Simulating quantum circuits with spin 1/2

G.C Liu

August 22, 2021

"Nature is quantum, god damn it! So if we want to simulate it, we need a quantum computer."

— Richard Feynmann

Contents

§1 How to stimulate quantum computer

First I will introduce the five necessary conditions for constructing the quantum computer[1] which is also known as DiVincenzo's criteria:

- A scalable physical system with well characterized qubits
- The ability to initialize the state of the qubits to a simple fiducial state, such as $|000\cdots\rangle$
- Long relevant decoherence times
- A "universal" set of quantum gates
- A qubit-specific measurement capability

But first I will construct ideal quantum circuit so we don't need to consider the the first three condition.I will discuss the physics implement in the below section.So what need is to find the universal set of quantum gates and the capability of measurement.

Like the classical computer Xor gate can construct any gate you want so what we need is to find the universal gate sets and find a method to construct any unitary n qubit gate with the universal gate.

§1.1 Single qubit operations

A single qubit is a vector

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

where α and β is complex number so that is easy to construct because we have Z-Y-Z decomposition

Theorem 1.1

Suppose U is a unitary operation on a single qubit.Then there must exist real numbers $\alpha, \beta, \gamma, \delta$

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta) \tag{2}$$

and there exist the lemma

Lemma 1.2

Suppose U is a unitary operation on a single qubit.Then there must exist unitary operators A,B,C on a single qubit such that $ABC = I$ and $U = e^{i\alpha} AXBXC$

set

$$A \equiv R_z(\beta)R_y(\gamma/2), B \equiv R_y(-\gamma/2)R_z(-(\delta+\beta)/2), C \equiv R_z((\delta-\beta)/2) \tag{3}$$

Note that

$$ABC = R_z(\beta)R_y(\gamma/2)R_y(-\gamma/2)R_z(-(\delta+\beta)/2)R_z((\delta-\beta)/2)) = I \tag{4}$$

Since $X^2 = I$

$$XBX = XR_y(-\gamma/2)XXR_z(-(\delta+\beta)/2)X = R_y(\gamma/2)R_z(\delta+\beta)/2) \tag{5}$$

Thus

$$AXBXC = R_z(\beta)R_y(\gamma/2)R_y(\gamma/2)R_z(\delta+\beta)/2)R_z((\delta-\beta)/2) = R_z(\beta)R_y(\gamma)R_z(\delta) \tag{6}$$

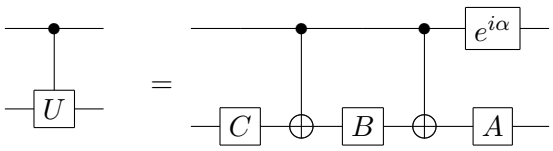
and use the theorem 1.1 we can construct any single-qubit gate
so we can use the lemma 1.2 to construct any control-U gate

§1.2 Two qubit operations

The most common two qubit gate is CNOT gate,and we can use the CNOT gate and some single qubit gate to construct any two qubit operation.

$$U = e^{i\alpha}AXBXC \tag{7}$$

so we can construct any control - U gate as follow:



§1.3 Universal Gate Set

A common universal gate set is the Clifford + T gate set, which is composed of the CNOT, H, S and T gates. The Clifford set alone is not universal and can be efficiently simulated classically by the Gottesman-Knill theorem. -Wiki

If we find the universal gate sets and find the method how to construct it thats what classical computer do!

Theorem 1.3

Gottesman-Knill theorem[2]

A quantum circuit using only the following elements can be simulated efficiently on a classical computer:

1. Preparation of qubits in computational basis states
2. Clifford gates (Hadamard gates, controlled NOT gates, phase gate S)
3. Measurements in the computational basis

so we find a universal gates to construct next what we need is to construct the four gates

§1.3.1 Single Gate

The main idea to construct the four gate is to use the "tensor product":

Also single-qubit gate can easily construct by matrix form

But if you have more than one qubit you need to have the fundamental mathematics knowledge below:

The state of n qubit is the tensor product of n qubit:

like:

$$\mathbf{A} \otimes \mathbf{B} \tag{8}$$

and tensor product have the following properties:

