Each Quantum computer has a physical implemented set of gates which is not necessarily the same set provided by its SDK

for example, IBM has three physical gates

$$\{U_1(\lambda), R_x(\pi/2), CNOT\}$$

however SDK offers

$$\{I, X, Y, Z, H, S, S^+, T, T^+, U_1(\lambda), U_2(\lambda,\phi), U_3(\lambda,\phi,\theta), CNOT\}$$

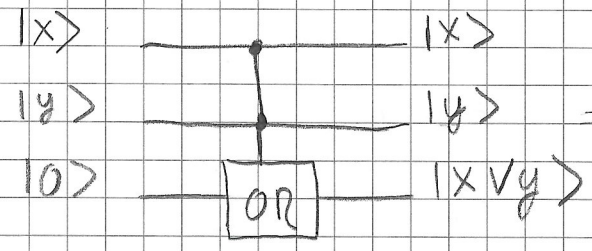
We will also need several arithmetic and logic quantum gates for general computation

OR Gate

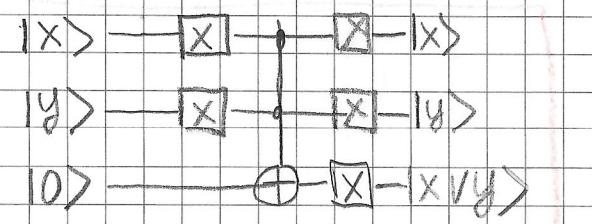
The OR gate is easily implemented using only X and CCX (Toffoli) gates

$$x \vee y = \neg(\neg x \wedge \neg y)$$

(Boole algebra binding, NOT, OR, AND)



=



OR Gate

GÜLPAŞ

The Quantum Fourier Transform

For simplification let $e(t) = e^{2\pi i t}$

Let $a \in Z_{2^n}$ the additive group of integers modulo 2^n. Let $a_n a_{n-1} \dots a_2 a_1$ be the binary representation of a where $a = a_n 2^{n-1} + a_{n-1} 2^{n-2} + \dots + a_2 2^1 + a_1 2^0$

Then $|a\rangle = |a_n\rangle \otimes |a_{n-1}\rangle \otimes \dots \otimes |a_2\rangle \otimes |a_1\rangle$

The quantum Fourier transform (QFT) of $|a\rangle$ is the mapping

$$|a\rangle \xrightarrow{F_{2^n}} \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e(ak/2^n)\,|k\rangle$$

Set

$$|\phi_k(a)\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e(a/2^k)\,|1\rangle\right)$$

Then

$$\sum_{k=0}^{2^n-1} e(ak/2^n)\,|k\rangle = |\phi_n(a)\rangle \otimes \dots \otimes |\phi_2(a)\rangle \otimes |\phi_1(a)\rangle$$

Consider "conditional phase gate" i.e. a conditional rotation about the z-axis

$$R_k = \quad = \quad \doteq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e(\frac{1}{2^k}) \end{bmatrix}$$

Let's decompose a into its representation as a binary number

$$a = \sum_{i=0}^{n-1} 2^i a_i$$

(a_i — binary digits of a)

Notice multiplication of this by 2^{k-1} gives

$$e\left(\frac{2^{k-1}a}{2^k}\right) = e^{\pi i (a_0)} = (-1)^{a_0}$$

Thus we can write the most significant qubit is

$$|0\rangle + (-1)^{a_0}|1\rangle$$

which is a Hadamard gate applied to least significant digit of the input

The next significant digit is

$$|0\rangle + e\left(\frac{2^{k-2}}{2^k}a\right)|1\rangle$$

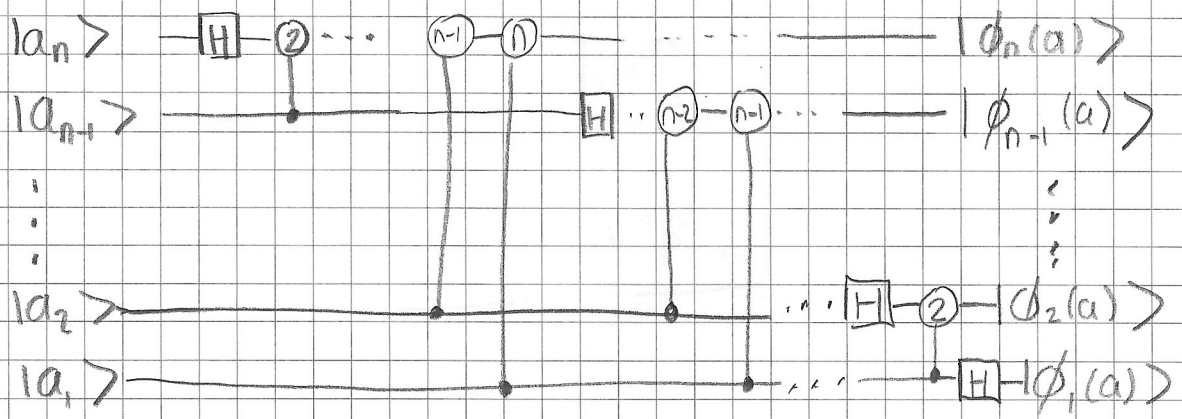
$$|0\rangle + e\left(\frac{2^{k-2}}{2^k}a_0\right) H|a_1\rangle$$

