Imperial College
London

INDIVIDUAL STUDY OPTION

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Quantum programming

Author:
Tristan NEMOZ (CID: 01811909)
`tristan.nemoz19@imperial.ac.uk`

Supervisor:
Mario BERTA
`m.berta@imperial.ac.uk`

Date: April 30, 2020

A GENERIC QUANTUM CIRCUIT FOR A RECOMMENDATION SYSTEM

Tristan NEMOZ
Imperial College London
MSc Computing (Security & Reliability
`tristan.nemoz19@imperial.ac.uk`

April 30, 2020

ABSTRACT

In this work, we design a generic quantum circuit to implement a recommendation system following the algorithm of Kerenidis and Prakash. This algorithm takes as input a $m \times n$ binary matrix which has a good rank-k approximation and returns a product recommendation for an user in time $O(\log(k)\ \log(m\,n))$.
We discuss the limitations of implementing this algorithm on a real quantum computer and the differences between the high-level description of the algorithm and its low-level implementation. In particular, our implementation's complexity matches the original algorithm' one of $O(\text{polylog}(m\,n)\ \text{poly}(k))$, but succeeds with only a probability of $1 - \frac{1}{\log(m\,n)}$, where the original algorithm succeeds with probability $1 - \frac{1}{\text{poly}(m\,n)}$.

1 Introduction

In spite of the lack of actual quantum computers, quantum computing has been studied for several dozens of year because of its promising results. Since the proposal of an algorithm for prime factorization [11] by Shor in 1997, several other quantum algorithms were designed to provide a solution to classical problems with an exponential speed-up in complexity.

Amongst them is the HHL algorithm [4] whose goal is to solve a linear system $\mathbf{A}\,\mathbf{x} = \mathbf{y}$ where $\mathbf{A} \in \mathbf{R}^{n\times n}$ in time $O\left(\log(n)\,\kappa^2\right)$, κ being the condition number of $\mathbf{A}$. Further studies of this algorithm generalized its routines to apply them to other problems. For instance, creating efficiently a quantum state $|x\rangle$ corresponding to a real vector $\mathbf{x} \in \mathbf{R}^n$, which is essential to use the algorithm, has been found to be possible using Quantum RAM [3, 7, 10].

HHL algorithm was one of the first algorithms to use Quantum Computing for solving a Machine Learning problem. Other followed and papers have continued to be published to apply Quantum Computing to specific Machine Learning tasks like Deep Convolutional Neural Networks [5].

In this context, Kerenidis and Prakash proposed in 2016 a quantum algorithm for a recommendation system which provided a solution with an exponential speed-up gain in time complexity regarding classical solutions at that time.

We present a generic quantum circuit for implementing this algorithm so that it can be easily reproducible using any framework, given an implementation of a Quantum RAM, that is, a general circuit made of elementary gates that matches the original algorithm. Because of the limitations of the quantum hardware, which are discussed in subsection 2.2, only a limited set of gates can be used. Furthermore, some operations used in the original algorithm are not currently possible to implement on a quantum computer. For such problems, we present workarounds that leave the complexity untouched but cut the probability of success of the algorithm.

Hence, our goal is to present the differences between a theoretic implementation of the algorithm of Kerenidis and Prakash and a currently real one.

This paper is organized as follows:

1. In section 2, we provide the reader with basics of Quantum Computing, so that this document is *mostly* self-contained.
2. In section 3, we provide a high-level description of the algorithm of Kerenidis and Prakash, as well as a formal mathematical definition of the problem. We also present in this section the challenges to be tackled.
3. In section 4, we present our solutions to cope with these problems when implementing the algorithm on a real quantum computer.

2 Quantum computing preliminaries

In this section, we ought to present the fundamentals of Quantum Computing that will be used throughout this document in subsection 2.1. Note that only the notions that will be used in the actual implementation of the algorithm will be presented.

Once the formalism of Quantum Computing presented, we discuss the limitations enforced by the current quantum computers in subsection 2.2.

2.1 Quantum computing theory

The principle of Quantum Computing is to design a formalism so that one can encode information as binary words, just like in classical computing. Linking this formalism to the physics principles is a problem we will not discuss here, but whose history is summarized in [1].

2.1.1 Quantum computing objects

Quantum computing operates with qubits, which are unitary vectors of $\mathbf{C}^2$. Such a vector represent a quantum particle state which we denote $|x\rangle$. We define $|0\rangle$ and $|1\rangle$ to be the vectors of the computational basis of $\mathbf{C}^2$ seen as a $\mathbf{C}$-vector field. As such, given $|x\rangle \in \mathbf{C}^2$ we can write:

$$|x\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha \begin{pmatrix}1\\0\end{pmatrix} + \beta \begin{pmatrix}0\\1\end{pmatrix} = \alpha\,|0\rangle + \beta\,|1\rangle$$

with $\alpha\,\overline{\alpha} + \beta\,\overline{\beta} = 1$. $|x\rangle$ is then said to be a superposition of the vector of the computational basis.