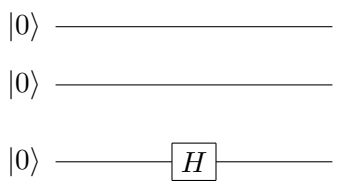
$$\begin{aligned} &(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD} \\ &a\mathbf{A} \otimes b\mathbf{B} = ab(\mathbf{A} \otimes \mathbf{B}) \\ &(\mathbf{A} + \mathbf{B}) \otimes (\mathbf{C} + \mathbf{D}) = \mathbf{A} \otimes \mathbf{C} + \mathbf{A} \otimes \mathbf{D} + \mathbf{B} \otimes \mathbf{C} + \mathbf{B} \otimes \mathbf{D} \end{aligned} \tag{9}$$

for example if we want to initiate qubit $|000\rangle$

we can rewrite the $|000\rangle$ like:

$$|000\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle \tag{10}$$

so the gate can also combine with tensor product for example:



If you want to represent the circuit you can use:

$$I \otimes I \otimes H \tag{11}$$

Because after the H operation the state:

$$|\psi\rangle = I|0\rangle \otimes I|0\rangle \otimes H|0\rangle \tag{12}$$

so use the property$((\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD})$you can get the result:

$$|\psi\rangle = (I \otimes I \otimes H)(|0\rangle \otimes |0\rangle \otimes |0\rangle) \tag{13}$$

And so on you can easily express the n-qubit single qubit gate

The program we prepare is to input the location of the single-gate and the gate,output the matrix form of the gate.

What package we use above is Scipy.sparse and numpy which are the useful tools to deal with matrix

```
from scipy import sparse
import numpy as np
```

First we construct common single qubit gate

```
Id = np.identity(2)
j =1j
X = np.array([[0,1],[1,0]])
Y = np.array([[0,-j],[j,0]])
Z = np.array([[1,0],[0,-1]])
S = np.array([[1,0],[0,j]])
T = np.array([[1,0],[0,np.exp(j*(np.pi/4))]])
```

before we construct any gate we first have initial qubit:

INPUT:[0,1,0,$\cdots$]

OUTPUT: Transform INPUT to the Column Vector

```
def init_q(qubit):
    q = single_q(qubit[0])
    for i in range(1,len(qubit)):
        q = sparse.kron(q,single_q(qubit[i]))
        #tensor product
    q = q.toarray()
    return q.T
```

Example 1.4

INPUT:qubit = [0,0,0]

Process: init(qubit)

OUTPUT:array([[1, 0, 0, 0, 0, 0, 0, 0]]).T

Next we construct the Hadmard gate:

```
def Hadmard_Gate(hl):
#Location of Hadamard Gate:Input array
    h = (1/2**0.5)*np.array([[1,1],[1,-1]])
    H = h*hl[0]+Id*(not hl[0])
    for i in range(1,len(hl)):
        H = sparse.kron(H,h*hl[i]+Id*(not hl[i]))
    return H.toarray()
```

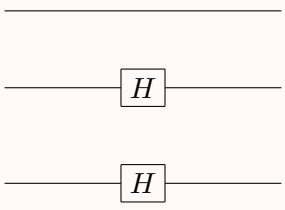
Single-Qubit-Gate:

```
def Single_Gate(gl,Gate):#Location of Single_Gate:Input array
    gate = Gate*gl[0]+Id*(not gl[0])
    for i in range(1,len(gl)):
        gate = sparse.kron(gate,Gate*gl[i]+Id*(not gl[i]))
    return gate.toarray()
```

Example 1.5

INPUT : Hadmard-Gate([0,1,1])/Single-Gate([0,1,1],h)

OUTPUT:



§1.3.2 CNOT

The most important gates is CNOT gates:

CNOT gate is a little complicated,But we have a better algorithmic way to think about quantum control gates is by using tensor products. Suppose we have a two qubit system.