conditional phase gate to lower bit

In general qubit n-j is

$$|0\rangle + \prod_{i<j-1} e\left(\frac{2^{k-j+i}}{2^k}a_i\right)|a_{j-1}\rangle$$

which is a Hadamard gate followed by a sequence of conditional rotations



$$|a_n\rangle \longrightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e(0.a_n)|1\rangle\right) \quad \text{Hadamard}$$

$$\longrightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e(0.a_n a_{n-1})|1\rangle\right) \quad R_2 \text{ rot. cond. on } a_{n-1}$$

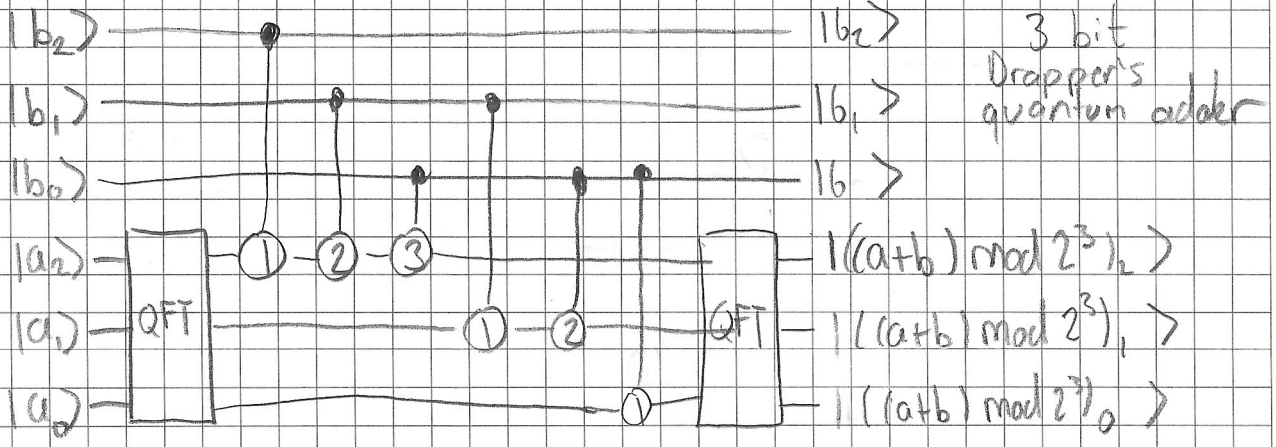
$$\vdots$$

$$\longrightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e(0.a_n a_{n-1}\cdots a_1)|1\rangle\right) \quad R_n \text{ rot. cond. on } a_1$$

$$= |\phi_n(a)\rangle$$

Draper's Quantum adder

Implementation of quantum adder is very similar to the QFT.