For a given matrix $\mathbf{A}$, we define $\mathbf{A}^\dagger$ to be the conjugate of the transposed matrix of $\mathbf{A}$, that is:

$$\mathbf{A}^\dagger \overset{\text{def}}{=} \overline{\mathbf{A}^\top} = \overline{\mathbf{A}}^\top .$$

Similarly, we define $\langle x|$ to be:

$$\langle x| \overset{\text{def}}{=} |x\rangle^\dagger .$$

Each quantum state $|x\rangle$ evolution satisfies Schrödinger's equation:

$$\mathrm{i}\,\hbar\,\frac{\partial}{\partial t}\,|x\rangle = \mathbf{H}\,|x\rangle$$

with $\mathbf{H}$ being a self-adjoint matrix:

$$\mathbf{H}^\dagger = \mathbf{H}.$$

The solution to Schrödinger's equation is given by $\exp\left(-\frac{\mathrm{i}}{\hbar}\,\mathbf{H}\,t\right)$, which is unitary:

$$\exp\left(-\frac{\mathrm{i}}{\hbar}\,\mathbf{H}\,t\right)^\dagger \exp\left(-\frac{\mathrm{i}}{\hbar}\,\mathbf{H}\,t\right) = \mathbf{I}.$$

As a consequence, every operation $\mathbf{U}$ performed on a quantum state $|x\rangle$ must be unitary. Reciprocally, every unitary matrix $\mathbf{U}$ can be applied to a quantum state $|x\rangle$. Such an operation is called a quantum gate.

A quantum algorithm operates with several qubits. Let $|\varphi\rangle$ and $|\psi\rangle$ be two qubits. Then the quantum state containing $|\varphi\rangle$ in a first register and $|\psi\rangle$ in a second register is given by the tensor product $|\varphi\rangle \otimes |\psi\rangle$ of $|\varphi\rangle$ and $|psi\rangle$. An usual convention is to omit the symbol of the tensor product. Hence, such a state would be noted $|\varphi\rangle\ |\psi\rangle$, or even $|\phi\psi\rangle$. Then, if a gate $\mathbf{U}_1$ is applied on $|\varphi\rangle$, and a gate $\mathbf{U}_2$ is applied on $|\psi\rangle$, then the resulting quantum state is given by:

$$(\mathbf{U}_1 \otimes \mathbf{U}_2)\ (|\varphi\rangle \otimes |\psi\rangle)\ .$$

To make an analogy with classical computing, a quantum algorithm encodes information as a tensor product of several qubits, apply one or more quantum gates to them, and then read out the result. Both the input and the output of a quantum algorithm is a tensor product of qubits, each possibly in a superposed state.

The big difference with classical computing however, is that every operation performed on a quantum state is reversible, since every operation is represented by a unitary, hence inversible, matrix. The only exception to this rule is the reading operation, also called the Measurement operation, described in subsubsection 2.1.3.

2.1.2 Quantum circuit

Let $|x\rangle$ and $|y\rangle$ be two qubits and let $\mathbf{U}_1$, $\mathbf{U}_2$ and $\mathbf{U}_3$ be three quantum gates that can be applied on single qubits. Applying $\mathbf{U}_1$ followed by $\mathbf{U}_2$ on $|x\rangle$ while applying $\mathbf{U}_3$ to $|y\rangle$ is graphically represented as shown below:

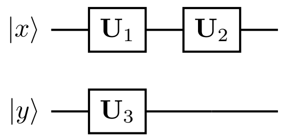


Figure 1: An example of a quantum circuit

or equivalently, since $\mathbf{U}_1 \otimes \mathbf{U}_2$ is a $2^2 \times 2^2$ unitary matrix, and as such is a quantum gate that operates on two qubits:

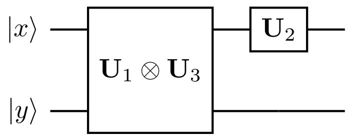


Figure 2: An example of a quantum circuit with a gate applied on several qubits

It is possible to apply a gate on a given qubit $|y\rangle$ only if another qubit is in the state $|1\rangle$. Such a gate is called a controlled gate. Let us take the following circuit as an example:

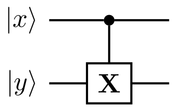


Figure 3: An example of a quantum circuit with a conditoned gate

Here, $\mathbf{X}$ is the Pauli $\mathbf{X}$-gate, which will be extensively used throughout this document:

$$\mathbf{X} \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\ .$$

$\mathbf{X}$ is also called the NOT gate, since it maps the state $|0\rangle$ to the state $|1\rangle$ and reciprocally. Let us write $|x\rangle$ as $|x\rangle = x_1\ |0\rangle + x_2\,|1\rangle$ and $|y\rangle$ as $y_1\ |0\rangle + y_2\ |1\rangle$. Then the global quantum state at the beginning of this circuit is given by:

$$|x\rangle \otimes |y\rangle = x_1\,y_1\ |00\rangle + x_1\,y_2\ |01\rangle + x2\,y_1\ |10\rangle + x_2\,y_2\ |11\rangle\ .$$