The CNOT gate can be represented as:

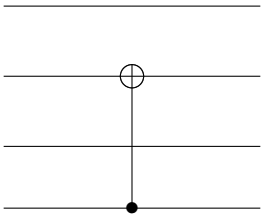
$$CNOT = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X \tag{14}$$

Because the first qubit is $|0\rangle$ the CNOT operator just like a Identity operator and if the first qubit is $|1\rangle$ the CNOT operator is the X operator on the second qubit

Based on this, we can construct a CNOT gate of n qubit:

For example:

Ff you want to the control bit is [0,0,0,1] and the target bit is [0,1,0,0]:like the circuit above:



The tensor product of the CNOT gate is

$$I \otimes I \otimes I \otimes |0\rangle\langle 0| + I \otimes X \otimes I \otimes |1\rangle\langle 1| \tag{15}$$

So based on this we can construct CNOT gate

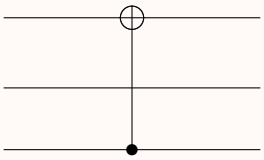
Code implement:

```
def CNOT_Gate(cl,tl):
#cl:Location of Control qubit Gate:Input array
#tl:Location of Target qubit Gate:Input array
    CNOT1 = P0*cl[0]+Id*(not cl[0])
    for i in range(1,len(cl)):
        CNOT1 = sparse.kron(CNOT1,P0*cl[i]+Id*(not cl[i]))
    CNOT2 = P1*cl[0]+Id*(not cl[0])+ X*(tl[0])
    for i in range(1,len(cl)):
        CNOT2 = sparse.kron(
        CNOT2,P1*cl[i]+
        Id*((not cl[i])and(not tl[i]))+ X*(tl[i]))
    return CNOT1+CNOT2
```

Example 1.6

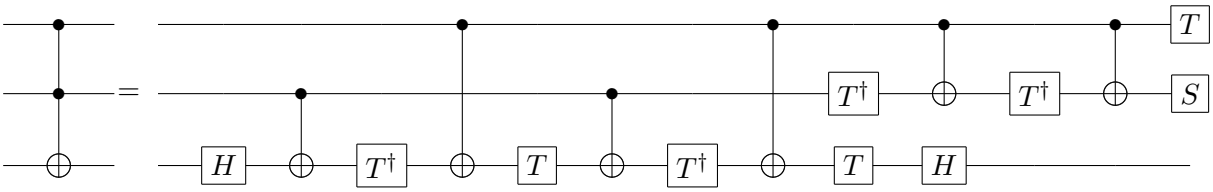
INPUT:CNOT-Gate([0,0,1],[1,0,0])

OUTPUT:



And use the gate above we can verify something

for example:



Try it by yourself

We compare the right part and left part

L = Toffoli-Gate

R =

Hadmard-Gate([0,0,1]),CNOT-gate([0,1,0],[0,0,1],qubit),

Single-gate([0,0,1],$T^\dagger$),CNOT-gate([1,0,0],[0,0,1],qubit),

Single-gate([0,0,1],T),CNOT-gate([0,1,0],[0,0,1],qubit), Single-gate([0,0,1],$T^\dagger$),

CNOT-gate([1,0,0],[0,0,1],qubit),Single-gate([0,1,0],$T^\dagger$),

Single-gate([0,0,1],T),CNOT-gate([1,0,0],[0,1,0],qubit),

Hadmard-Gate([0,0,1]),Single-gate([0,1,0],$T^\dagger$),CNOT-gate([1,0,0],[0,1,0],qubit),

Single-gate([1,0,0],T),Single-gate([0,1,0],S)

Code implement:

```
def ccnotgate(qubit):
    q = init_q(qubit)
    q = Hadmard_Gate([0,0,1])@q
    q = CNOT_Gate([0,1,0],[0,0,1])@q
    q = Single_Gate([0,0,1],Td)@q
    q = CNOT_Gate([1,0,0],[0,0,1])@q
    q = Single_Gate([0,0,1],T)@q
    q = CNOT_Gate([0,1,0],[0,0,1])@q
    q = Single_Gate([0,0,1],Td)@q
```

```
    q = CNOT_Gate([1,0,0],[0,0,1])@q
    q = Single_Gate([0,1,0],Td)@q
    q = Single_Gate([0,0,1],T)@q
    q = CNOT_Gate([1,0,0],[0,1,0])@q
    q = Hadmard_Gate([0,0,1])@q
    q = Single_Gate([0,1,0],Td)@q
    q = CNOT_Gate([1,0,0],[0,1,0])@q
    q = Single_Gate([1,0,0],T)@q
    q = Single_Gate([0,1,0],S)@q
    return q
```