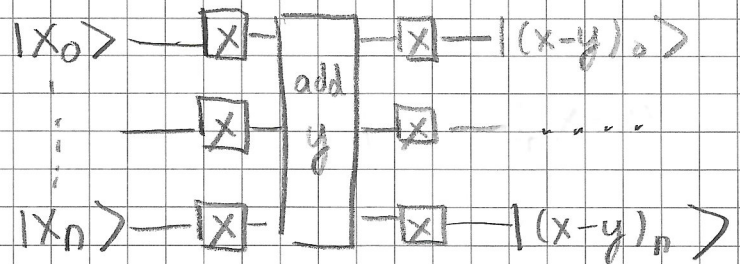


3 bit Draper's quantum adder

SUB gate

Implementation of a subtractor is trivial once an adder is available

$$a-b = (a' + b)'$$ ← bitwise complement

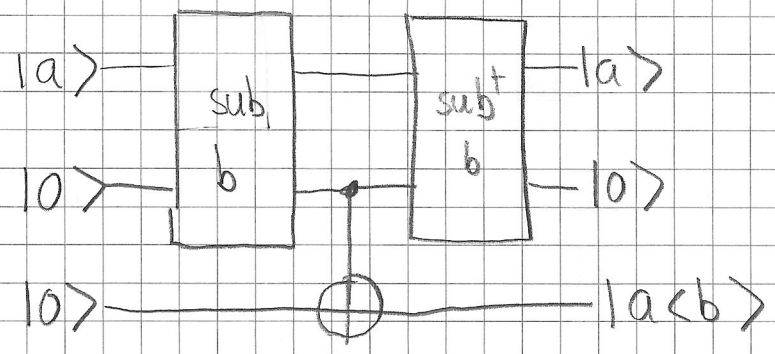


Here the y value encoding is intentionally ommitted. Either the y value is encoded in the adder in a quantum register or directly implemented in the circuit

Note that y value is not negated

CMP gate

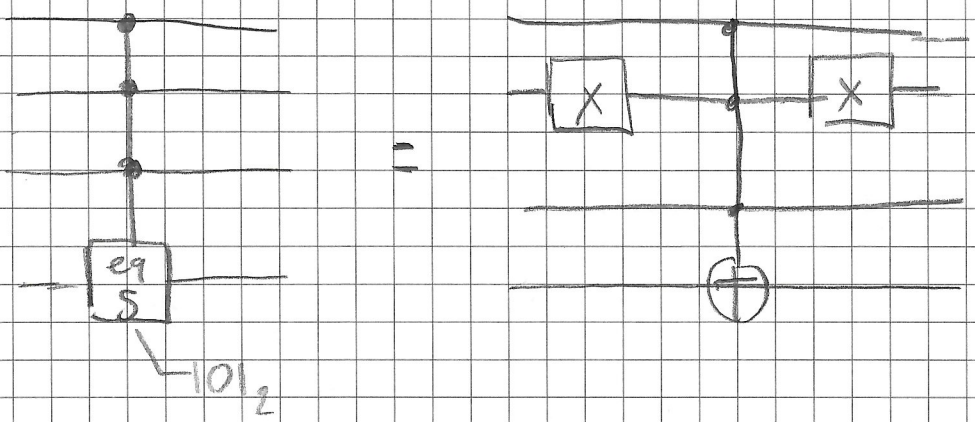
A strategy to construct a comparison gate is to compute the high bit of expression $a-b$, if this high bit is in $|1\rangle$, then $a<b$



The EQ gate

EQ gate tests equality between an integer stored in a quantum register and a generation time constant integer

The implementation requires a multi controlled Toffoli gate and a few X gates. These gates allow us to be flexible with generation time constant integer since Toffoli gate will set when all of its controls are 1



Partial Derivative Equation Example: 1D wave equation

Problem Considered

In this chapter we will consider a simplified version of the wave equation, and design a quantum circuit to solve it

Consider a 1-Dimensional line $[0,1]$ where the propagation speed is constant ($c=1$)

$$\frac{\partial^2}{\partial t^2}\phi(x,t) = \frac{\partial^2}{\partial x^2}\phi(x,t)$$

Moreover, we only consider the Dirichlet boundary conditions

$$\frac{\partial}{\partial x}\phi(0,t) = \frac{\partial}{\partial x}\phi(1,t) = 0$$

No assumptions are made on initial speed $\phi(x,0)$ and initial velocity $\frac{\partial \phi}{\partial t}(x,0)$

Hamiltonian simulation is the problem of constructing a quantum circuit that will evolve a quantum state according to a Hamiltonian matrix, following the Schrödinger equation

In other words, Hamiltonian simulation algorithms generate a quantum circuit performing the unitary transformation U such that

$$\|U - e^{-iHt}\| < \epsilon$$

H being the given Hamiltonian matrix, t a time evolution, and ϵ is the precision

One of the strategies to approach this problem is the product-formula approach. This strategy has three steps : decompose, simulate, recompose

s-sparse matrix : A s-sparse matrix with $s \in \mathbb{N}^*$ is a matrix that has at most s non-zero entries per row and per column

sparse matrix : A sparse matrix is a s-sparse matrix with $s \in O(\log(N))$, N being the size of the matrix

Product-Formula algorithm first decomposes s-sparse H as a sum of Hermitean matrices H_j that should be easy to simulate

$$H = \sum_{j=0}^{m-1} H_j$$

The second step is about implementing e^{-iH_jt} and the last step is to bring these together to approximate e^{-iHt}

Definition Easy to simulate matrix:

A Hermitean matrix H is easy to simulate if an algorithm takes as an input time t and matrix H and outputs a quantum circuit $C(H)_t$ such that