Applying $\mathbf{X}$ conditioned on $|x\rangle$ means that the gate will be applied on the parts of the state that have $|0\rangle$ on their first qubits. Hence, the resulting state of this circuit is:

$$x_1\, y_1\; |00\rangle + x_1\, y_2\; |01\rangle + x2\, y_2\; |10\rangle + x_2\, y_1\; |11\rangle\ .$$

Note that for convenience purposes, it is also possible to apply a gate when a given qubit is in the state $|0\rangle$, which we represent like this:

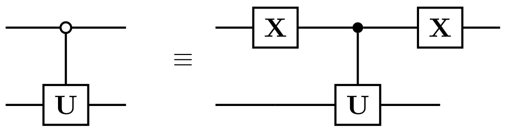


Figure 4: An example of a quantum circuit with a gate conditioned on the state $|0\rangle$

2.1.3 Measurement

Let $|x\rangle$ be a quantum state of n qubits. We denote by $|k\rangle$ the k-th vector of the computational basis of $\mathbf{C}^{2^n}$. The notation $|k\rangle$ will always be used in a non-ambiguous way. Hence, it will only be used to write a quantum state whose number of qubits is known. As such, we have:

$$|x\rangle = \sum_{k=1}^{n} \alpha_k\; |k\rangle$$

with $\alpha_k \in \mathbf{C}$ for $k \in [\![1\,;\,n]\!]$. Measuring $|x\rangle$ in the computational basis means projecting $|x\rangle$ onto one vector of the computational basis of $\mathbf{C}^{2^n}$. Measuring is the only operator that one is allowed to apply on a quantum state that is not unitary. The state onto which $|x\rangle$ is mapped is randomly determined by its associated coefficient. More clearly, $|x\rangle$ will be mapped on the state $|k\rangle$ with probability $\alpha_k\, \overline{\alpha_k}$. The measurement allows us to classically known onto which state $|x\rangle$ has been projected. The representation of a measurement in a quantum circuit is the following:

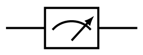


Figure 5: Representation of a measurement

Measuring is usually the last thing performed in a quantum algorithm, since it is the only way to go from a quantum state to a classical one. However, a quantum algorithm designed to be used a part of another, bigger, quantum algorithm, does not necessarily perform a measurement. The HHL algorithm [4] is often used as such.

2.1.4 Entanglement

A state $|x\rangle$ is entangled whenever it is not possible to find n qubits $|q_1\rangle$, $|q_2\rangle$, $\cdots$, $|q_n\rangle$ such that:

$$|x\rangle = \bigotimes_{k=1}^{n} |q_k\rangle\ .$$

But why is entanglement important? Let us consider the following circuit:

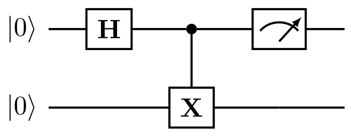


Figure 6: A quantum circuit for a demonstration of the impact of entanglement on measuring

Here, $\mathbf{H}$ is the Hadamard gate, which maps $|0\rangle$ to $\frac{1}{\sqrt{2}}\,(|0\rangle + |1\rangle)$. Hence, just before the measurement, the quantum state is given by:

$$|\varphi\rangle = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle\ .$$

When measuring $|\varphi\rangle$, we will obtain $|0\rangle$ with probability $\frac{1}{2}$ or $|1\rangle$ also with probability $\frac{1}{2}$. But even though we only measured the first qubit, we forced $|\varphi\rangle$ to be either in the state $|00\rangle$ or $|11\rangle$ after the measurement. Hence, measuring the first qubit affected somehow the second qubit.

Often, efficient quantum algorithms make use of entangled state. For this reason, it is crucial to ensure when measuring a quantum state that it is not entangled.

2.1.5 Quantum gates and algorithms

Since a quantum algorithm is actually applications of quantum gates, we can show that it is of course also unitary. For this reason, we often do not make the difference between a quantum gate and a quantum algorithm, especially when no measurement is performed.

We need to introduce some gates that will be used in our implementation and provide their complexity. We define a set of elementary gates:

$$\mathbf{X} \overset{\text{def}}{=} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \mathbf{Y} \overset{\text{def}}{=} \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix} \quad \mathrm{Z} \overset{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \mathrm{H} \overset{\text{def}}{=} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \mathrm{R}_\theta \overset{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\,\theta} \end{pmatrix} \quad \mathrm{I} \overset{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

with $\theta \in \mathbf{R}$. For every matrix $\mathbf{U}$ in this elementary set, we also define its controlled equivalent $\mathbf{CU}$ as elementary. Note that this does not include CU itself: a gate that is controlled on more than 2 qubits must be implemented using gate controlled on a single qubit. Finally, we add one more elementary gate (without adding its controlled equivalent), which is the Toffoli gate $\mathbf{CCX}$, also known as the CCNOT gate. Its circuit representation is:

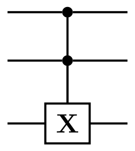


Figure 7: The Toffoli gate

Hence, the set $\mathcal{E}$ os elementary gates is:

$$\mathcal{E} \overset{\text{def}}{=} \{\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{CX}, \mathbf{CZ}, \mathbf{CCX}\}\,.$$

Assumption 2.1. *Every gate of $\mathcal{E}$ can be implemented in $O(1)$.*

Hence, we can define the complexity of a gate by decomposing it in a succession of elementary gates.