Example 1.7

INPUT:[1,1,0]

Expected output:[1,1,1]

PROCESS(ccnotgate([1,1,0]))

Actual output:[1,1,1]

§1.4 Measure

For a single qubit:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{16}$$

so the probability of $|0\rangle$ is $|\alpha|^2$ and the probability of $|1\rangle$ is $|\beta|^2$

and if we have two qubit:

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \tag{17}$$

so after the simplification we get:

$$|\psi\rangle = \alpha_1\alpha_2\,|00\rangle + \alpha_1\beta_2\,|01\rangle + \beta_1\alpha_2\,|10\rangle + \beta_1\beta_2\,|11\rangle \tag{18}$$

So the probability of $|00\rangle$ is $|\alpha_1\alpha_2|^2$ and so on.By the way $|\alpha_1|^2 + |\beta_1|^2 = 1, |\alpha_2|^2 + |\beta_2|^2 = 1$so we can infer the probability of single qubit such that:

$$|\alpha_1|^2 = |\alpha_1|^2(|\alpha_2|^2 + |\beta_2|^2) = |\alpha_1\alpha_2|^2 + |\alpha_1\beta_2|^2 \tag{19}$$

So on and so forth we can get the probability of qubit

Code implement:

```
def measure(ml,tf,q):
#the Location of Measure:ml(input [0,0,1])
#tf:0 or 1 the value you want to measure
```

```
#q:final state of psi
    ls = []
    n = len(ml)
    qs = []
    m = []
    dic = {}
    result = []
    for i in range(2**len(ml)):
        ls.append(tobinary(i,len(ml)))
    for j in range(2**len(ml)):
        qs.append(single_q(ls[j][0]))
    for j in range(2**len(ml)):
        for i in range(1,n):
            qs[j] = np.kron(qs[j],single_q(ls[j][i]))
    for k in range(len(qs)):
        dic[f'{tobinary(k,n)}'] = np.dot(qs[k],q)[0]
        m.append(np.dot(qs[k],q)[0])
        result.append(tobinary(k,n))
    for i in range(len(ml)):
        if ml[i]!=0:
            ord = i
    prob = 0
    for i in result:
        if i[ord] == tf:
            prob += dic[f"{i}"]**2
    return prob
```

That's also the implement of the final s DiVincenzo' s criterion:A qubit-specific measurement capability.Actually in the real quantum physics system measure will realized in other way.The concrete method to measure will discuss in the section 3.

The measure there is not the practical measurement but it is the measure that you can get the probability for every qubit you want.That means if you apply the measure in the circuit,it will output the accurate probability of the qubit not the approximation.But the "real" measure will just output the state of qubit "0 or 1"

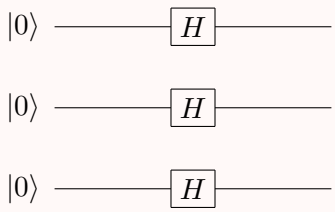
Also in the circuit we can use the above measure to construct a "real" measure:Because we have the probability of "0 or 1" and we just need a function:

INPUT:[0,1],[The probability of 0,The probaility of 1]:

OUTPUT:"0 or 1" and the frequency of occurrence of 0 appears by the probability of 0,1 is the same

Example 1.8

First you have the circuit below:

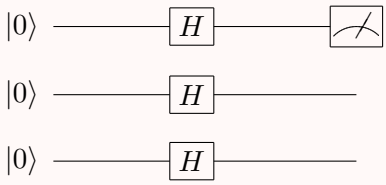


If you want to measure the first circuit and get the probability of $|0\rangle$,you can code like below:

STEP:

1. q = init-q([0,0,0])
2. q = Hadmard-Gate([1,1,1])@q
3. measure([1,0,0],0,q)

the circuit of above is:



OUTPUT:0.5

And the realmeasure can implement below:

Code Implement:

```
import random
def rand_pick(seq,prob):
    x = random.uniform(0,1)
    iprob = 0
    for item,item_prob in zip(seq,prob):
        iprob+=item_prob
        if x<iprob:
            break
    return item
def real_measure(ml,q):
    seq = [0,1]
    prob = [measure(ml,0,q),measure(ml,1,q)]
```

```
    return rand_pick(seq,prob)
```

Now the real measure function is almost the same as the actually measure

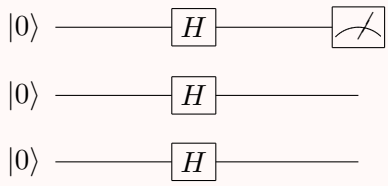
Example 1.9

We also use the example above to explain:

STEP:

1. q = init-q([0,0,0])
2. q = Hadmard-Gate([1,1,1])@q
3. realmeasure([1,0,0],q)

the circuit of above is:



OUTPUT:0:4975024975024975,1:0.5024975024975025(loop 1000)

Now we have constructed universal gates set and have the capability of measuring,The last problem we need to solve is to find a universal method to construct any n-qubit unitary transform.

§2 Implementation of some quantum algorithms

§2.1 Deutsch algorithm

Deutsch's algorithm[3] is a special case of the general Deutsch–Jozsa algorithm. We need to check the condition $f(0) = f(1)$.It is equivalent to check $f(0) \oplus f(1)$(where $\oplus$ is addition modulo 2,which can also seen as quantum Xor gate),if zero f is constant,otherwise f is not constant.

We begin with the two qubit state $|01\rangle$ and apply a Hadamard gate to each qubit so we get:

$$\frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \tag{20}$$

We are given a quantum implementation of a function f that transform $U|x\rangle|y\rangle \rightarrow |x\rangle|f(x) \oplus y\rangle$So apply the function to the state we have obtained

$$\begin{aligned}
&\frac{1}{2}(|0\rangle\left(|f(0)\oplus 0\rangle\right)-|f(0)\oplus 1\rangle)+|1\rangle\left(|f(1)\oplus 0\rangle\right)-|f(1)\oplus 1\rangle)) \\
&=\frac{1}{2}((-1)^{f(0)}|0\rangle\left(|0\rangle-|1\rangle\right)+(-1)^{f(1)}|1\rangle\left(|0\rangle-|1\rangle\right)) \\
&=(-1)^{f(0)}\frac{1}{2}(|0\rangle+(-1)^{f(0)\oplus f(1)})(|0\rangle-|1\rangle)
\end{aligned} \tag{21}$$

because:

$$|f(0)\oplus 0\rangle=\begin{cases}0 & f(0)=0\\ 1 & f(0)=1\end{cases} \tag{22}$$

so the function can be expressed as:

if $f(0)=0$

so $|0\rangle\left(|f(0)\oplus 0\rangle\right)-|f(0)\oplus 1\rangle)=|0\rangle\left(|0\rangle-|1\rangle\right)=(-1)^0|0\rangle\left(|0\rangle-|1\rangle\right)$

and so on we final get the expression:

$$(-1)^{f(0)}\frac{1}{2}(|0\rangle+(-1)^{f(0)\oplus f(1)}|1\rangle)(|0\rangle-|1\rangle) \tag{23}$$

We ignore the global phase,we have the state:

$$\frac{1}{\sqrt{2}}(|0\rangle+(-1)^{f(0)\oplus f(1)}|1\rangle)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle) \tag{24}$$

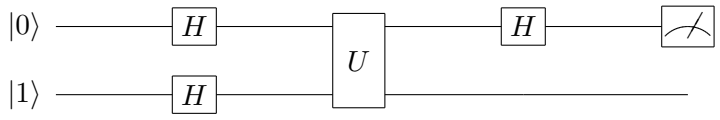
Apply the Hadamard gate to the first state:

we have:

$$\frac{1}{2}((1+(-1)^{f(0)\oplus f(1)})|0\rangle+(1-(-1)^{f(0)\oplus f(1)})|1\rangle) \tag{25}$$