1.) The quantum circuit $(C(H)_t$ implements exactly the unitary transformation

$$\| e^{-iHt} - C(H)_t \| = 0$$

2.) The algorithm only needs $O(1)$ calls to the oracle of H and $O(\log N)$ additional gates, N being the dimension of H

The first and easiest matrices that fulfill the easy to simulate matrix requirements are the multiples of the Identity matrix

αI (easy to simulate)

A larger class of matrices that are easy to simulate are the 1-sparse, integer weighted, Hermitean matrices

Lie - Trotter - Suzuki product formula

$$\exp\left(\lambda \sum_{j=0}^{m} a_j H_j\right)$$

$$\approx S_2(\lambda) = \prod_{j=0}^{m-1} e^{a_j H_j \lambda/2} \prod_{j=m-1}^{0} e^{a_j H_j \lambda/2}$$

this can be generalised to higher orders

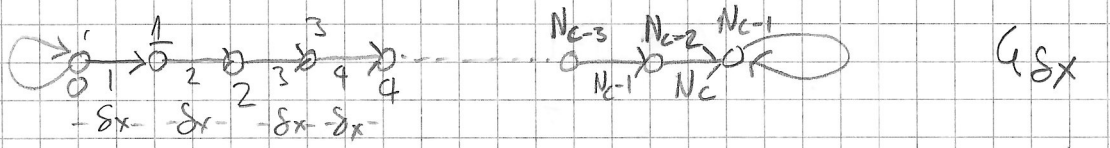
$$S_{2k}(\lambda) = \left[S_{2k-2}(p_k\lambda)\right]^2 \times S_{2k-1}((1-4p_k)\lambda)\left[S_{2k-2}(p_k\lambda)\right]^2$$

where $p_k = (4 - 4^{1/(2k-1)})^{-1}$ for $k \geq 1$

Then

$$e^{\lambda H} = \left[S_{2k}\left(\frac{\lambda}{n}\right)\right]^n + O\left(\frac{|\lambda|^{2k+1}}{n^{2k}}\right)$$

Graph representation of the problem



$G_{\delta x}$

In order to devise the Hamiltonian matrix to be simulated, we need to discretize the problem with respect to space first.

Such a discretization can be seen on graph $G_{\delta x}$.

The graph Laplacian

$$L(G_{\delta x})_{ij} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } (i \neq j) \wedge (v_i \text{ adjacent to } v_j) \\ 0 & \text{otherwise} \end{cases}$$

can be used to approximate the differential operator $\frac{\partial^2}{\partial x^2}$. By using discretization

$$\frac{\partial^2 \phi}{\partial x^2}(i\delta x, t) \approx \frac{\phi_{i-1,t} - 2\phi_{it} + \phi_{i+1,t}}{\delta x^2}$$

this gives the linearized matrix

$$A = -\frac{1}{\delta x^2} L(G_{\delta x})$$

$$[A\phi]_i = \frac{\phi((i-1)\delta x, t) - 2\phi(i\delta x, t) + \phi((i+1)\delta x, t)}{\delta x^2}$$

plugging this back to wave equation

$$\frac{\partial^2 \phi}{\partial t^2} = -\frac{1}{\delta x^2} L(G_{\delta x})\phi$$

Simulating

$$H = \begin{pmatrix} 0 & B \\ B^\dagger & 0 \end{pmatrix} \text{ with } BB^\dagger = L$$

constructs a quantum circuit that will evolve a part of the quantum state it is applied on according to the discretized wave equation

Such a matrix can be found using the following algorithm

$$B_{ij} = \begin{cases} 1 & \text{if edge } j \text{ is a self loop of vertex } i \\ 1 & \text{if edge } j \text{ has vertex } i \text{ as source} \\ -1 & \text{if edge } j \text{ has vertex } i \text{ as sink} \\ 0 & \text{otherwise} \end{cases}$$

$$B_d = \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{pmatrix}$$

$$B_d B_d^{\dagger} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & 2 & -1 \\ \vdots & & & -1 & 2 \\ 0 & & & & \end{pmatrix}$$ (gives the discretization)

Discretized Hamiltonian $\tilde{H}_D$ is then

$$\tilde{H}_D = \frac{1}{\delta x} \begin{pmatrix} 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & 0 & -1 & 1 & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & \cdots & -1 & 1 \\ 1 & & 0 & 0 & \cdots & \cdots & 0 & \\ 1 & -1 & \ddots & 0 & \cdots & \cdots & 0 & \\ 0 & \ddots & \ddots & -1 & \ddots & \ddots & \vdots & \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & \end{pmatrix}$$

There are a lot of valid decompositions for this, but one must choose considering gate complexity

As explained before we want to split the Hamiltonian as a sum of 1-sparse matrices

