaluate the energy of the configuration, then we need $n \cdot 2^{1000} \sim 10^{301} n$ floating point operations
- The most powerful supercomputer today (Frontier, top500) can operate at about 1 ExaFlop/s = 10^{18} Flop/s, so it would take about $10^{283} n$ seconds to evaluate the exact solution (the age of the universe is about 4.3×10^{17} s)
- For classical statistical problems and problems that can be reduced to those (such as Euclidean lattice gauge theory) stochastic methods can often help to obtain approximate solutions

Scaling

- For quantum problems (e.g. time evolution of a *quantum* Ising model would require exponentiating a $2^{N^3} \times 2^{N^3}$ matrix) or where Monte Carlo is inapplicable (often due to the *sign problem*) classical algorithms scale *exponentially* with the number of degrees of freedom

- Feynman (1982):

  "The rule of simulation that I would like to have is that the number of computer elements required to simulate a large physical system is only to be proportional to the space-time volume of the physical system. I don't want to have an explosion."

Some history

- P. Benioff (1980), “The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines”
- R.P. Feynman (1982), “Simulating Physics with Computers”:

  “Can a *universal* quantum simulator be built?”
- S. Lloyd (1996), “Universal Quantum Simulators”:

  “Yes”

- How can we make quantum computing work for the problems we want to solve?

Complex vector space

- Is a non-empty set of V whose elements are called *vectors*, denoted $|v\rangle$, with three operations:
  - addition $+$: $V \times V \to V$
  - negation $-$: $V \to V$
  - scalar multiplication $\cdot$: $C \times V \to V$

  and a distinguished element *zero vector* $0 \in V$
- These operations and zero satisfy the following properties:
  - commutativity and associativity of the addition
  - zero is an additive identity and every vector has an inverse
  - scalar multiplication has 1 and respects complex multiplication
  - scalar multiplication distributes over addition and complex addition

Complex vector space

- We are mainly interested in complex vector space C^n: n-tuples of complex numbers
- In the simplest case, C^2:

$$|z\rangle = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

- A *spanning set* is a set of vectors $|v_1\rangle, \ldots, |v_n\rangle$ such that for any vector $|v\rangle$: $|v\rangle = \sum_i a_i |v_i\rangle$
- For C^2 we can choose:

$$|v_1\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |v_2\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Complex vector space

- A set of non-zero $|v_1\rangle, \ldots, |v_n\rangle$ is *linearly dependent* if $a_1, \ldots, a_n$ exist that $a_1 |v_1\rangle + \ldots + a_n |v_n\rangle = 0$ with $a_i \neq 0$ for at least one value of i
- A set of linearly independent vectors that span vector space V is a *basis* in V
- The number of elements in the basis is the *dimension* of V
- In quantum computing we work with *finite-dimensional* vector spaces

Complex vector space

- A *linear operator* between vector spaces V and W is any function $A : V \to W$ which is linear in its inputs:

$$A\left(\sum_i a_i |v_i\rangle\right) = \sum_i a_i A(|v_i\rangle)$$

- Normally use simpler notation: $A(|v\rangle) \equiv A|v\rangle$

- Let $|v_1\rangle, \ldots, |v_m\rangle$ be the basis for V and $|w_1\rangle, \ldots, |w_n\rangle$ for W, then for each j in the range $1,\ldots, m$ there exist complex numbers $A_{1j}, \ldots, A_{nj}$ such that

$$A|v_j\rangle = \sum_i A_{ij}|w_i\rangle$$

- A_{ij} is a *matrix representation* of the operator A

Inner product

- An *inner product* on a complex vector space V is a function

$$(\ldots, \ldots) : V \times V \to C$$

that satisfies the following

  - non-degenerate: $(|v\rangle, |v\rangle) \geq 0$ and $(|v\rangle, |v\rangle) = 0$ if and only if $|v\rangle = 0$
  - respects addition and scalar multiplication
  - skew-symmetric: $(|v\rangle, |w\rangle) = (|w\rangle, |v\rangle)^*$

- V with inner product is called *inner product space (Hilbert space)*
- The *dual vector* $\langle v|$ for $|v\rangle$ is a linear operator from the inner product space V to the complex numbers C such that

$$\langle v|(|w\rangle) \equiv \langle v|w\rangle \equiv (|v\rangle, |w\rangle)$$

- Dirac notation: $|v\rangle$ — *ket-vector* (column), $\langle v|$ — *bra-vector* (row)