So measure the state we can get the whether $f(0)\oplus f(1)=0or1$

The circuit below:



Code Implement:

```
q = init_q([0,1])
q = Hadmard_Gate([1,1])@q
q = Uf(q)
q = Hadmard_Gate([1,0])@q
real_measure([1,0],q)
```

§2.2 Quantum Fourier Transform

The quantum Fourier transform is the classical discrete Fourier transform applied to the vector of amplitudes of a quantum state, where we usually consider vectors of length $N = 2^n$

So from the classical fourier transform:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k} |k\rangle \tag{26}$$

So through the characteristics of Fourier transform, we can only use Hadamard gate and controlled phase gate R_k gate to construct.

$$R_k \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix} \tag{27}$$

For example we can construct 3 qubit quantum fourier transform below:

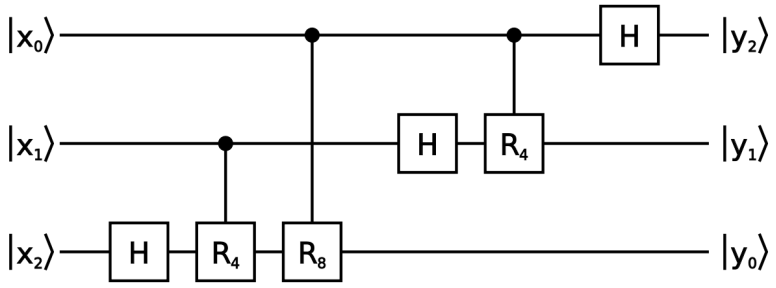


Figure 1: Three qubit fourier transform circuit

Fisrt we need to construct the CNOT-R4 and CNOT-R8:

So we must have found a efcient decomposition methods for controlled-Rn[4]

Now we have a universal method to implement Controlled$-Z^{\frac{1}{2^n}}$ gate we would need H, CNOT and $Z^{\frac{1}{2^{n+1}}}$(along with its adjoint version) gate.

We can use the same way of constructing the CNOT gate to construct any CU gate you want:

Code Implement:

```
def CR_Gate(cl,tl,Gate):
#cl:Location of Control qubit Gate:Input array
#tl:Location of Target qubit Gate:Input array
    CR1 = P0*cl[0]+Id*(not cl[0])
    for i in range(1,len(cl)):
        CR1 = np.kron(CR1,P0*cl[i]+Id*(not cl[i]))
    CR2 = P1*cl[0]+Id*(not cl[0])+ Gate*(tl[0])
```

```
    for i in range(1,len(cl)):
        CR2 =
        np.kron(CR2,P1*cl[i]
        +Id*((not cl[i])and(not tl[i]))
        +Gate*(tl[i]))
    return CR1+CR2
```

So we can use our gate to implement the three qubit fourier transform:

```
#Three Qubit Fourier Transform
def threequbitfft():
    fft = Hadmard_Gate([0,0,1])
    fft = np.dot(CR_Gate([0,1,0],[0,0,1],R(2)),fft)
    fft = np.dot(CR_Gate([0,1,0],[0,0,1],R(2)),fft)
    fft = np.dot(CR_Gate([1,0,0],[0,0,1],R(3)),fft)
    fft = np.dot(Hadmard_Gate([0,1,0],R(3)),fft)
    fft = np.dot(CR_Gate([1,0,0],[0,1,0],R(2)),fft)
    fft = np.dot(Hadmard_Gate([1,0,0]),fft)
    return fft
```

§3 Physics implement of quantum gate with spin 1/2

Any physically realizable quantum computer is a complicated many-body system that interacts with its environment.

In quantum statistical mechanics and quantum chemistry, it is well known that simulating an interacting quantum many-body system becomes exponentially more difficult as the size of the system grows.