B_d can be decomposed as two 1-sparse matrices

$$B_1 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & & 1 & \ddots & \vdots \\ \vdots & & & \ddots & 0 \\ 0 & \cdots & & 0 & 1 \end{pmatrix} \qquad B_{-1} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & & -1 & 0 \end{pmatrix}$$

Notice $B_d = B_1 + B_{-1}$ hence when

$$\tilde{H}_1 = \frac{1}{\delta x}\begin{pmatrix} 0 & B_1 \\ B_1^\dagger & 0 \end{pmatrix}, \quad \tilde{H}_{-1} = \frac{1}{\delta x}\begin{pmatrix} 0 & B_{-1} \\ B_{-1}^\dagger & 0 \end{pmatrix}$$

then

$$\tilde{H}_d = \tilde{H}_1 + \tilde{H}_{-1}$$

for convenience define

$$H_1 = \begin{pmatrix} 0 & B_1 \\ B_1^\dagger & 0 \end{pmatrix} \qquad H_{-1} = \begin{pmatrix} 0 & B_{-1} \\ B_{-1}^\dagger & 0 \end{pmatrix}$$

$$H_d = H_1 + H_{-1}$$

i.e. $\tilde{H}_1$, $\tilde{H}_{-1}$ and $\tilde{H}_d$ are rescaled matrices that contain only integers and H_1, H_{-1} and H_d are unscaled counterparts

Due to how we constructed them, simulating $\tilde{H}_d$ for a time t is equal to simulating H_d for a time $\frac{t}{\delta x}$. So we can use H_1, H_{-1} and H_d

The dimensions of H_d gives the lower bound of number of qubits needed for N_d discretization points. The non-empty upper left block of H_d is of dimension $(2N_d - 1) \times (2N_d - 1)$. Without the ancilla qubits, this will require $\log_2(2N_d - 1)$ qubits

Oracle construction

Oracles can be seen as the interface between a quantum procedure and classical data. It encodes classical data into quantum system as efficiently as possible

The interface: Bridges classical computation and quantum computation

In our case we will use an interface like this:

$$O|x_0\rangle|0\rangle|0\rangle|0\rangle = |x_0\rangle|m(x_0)\rangle|V(x_0)\rangle|S(x_0)\rangle$$

- $|x_0\rangle$: encodes a row index as a unsigned integer
- $|m(x_0)\rangle$: Returns the only non-zero element column index
- $|V(x_0)\rangle$: $= w(x)$ the absolute value of the weight
- $|S(x_0)\rangle$: $S(x) = \begin{cases} 0 & \text{if } w(x) \geq 0 \\ 1 & \text{else} \end{cases}$ (the sign bit)

The interface of the oracle O can be seperated into three oracles M, V, S

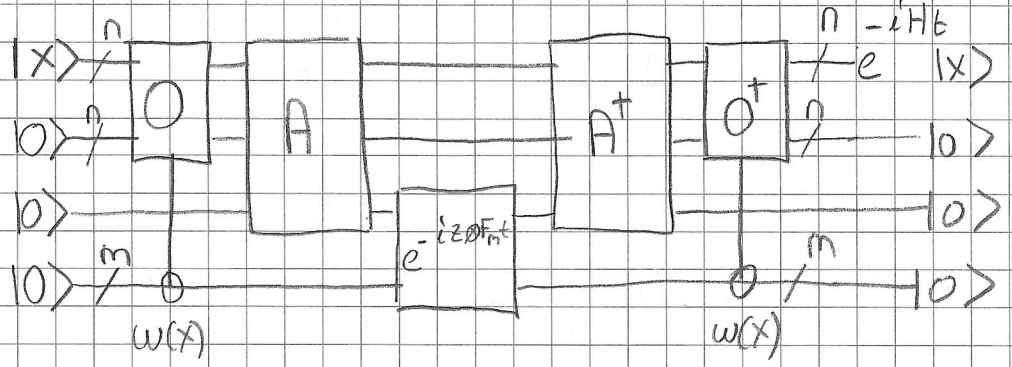
$$M|x_0\rangle|0\rangle = |x_0\rangle|m(x)\rangle$$
$$V|x_0\rangle|0\rangle = |x_0\rangle|V(x)\rangle$$
$$S|x_0\rangle|0\rangle = |x_0\rangle|S(x)\rangle$$

Let's bring all together

$$|\Psi(t)\rangle = e^{-iHt/\hbar}|\Psi(0)\rangle$$

$$\frac{\partial^2\phi}{\partial t^2} = -\frac{1}{\delta x^2}L(G_{\delta x})\phi = A\phi$$

m time steps, n discretization



In this circuit, when the weight w(x) (oracle V) goes zero O becomes the identity matrix