Definition 2.1 (SWAP gate). *The swap gate is used to exchange the value of two qubits. It is implemented as follows:*

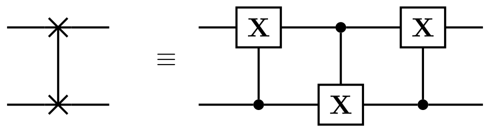


Figure 8: The SWAP gate

Hence, the time complexity of a SWAP gate is also $O(1)$.

Definition 2.2 (Quantum Fourier Transform). *The Quantum Fourier Transform gate* $\mathbf{F}$ *matches the classical definition of the Fourier transform. Hence, we have, for n qubits:*

$$\forall j \in [\![0\,;\,2^n-1]\!]\,, \mathbf{F}\ |j\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \mathrm{e}^{\frac{2\,\mathrm{i}\,\pi\,k\,j}{2^n}}\ |k\rangle$$

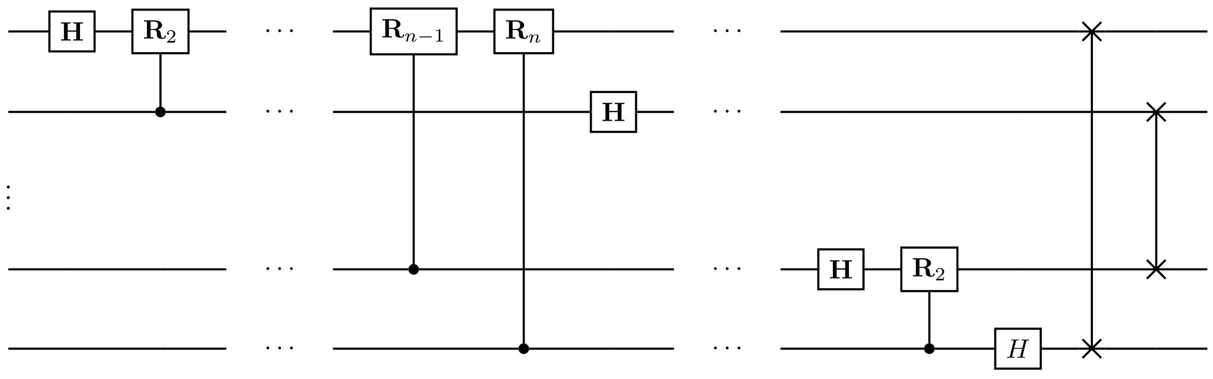


Figure 9: Implementation of the Quantum Fourier Transform

Its implementation is given in [8]:

where $\mathbf{H}$ *is the Hadamard gate:*

$$\mathbf{H} \overset{def}{=} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

and where $\mathbf{R}_k$ *is defined as, for* $k \in [\![2\,;\, n]\!]$*:*

$$\mathbf{R}_k \overset{def}{=} \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\frac{2\,\mathrm{i}\,\pi}{2^k}} \end{pmatrix} .$$

We assume both gates can be implemented in $O(1)$*.*

Note that is it possible to see $\mathbf{F}$ *implemented without the SWAP gates at the end for performance reasons. Thanks to this decomposition, we can implement* $\mathbf{F}$ *in time* $O\left(n^2\right)$*.*

Definition 2.3 (Quantum phase estimation).

2.2 Quantum hardware considerations

Whilst the previous formalism is useful to design quantum gates, it is important to keep in mind that some operations are not allowed, or are subject to some limitations because of current quantum hardware. Two of them are particularly important.

3 A quantum recommendation system

In this section, we intend to describe what a recommendation system is and how the quantum algorithm of Kerenidis and Prakash solves it.

3.1 Principles of a recommendation system

3.2 A quantum algorithm as a recommendation system

4 Designing a quantum circuit for a recommendation system

4.1 Loading from QRAM

4.2 Applying the Quantum Phase Estimation algorithm

Now that the state $|x\rangle$ has been loaded from QRAM, we are to append a quantum register of size $\lceil \log(n) \rceil$ before the quantum register in which $|x\rangle$ has been loaded and to apply a unitary $\mathbf{Q}$ to get the state:

$$|\mathbf{Q}\, x\rangle = \sum_i \alpha_i \left|\widetilde{\mathbf{A}}, j\right\rangle .$$

Since $\widetilde{\mathbf{A}}$ is easily loadable via QRAM, we only have to apply the loading gate $\mathbf{L}_{\widetilde{\mathbf{A}}}$ to the first register to get the desired state.