§3.1 Spin1/2 Algebra

Measurements of a component of the spin (=internal angular momentum) of particles such as electrons,protons, and neutrons along any direction yield either $-\frac{\hbar}{2}$ or $\frac{\hbar}{2}$. The convention is to call the state of the spin that corresponds to the outcome $\frac{\hbar}{2}$ "spin up" ($|\uparrow\rangle$) and the other state "spin down" ($|\downarrow\rangle$).[5]

So in the context of quantum computation, it is convenient to define:

$$|0\rangle = |\uparrow\rangle, |1\rangle = |\downarrow\rangle \tag{28}$$

§3.2 State Representation

Just like the initial operation in the fisrt section:we need to express the state like $|[0,0,0,1]\rangle$ to $|0\rangle \cdots |[1,1,1,1]\rangle$ to $|15\rangle$

§3.3 Physics implement of quantum gate

Conceptually, the main differences between simulating quantum computers on a quantum-gate level (section 1) and physical models of quantum computers are that the dynamics of the latter are specified in terms of a time-dependent Hamiltonian (not in terms of unitary matrices) and that the physical quantum computer is a single physical system.

Candidate technologies for building quantum gates include ion traps, cavity QED, Josephson junctions, and nuclear magnetic resonance (NMR) technology

The biggest difference between different physics implementations is that different implementations have different Hamiltonian.

for example:

The simplest Ising model of a universal quauntum computer:

$$H_{Ising} = -\sum_{i,j=1}^{L} J_{i,j}^{z}(t) S_i^z S_j^z - \sum_{j=1}^{3} \sum_{\alpha=x,y,z} h_j^{\alpha}(t) S_j^{\alpha} \tag{29}$$

An approximate model for the linear arrays of quantum dots reads:

$$H(t) = -\sum_{j=1}^{L} E_j S_j^z S_{j+1}^z - \sum_{j=1}^{L} h_j^x(t) S_j^x + E_0 \sum_{j=1}^{L} P_j(t) S_j^z \tag{30}$$

where $E_j = E_0\,(E_j = 2E_0)$ when j is odd (even) and $h_j^x(t)$ and $P_j(t)$ are external control parameters [111]. Projection of the Josephson-junction model onto a subspace of two states per qubit yields.Its Hamiltonian is:

$$H(t) = -2E_I(t) \sum_{j=1}^{L} S_j^y S_{j+1}^y - E_J \sum_{j=1}^{L} S_j^x - \sum_{j=1}^{L} h_j^z(t) S_j^z \tag{31}$$

§3.4 NMR Quantum Computer

From the NMR we can learn how to control the outfield to control the Hamiltonian to operate any unitary transform

Summary to the NMR hamilton,the hamilton of n spin system H is :

$$H = H_{sys} + H_{con} + H_{env} \tag{32}$$

where H_{sys} is the hamilton of system and so on.

§3.4.1 The Hamilton of System

The Hamiltonian of the system includes the interaction between nuclear spin and static magnetic field, and the interaction between different spins.

Zeeman Interaction

A spin 1/2 particle and the magnetic field $B_0$with direction Z,its hamiltonian is also known as Zeeman Hamiltonian,

$$H_Z = -\hbar\gamma B_0 I_z = -\hbar w_0 I_z \tag{33}$$

where γ is the ratio of nucleus gyromagnetic ratio,w_0 is the larmor frequency,I_z is the Angular momentum operator of Z direction.And its connection with Pauli Matrix is :

$$I_z = \frac{1}{2}\sigma_z \tag{34}$$

(32) can also be considered as the single spin 1/2 particle interact with the static magnetic field B_0

This energy level split is also called Zeeman effect

The frequency between the two level $w_0 = \gamma B_0$ is usual hundreds MHZ.If the frequency of addtional electromagnetic wave is almost w_0.This nucleus will resonate

So for a nucleus with n spin 1/2 the all interaction is:

$$H_z = -\sum_{k=1}^{n} \hbar w_0^k I_z^k \tag{35}$$

Interaction between nucleus and nucleus

nucleus with n spin 1/2

the hamiltonian is:

$$H_{sys} = H_z + H_D + H_J \tag{36}$$

where H_D is the magnetic dipole and magnetic dipole interaction,H_J is the J coupling.