If there is no "memory" effect, then $m \to 1$

$$e^{-\mathbb{Z}\otimes F t} = \exp\left[-i\begin{pmatrix} F & 0 \\ 0 & -F \end{pmatrix} t\right]$$

$$= \begin{pmatrix} e^{-iFt} & 0 \\ 0 & e^{iFt} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-it} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix}$$

and if there is memory effect (i.e. you need more time points)

$$F_m = \begin{pmatrix} 0 & & & & & \\ & 1 & & & & \\ & & 2 & & & \\ & & & 3 & & \\ & & & & 4 & \\ & & & & & \ddots \\ & & & & & & 2^m-1 \end{pmatrix}$$

Notice that x^{th} row of the considered Hermitean matrix has no non-zero entries $\mathcal{O}$ is ignored, meaning when V is $\mathcal{O}$ M and S are ignored

This gives us freedom to "optimize" H_1 and H_{-1} and fill them with non-zero elements as desired, keeping the V oracle implements right transformation

$$H_1^{opt} = \begin{pmatrix} 0 & \cdots & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & & & \vdots & \vdots & & \ddots & \ddots & \vdots & & & & & \vdots \\ \vdots & & & \vdots & \vdots & & & \ddots & 0 & & & & & \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0 & 1 & & & & & \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 & & & & & \\ 1 & \ddots & & \vdots & \vdots & & & & \vdots & & & & & \\ 0 & \ddots & \ddots & \vdots & \vdots & & & & \vdots & & & & & \\ \vdots & & \ddots & 0 & \vdots & & & & \vdots & & & & & \\ 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 & & & & & \\ 0 & \cdots & \cdots & 0 & & & & & & & & & & \\ \vdots & & & & & & & & & & & & & \\ \vdots & & & & & & & & & & & & & \\ \vdots & & & & & & & & & & & & & \\ 0 & - & - & - & - & - & - & - & - & - & - & - & - & - \end{pmatrix}$$

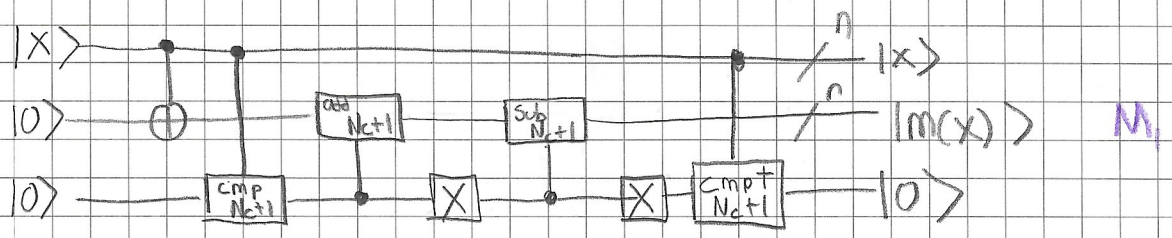
N_c

N_c+1

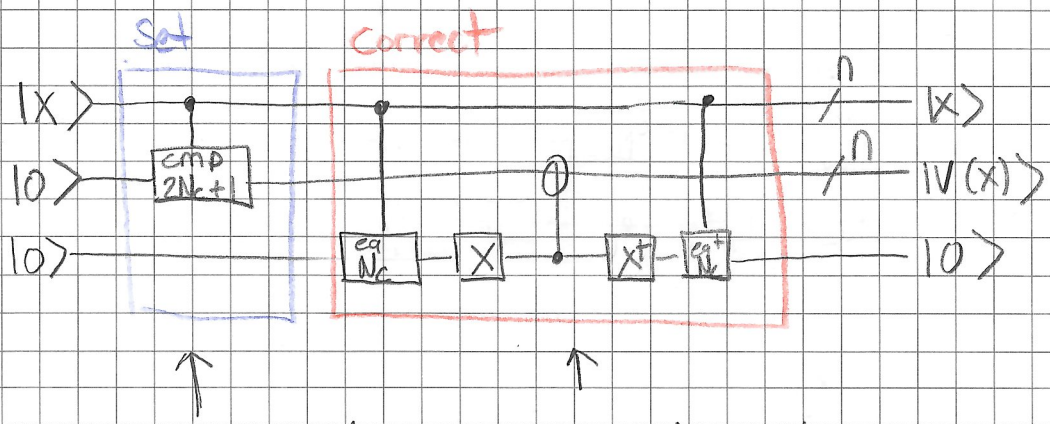
$2^q-(2N_c+1)$

using the definitions of oracles M V and S

$$M_1 |x\rangle|0\rangle \to \begin{cases} |x\rangle|x+(N_c+1)\rangle & \text{if } x<(N_c+1) \\ |x\rangle|x-(N_c+1)\rangle & \text{else} \end{cases}$$



$$V_1 |x\rangle|0\rangle \mapsto \begin{cases} |x\rangle|1\rangle & \text{if } (x<2N_c+1) \wedge (x \neq N_c) \\ |x\rangle|0\rangle \end{cases}$$