4.3 Separating states according to a threshold

Once the QPE has been applied, the state of the system is $\sum_i \alpha_i\, |v_i\rangle\, |a_i\rangle$, where $|a_i\rangle$ is the output of the phase estimation algorithm associated to the eigenvector $|v_i\rangle$ of $\mathbf{U}\,\mathbf{V}$, that is:

$$\forall i \in [\![1\,;\,\lceil \log(n)\rceil]\!], \mathbf{U}\,\mathbf{V}\, |v_i\rangle \approx \mathrm{e}^{2\,\mathrm{i}\,\pi\, a_i}\, |v_i\rangle .$$

What we want to do now is to define a threshold unitary $\mathbf{T}_\sigma$ such that:

$$\mathbf{T}_\sigma : |t\rangle\, |0\rangle \mapsto \begin{cases} |t\rangle\, |1\rangle & \text{if } t < \sigma\,\left(1-\frac{\kappa}{2}\right) \\ |t\rangle\, |0\rangle & \text{otherwise} \end{cases} .$$

In the original paper, $\mathbf{T}_\sigma$ is applied on the second quantum register, which contains the approximation of the singular values of $\mathbf{A}$. Hence, what we want to do is to design $\mathbf{T}_\sigma$ so that it can be applied to the second register directly following the QPE. Let us consider an approximation $\overline{\sigma_i}$ of a singular value of $\mathbf{A}$. By definition:

$$\overline{\sigma_i} = \cos\left(\frac{\theta_i}{2}\right)\, \|\mathbf{A}\|_F$$

with $\theta_i \in [-\pi\,;\,\pi[$. Since $a_i \in [0\,;\,1[$, we can write:

$$\theta_i = \begin{cases} 2\,\pi\, a_i & \text{if } a_i \in \left[0\,;\,\frac{1}{2}\right[\\ 2\,\pi\,(a_i - 1) & \text{if } a_i \in \left[\frac{1}{2}\,;\,1\right[\end{cases} .$$

Indeed, for $a_i \in \left[0\,;\,\frac{1}{2}\right[$, θ_i grows linearly from 0 to π. At $a_i = \frac{1}{2}$, θ_i grows linearly from $-\pi$ to 0.

Hence, the following holds:

$$\begin{aligned} & \sigma_i < \sigma\,\left(1-\frac{\kappa}{2}\right) \\ \iff & \cos\left(\frac{\theta_i}{2}\right) < \frac{\sigma\,\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F} \\ \iff & \exists k \in \mathbf{Z}, \pm\frac{\theta_i}{2} \in \left]2\,k\,\pi + \arccos\left(\frac{\sigma\,\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right)\,;\, 2\,k\,\pi + \frac{\pi}{2}\right] . \end{aligned}$$

Since $\frac{\theta_i}{2} \in \left[-\frac{\pi}{2}\,;\,\frac{\pi}{2}\right[$, this is equivalent to:

$$\sigma_i < \sigma\,\left(1-\frac{\kappa}{2}\right) \iff \left|\frac{\theta_i}{2}\right| \in \left]\arccos\left(\frac{\sigma\,\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right)\,;\,\frac{\pi}{2}\right] .$$

By replacing θ_i by its definition, this gives us:

$$\sigma_i < \sigma\,\left(1-\frac{\kappa}{2}\right) \iff \begin{cases} a_i \in \left]\frac{1}{\pi}\,\arccos\left(\frac{\sigma\,\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right)\,;\,\frac{1}{2}\right] & \text{if } a_i < \frac{1}{2} \\ 1 - a_i \in \left]\frac{1}{\pi}\,\arccos\left(\frac{\sigma\,\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right)\,;\,\frac{1}{2}\right] & \text{if } a_i \geqslant \frac{1}{2} \end{cases} .$$

Which gives us our final criteria:

$$\sigma_i < \sigma\left(1-\frac{\kappa}{2}\right) \iff \begin{cases} \frac{1}{\pi}\arccos\left(\frac{\sigma\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right) < a_i & \text{if } a_i < \frac{1}{2} \\ 1-\frac{1}{\pi}\arccos\left(\frac{\sigma\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right) > a_i & \text{if } a_i \geqslant \frac{1}{2} \end{cases}.$$

Since we know σ and κ beforehand, and since $\|\mathbf{A}\|_F$ is easily accessible, we can compute $b_1 = \frac{1}{\pi}\arccos\left(\frac{\sigma\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right)$ and $b_2 = 1-b_1$ efficiently and find their binary writing $\overline{b_{1,1}b_{1,2}\cdots b_{1,\lceil\log(n)\rceil}}^2$ and $\overline{b_{2,1}b_{2,2}\cdots b_{2,\lceil\log(n)\rceil}}^2$ representing them. Note that since $\frac{\sigma\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F} \in]0\,;\,1]$, we have $b_1 \in \left[0\,;\,\frac{1}{2}\right[$, hence $b_{1,1}=0$ and $b_{1,2}=1$.