And the H_D is:

$$H_D = \hbar \sum_{i<j} \pi D_{ij}(2I_z^i I_z^j - I_x^i I_x^j - I_y^i I_y^j) \tag{37}$$

But in the isotropic liquid the H_D is zero.

$$H_J = \hbar \sum_{i<j} 2\pi J_{ij} I^i \cdot I^j \tag{38}$$

Since the Weak coupling approximation

Thus:

$$H_J = \hbar \sum_{i<j} 2\pi J_{ij} I_z^i I_z^j \tag{39}$$

Thus:

$$H_{sys} = H_z + H_J = -\sum_{k=1}^{n} \hbar w_0^k I_z^k + \hbar \sum_{i<j} 2\pi J_{ij} I_z^i I_z^j \tag{40}$$

§3.4.2 The Hamilton of Field

From the H_{sys} above.What we can implement is the single Z gate like the single qubit gate in section 1

And we can apply additional electromagnetic field with plane x-y and the frequency in the radio frequency range.which is also known as Radio Frequency.The additional electromagnetic field in the plane x-y rotation with the frequency w_{rf},with the initial angle ϕ,The amplitude is B_1

So the additional electromagnetic field is:

$$B_{rf}(t) = B_1(\hat{e}_z cos\theta_{rf} + \hat{e}_x sin\theta_{rf}) cos(w_{rf}t + \phi) \tag{41}$$

And $H = \mu B$

Thus:

$$H_{rf} = -\hbar\gamma B_{rf}[cos(w_{rf}t + \phi)I_x - sin(w_{rf}t + \phi)I_y] \tag{42}$$

so the n spin 1/2 the H_{con} is:

$$H_{con} = -\sum_{i}^{n} -\hbar\gamma_i B_{rf}[cos(w_{rf}t + \phi)I_x^i - sin(w_{rf}t + \phi)I_y^i] \tag{43}$$

§3.4.3 The Hamilton of Environment

When operate the single qubit gate the H_{env} can be ignored because of the relaxation time is much longer than operator time.

§3.5 Implement of Single Qubit Gate

For a single qubit:

$$\begin{aligned} H_{sys} &= -\hbar w_0^i I_z^i \\ H_{con} &= -\hbar\gamma_i B_1(cos(w_{rf}t + \phi)I_x^i - sin(w_{rf}t + \phi)I_y^i) \end{aligned} \tag{44}$$

Establishment of rotating coordinate system

Because the t is in the Hamilton so we introduce the rotation axis.The origin hamilton is replaced by the new hamilton H_e

$$H_e = UHU^{-1} - iU\frac{dU^{-1}}{dt} \tag{45}$$

where $U = exp(-iI_z w_{rf} t)$

So the H_{con} can be replaced by the time-dependent $B_1(cos\phi I_x + sin\phi I_y)$

When in the rotation axis:

$$H_e = -\hbar(w_0 - w_{rf})I_z - \hbar\gamma B_1(cos\phi I_x - sin\phi I_y) \tag{46}$$

when $w_0 = w_{rf}$ and $\phi = \pi$,The hamilton is:

$$H_e = \hbar\gamma B_1 I_x \tag{47}$$

so with the evolution over time we finally get:

$$U(\tau) = exp(-i\hbar\gamma B_1 I_x \tau) \tag{48}$$

So now we implement any rotation with axis x,as the same way we can also implement the any rotation with axis y.

And in section we have:

$$R_z(\theta) = R_x(\frac{\pi}{2})R_y(\theta)R_x(-\frac{\pi}{2}) \tag{49}$$

and Theorem 1.1 so we can construct any single qubit gate in the this Hamilton.

References

[1] D. P. DiVincenzo, "The physical implementation of quantum computation," *Fortschritte der Physik: Progress of Physics*, vol. 48, no. 9-11, pp. 771–783, 2000.

[2] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, no. 5, p. 052328, 2004.

[3] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[4] T. Kim and B.-S. Choi, "Efficient decomposition methods for controlled-r n using a single ancillary qubit," *Scientific reports*, vol. 8, no. 1, pp. 1–7, 2018.

[5] H. De Raedt and K. Michielsen, "Computational methods for simulating quantum computers," *arXiv preprint quant-ph/0406210*, 2004.