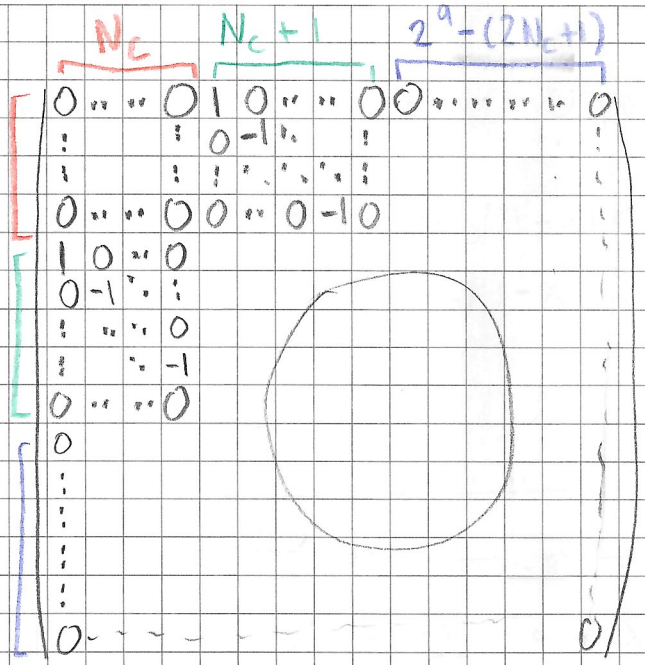
Sets weight to 1 for $x<2N_c+1$

Sets weight back to $|0\rangle$ when $|x\rangle = N_c$

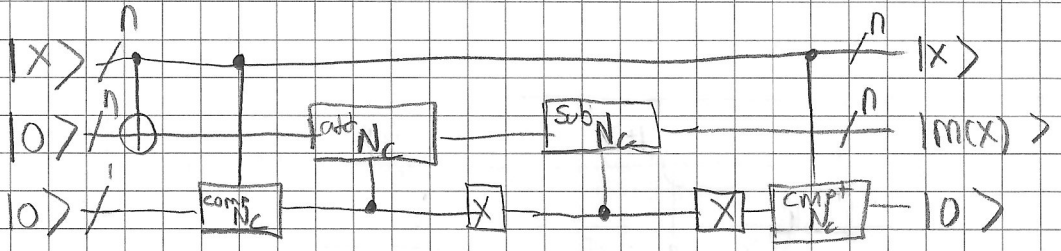
Since there are no negative elements

$$S = \underline{I}$$

$H_{-1}^{impl} =$

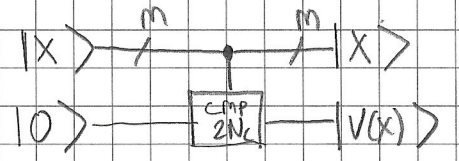


$$M_{-1}|x\rangle|0\rangle \mapsto \begin{cases} |x\rangle|x+N_c\rangle & \text{if } x<N_c \\ |x\rangle|x-N_c\rangle & \text{else} \end{cases}$$



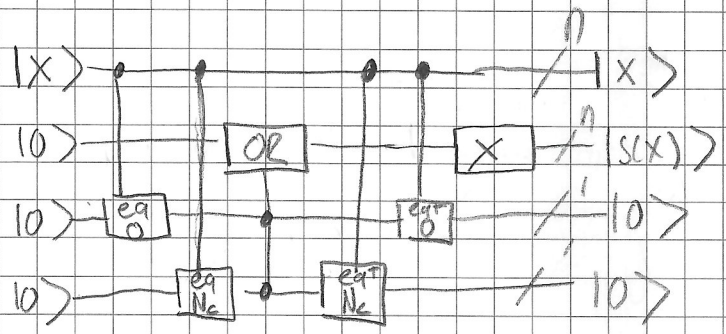
M_{-1}

$$V_{-1}|x\rangle|0\rangle \mapsto \begin{cases} |x\rangle|1\rangle & \text{if } x<2N_c \\ |x\rangle|0\rangle & \text{else} \end{cases}$$