Orthonormal bases

- Vectors $|v\rangle$ and $|w\rangle$ are *orthogonal* if $\langle v|w\rangle = 0$
- Norm of vector $|v\rangle$: $||\ |v\rangle\ || \equiv \sqrt{\langle v|v\rangle}$
- A set of vectors $|i\rangle$ is *orthonormal* if each vector is a unit vector, $||\ |i\rangle\ || = 1$ and distinct vectors in the set are orthogonal, $\langle i|j\rangle = \delta_{ij}$

- Most of the time we use matrix representations of linear operators with respect to orthonormal bases
- The inner product on a Hilbert space with an orthonormal basis:

$$\langle v|w\rangle = \left(\sum_i v_i|i\rangle, \sum_j w_j|j\rangle\right) = \sum_{ij} v_i^* w_j \langle i|j\rangle = \sum_i v_i^* w_i$$

Outer product representation

- Let $|v\rangle \in V$ and $|w\rangle \in W$. Define *outer product* as a linear operator

  $$|w\rangle\langle v| : V \to W$$

  such that $(|w\rangle\langle v|)|v'\rangle \equiv |w\rangle\langle v|v'\rangle = \langle v|v'\rangle |w\rangle$

- Completeness relation: for any orthonormal basis $|i\rangle$ in V

  $$\sum_i |i\rangle\langle i| = I$$

Quantum computing: data representation

- Classical computing: bits

$$b \in \{0,1\}$$

- Quantum computing: qubits

$$|q\rangle \in C^2$$

- Computational basis:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

- An arbitrary state is a *superposition*

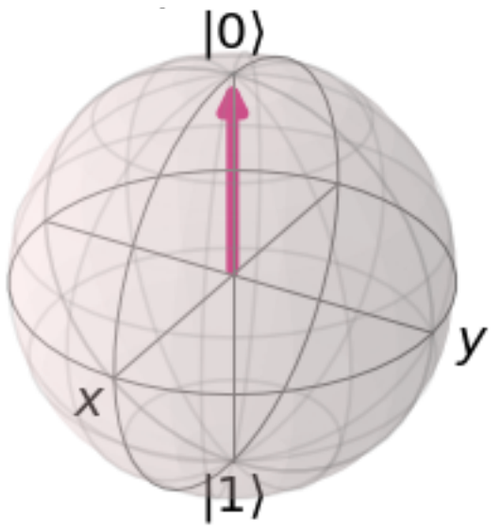
$$|q\rangle = \alpha|0\rangle + \beta|1\rangle \qquad |\alpha|^2 + |\beta|^2 = 1$$

or

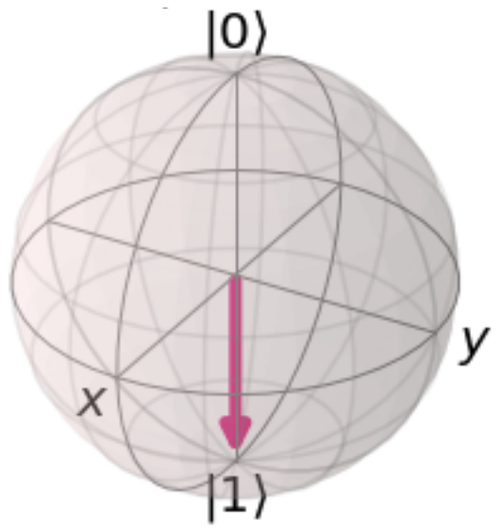
$$|q\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle\right)$$

Bloch sphere representation

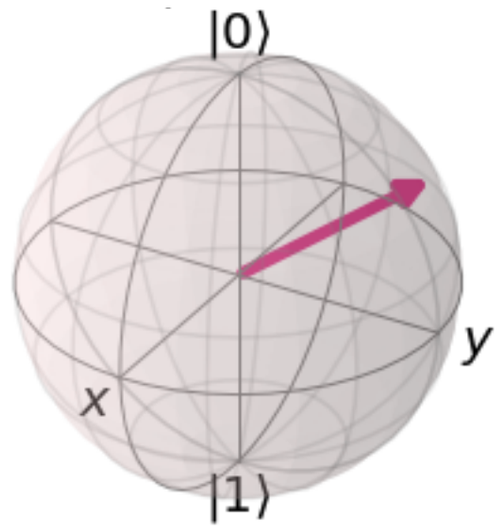
- Drop the global phase γ — it cancels in the observables
- The angles cover the sphere: $\theta \in [0,\pi]$, $\phi \in [0,2\pi)$, so we can treat them as spherical coordinates



$$|q\rangle = |0\rangle$$



$$|q\rangle = |1\rangle$$



$$|q\rangle = \alpha|0\rangle + \beta|1\rangle$$