Let b represent either b_1 or b_2. The goal is now to compare b and a_i bitwise to determine which one is superior. We consider the state $|\overline{a_i}\rangle = \left|a_{i,2}\cdots a_{i,\lceil\log(n)\rceil}\right\rangle$, which represents a_i if $a_i < \frac{1}{2}$, that is $a_{i,1}=0$, and $a_i - \frac{1}{2}$ if $a_i > \frac{1}{2}$, that is $a_{i,1}=1$. Hence, the algorithm will go through $\lceil\log(n)\rceil - 1$ steps for each one of b_1 and b_2. Without loss of generality, we will for now consider the case $a_i < \frac{1}{2}$, that is $b=b_1$. We identify each of the steps by the number of the bit it applies to. Hence, the first step is identified with 2 since there is no need to test the fist bit, the second one with 3 and the last one with $\lceil\log(n)\rceil$. Hence, at step j, the algorithm operates on qubit $|a_{i,j}\rangle$ and bit b_j.

Some circuits already exist for quantum bit-string comparison. For instance Oliveira and Ramos and describes different circuits to do such a task in [9]. However, these circuits use a whole quantum register to store one of the two strings. In order to take advantage from the fact that we have a classical representation of b, we build our own circuit to deal with this problem.

The first idea for constructing the circuit is to add an ancilla qubit whose goal is to determine whether the two bits are equal:

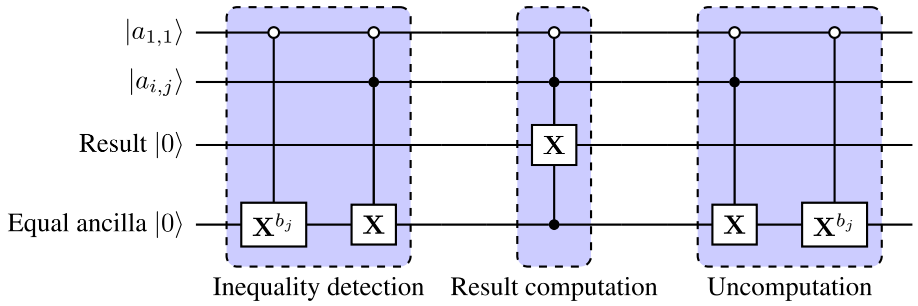


Figure 10: Idea of a partial implementation of $\mathbf{T}_\sigma$ using one ancilla qubit

By doing so, the result qubit will be in the state $|1\rangle$ if and only if $b_j < a_{i,j}$. Note that if we want the result to be $|1\rangle$ if $b > a_i$, when $a_i \geqslant \frac{1}{2}$ for instance, we would replace the $\mathbf{X}$ gate controlled on $a_{i,j}$ by a $\mathbf{X}^{b_j}$ gate. Of course, the problem is that it will be swapped once more if this situation happens once again. Hence, we need another ancilla qubit to know whether the circuit already has determined that $a_i > b$, as shown on the following:

The idea is to test for the equality between the bits only when the cicruit hasn't already determined whether $a_i > b$. But this design also has a flaw. Indeed, let us assume that the first inequality detected $a_{i,j} \neq b_j$ is at bit j. Now, we want to uncompute the state of the equal ancilla from $|1\rangle$ to $|0\rangle$. While the first intuition to do this is drawn on the circuit above, this implementation is incorrect. Indeed, we have:

$$|\varphi\rangle = |a_{i,j}\rangle\left|1^{b_j}\right\rangle|1\rangle\,|1\rangle$$

Hence, the $\mathbf{X}$ and $\mathbf{X}^{b_j}$ gates of the uncomputation steps won't be applied on the equal ancilla, since the result found ancilla is in the state $|1\rangle$. Controlling on the value $|0\rangle$ would not work either, since this would be applied at each following step, since the result found ancilla is supposed to stay in the state $|1\rangle$. Hence, we need to add another ancilla qubit whose goal is to store whether the result is set at the begininng of a step. This leads to another version of the partial implementation of $\mathbf{T}_\sigma$:

But there is still one problem: at the $j+1$ step, the ancilla will change its value again, eventually changing other registers. Hence, we need an ancilla for every step, which can by symbolized with the following circuit:

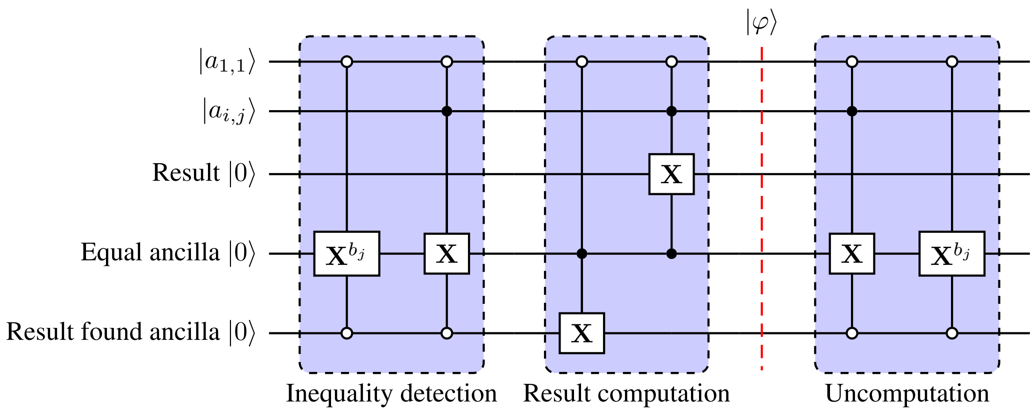


Figure 11: Idea of a partial implementation of $\mathbf{T}_\sigma$ using two ancilla qubits