V_{-1}

$$S_{-1}|x\rangle|0\rangle \to \begin{cases} |x\rangle|0\rangle & \text{if } (x=0) \vee (x=N_c) \\ |x\rangle|1\rangle & \text{else} \end{cases}$$



Conclusion: Although this method of simulation has better scaling than classical computation, there are so many ancillary "wasted" qubits that classical computers are faster and more accurate for any "sane" problem size

→ The culprit is the "quantum translated traditional operations, such as cmp, add etc"

A better strategy would be to combine the strength of quantum computer with the strength of classical computer. "Hybrid computation"

Other Hamiltonian simulation strategies

Linear Combination unitaries (LCU)

$|\psi\rangle$ is a n qubit state

M is a $2^n \times 2^n$ matrix

we want to prepare

$$\frac{M|\psi\rangle}{\|M|\psi\rangle\|}$$

Suppose M is a linear combination of unitaries

$$M = \sum_{j=1}^{m} \alpha_j V_j$$

(this does not mean M itself is unitary) α_j are non-negative reals.

$$\|\alpha_j\|_1 = \sum_j \alpha_j$$

Let W be a unitary acting on $\lceil \log m \rceil$ qubits that maps $|0\rangle \mapsto \frac{1}{\sqrt{\|\alpha\|_1}} \sum_j \sqrt{\alpha_j}\,|j\rangle$ also notice

$V = \sum_{j=1}^{m} |j\rangle\langle j| \otimes V_j$, hence it maps $|j\rangle|\phi\rangle \to |j\rangle V_j|\phi\rangle$

i.e first register "selects" which unitary V_j to apply to second register

Implement M using V and W

1) Start with two register state $|0\rangle|\psi\rangle$ ($\log m$ qubits)
2) Apply W to first register
3) Apply V to the whole state
4) Apply W^{-1} to the first register

This gives

$$\frac{1}{\|\alpha\|_1}|0\rangle M|\psi\rangle + \sqrt{1 - \frac{\|M|\psi\rangle\|^2}{\|\alpha\|_1^2}}\,|\phi\rangle$$

$|\phi\rangle$ is not something we care about. What we do is to find 0 outcome in first register that has a probability $p = \|M|\psi\rangle\|^2 / \|a\|_1^2$

however one can use amplitude amplification methods and once we amplify $|0\rangle$ in first register, the second register collapses to normalised version of $M|\psi\rangle$

Using the Taylor series

$$e^{iHt} = \sum_{k=0}^{\infty} \frac{(iHt)^k}{k!} = \sum_{k=0}^{\infty} \frac{(it)^k}{k!} \left(\sum_{j \in |m|} \alpha_j V_j \right)^k$$

$$= \sum_{k=0}^{\infty} \frac{(it)^k}{k!} \sum_{j_1 \dots j_k \in |m|} \alpha_{j_1} \dots \alpha_{j_k} V_{j_1} \dots V_{j_k}$$

Transforming block encoded matrices

This is a very general and flexible method

A is a n-qubit matrix $\|A\| \leq 1$

We can construct a $n+1$ qubit matrix as

$$U = \begin{pmatrix} A & \cdot \\ \cdot & \cdot \end{pmatrix}$$

where each $\cdot$ is a $2^n \times 2^n$ dimensional matrix

Since U is unitary

$$U: |0\rangle|\psi\rangle \longmapsto |0\rangle A|\psi\rangle + |1\rangle|\phi\rangle$$

We can't say much about subnormalised state $|\phi\rangle$ but,

$$(\langle 0| \otimes I)\, U\, (|0\rangle \otimes I) = A$$

Theorem 1 Let $P : [-1,1] \to \{c \in \mathbb{C} \mid |c| \leq 1/4\}$ be a degree-d polynomial and let U be a unitary a-qubit block-encoding of Hermitian matrix A. We can implement a unitary $O(a)$-qubit block encoding V of $P(A)$ using d applications of U and $U^\dagger$ one controlled application of U and $O(ad)$ other qubit gates

(see Singular value decomposition)

We want to approximate $f(x) = e^{ixt}$ using a polynomial P degree d (the first d terms of Taylor series of f)
Division by 4 makes the range obey the theorem
block encoding V of $P(H) \approx \frac{1}{4} e^{iHt}$

$$V : |0\rangle|\Psi\rangle \to |0\rangle P(H)|\Psi\rangle + |\phi\rangle$$

with a boost operation on $\frac{1}{4}$, gives the $P(H)|\Psi\rangle$

Amplitude amplification boost (Grover's algorithm)

1) $|U\rangle = A|0\rangle$
2) $O(1/p)$ times apply O_X and ARA^{-1}
3) Measure the first register