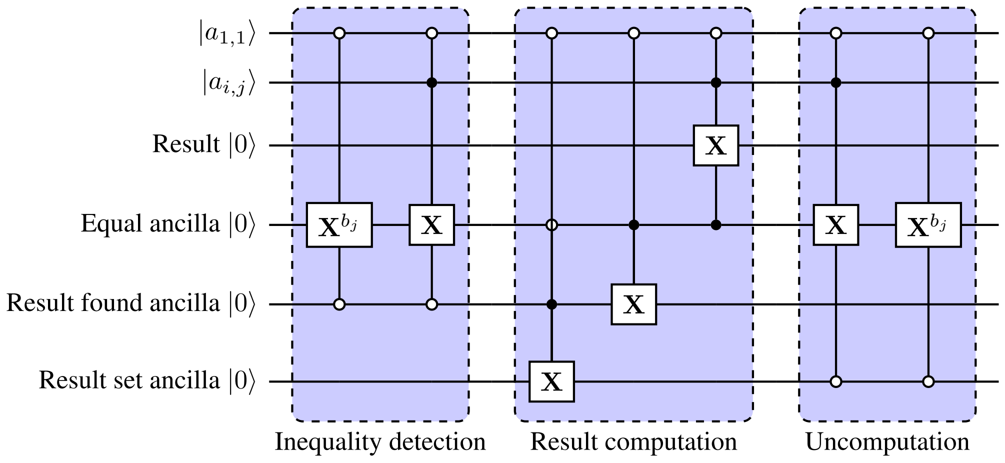


Figure 12: Partial implementation of $\mathbf{T}_\sigma$ using three ancilla qubits

Note that aside from the Equal ancilla and the Result found ancilla, we only need $n-1$ ancilla qubits. This is due to the fact that the very first step does not need such an ancilla qubit, since we know that the Result found ancilla will be in state $|0\rangle$.

Plus, since every operation is controlled on $a_{i,1}$, we can simply concatenate the two circuits to use the same ancillas for the case $a_i > \frac{1}{2}$. At every step, 7 controlled gates are applied. Hence, the total complexity of this circuit is linear in the number of steps it makes, that is $O(\log(n))$.

There are still two corner cases which we need to tackle, that are the cases $a_i = \frac{1}{2}$ and $a_i = b$. If $a_i = \frac{1}{2}$, it will only be considered by the second part of the circuit. Since $b_1 < \frac{1}{2}$, we know that $b_2 > \frac{1}{2}$. Hence, the second part will detect that $b_2 > a_i$ and outputs the state $|1\rangle$, which is what we desire. Concerning the case $a_i = b$, the circuit will never detect an inequality. Hence, it will output the state $|0\rangle$, which is also what we want. Indeed, as a recall, we have:

$$\sigma_i < \sigma\left(1-\frac{\kappa}{2}\right) \iff \begin{cases} \frac{1}{\pi}\arccos\left(\frac{\sigma\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right) < a_i & \text{if } a_i < \frac{1}{2} \\ 1-\frac{1}{\pi}\arccos\left(\frac{\sigma\left(1-\frac{\kappa}{2}\right)}{\|\mathbf{A}\|_F}\right) > a_i & \text{if } a_i \geqslant \frac{1}{2} \end{cases}.$$

In particular, $a_i = b$ implies that $\sigma_i \geqslant \sigma\left(1-\frac{\kappa}{2}\right)$. As such, the output of $\mathbf{T}_\sigma$ is to be $|0\rangle$.

5 Conclusion

Global QSVET:

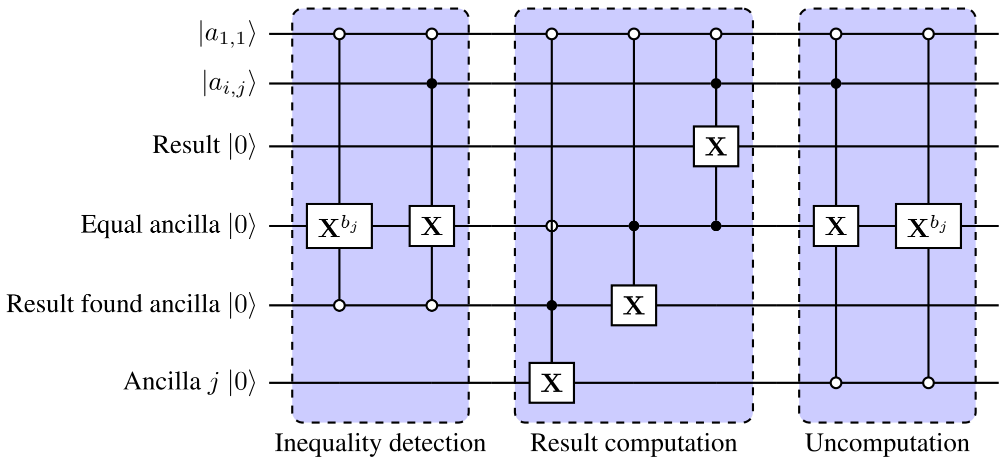
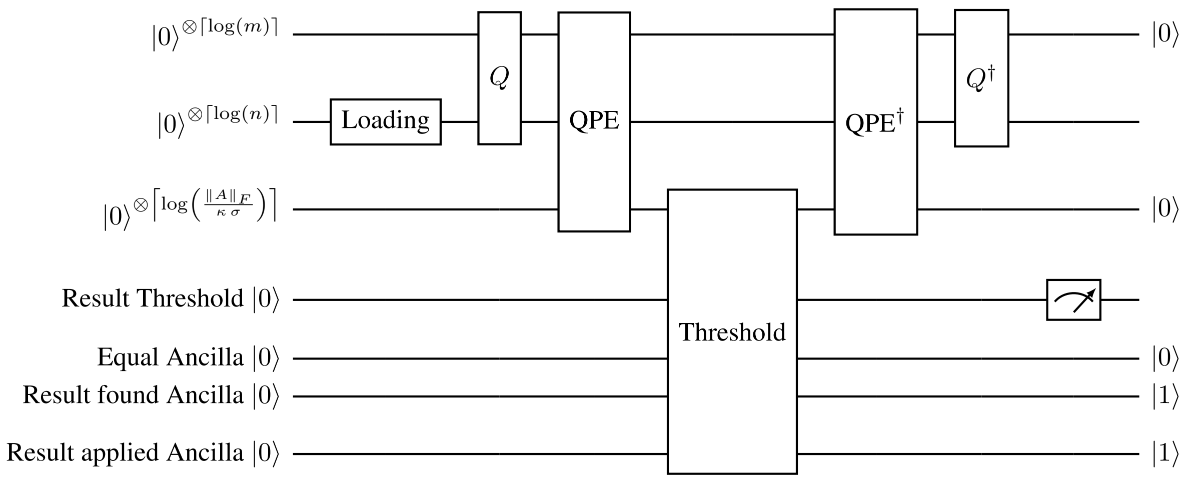


Figure 13: Partial implementation of $\mathbf{T}_\sigma$ using $\lceil \log(n) \rceil + 1$ ancilla qubits


$|0\rangle^{\otimes \lceil \log(m) \rceil}$
$|0\rangle^{\otimes \lceil \log(n) \rceil}$
$|0\rangle^{\otimes \left\lceil \log\left(\frac{\|A\|_F}{\kappa\,\sigma}\right)\right\rceil}$
Result Threshold $|0\rangle$
Equal Ancilla $|0\rangle$
Result found Ancilla $|0\rangle$
Result applied Ancilla $|0\rangle$
Loading
Q
QPE
Threshold
QPE†
$Q^\dagger$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|1\rangle$
$|1\rangle$

References

[1] Carlos M. Madrid Casado. *A brief history of the mathematical equivalence between the two quantum mechanics*. URL: `http://www.lajpe.org/may08/09_Carlos_Madrid.pdf`.

[2] Danial Dervovic *et al. Quantum linear systems algorithms: a primer*. 2018. arXiv: `1802.08227 [quant-ph]`.

[3] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. *Quantum Random Access Memory*. 2008. DOI: `10.1103/PhysRevLett.100.160501`. arXiv: `0708.1879 [quant-ph]`.

[4] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* 103.15 (2009). ISSN: 1079-7114. DOI: `10.1103/physrevlett.103.150502`. URL: `http://dx.doi.org/10.1103/PhysRevLett.103.150502`.

[5] Iordanis Kerenidis, Jonas Landman, and Anupam Prakash. *Quantum Algorithms for Deep Convolutional Neural Networks*. 2019. arXiv: `1911.01117 [quant-ph]`.

[6] Iordanis Kerenidis and Anupam Prakash. *Quantum Recommendation Systems*. 2016. arXiv: `1603.08675 [quant-ph]`.

[7] Olivia Di Matteo, Vlad Gheorghiu, and Michele Mosca. *Fault tolerant resource estimation of quantum random-access memories*. 2019. arXiv: `1902.01329 [quant-ph]`.

[8] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. USA: Cambridge University Press, 2011. ISBN: 1107002176.

[9] David Oliveira and Rubens Ramos. "Quantum bit string comparator: Circuits and applications". In: *Quantum Computers and Computing* 7 (Jan. 2007).

[10] Anupam Prakash. "Quantum Algorithms for Linear Algebra and Machine Learning." PhD thesis. EECS Department, University of California, Berkeley, 2014. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-211.html.

[11] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.