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 $O(\log(k)\ \mathrm{polylog}(m\,n))$.

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, we discuss on the reasons that could improve our design's complexity in the future.

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 [17] 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 [8] 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 [7, 12, 16].

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 [9].

In a similar way of the P=NP problem, we currently do not know how to determine whether a problem is only solvable in logarithmic complexity with a quantum computer. For instance, Shor's algorithm [17] does not have a classical equivalent that matches its complexity yet. However, a recent paper from Tang shows that the Quantum Recommendation system is not one of those. Indeed, they provide a classical algorithm for doing the same task classically in time complexity $O(\mathrm{poly}(k)\ \log(m\,n))$, which is only slower by a polynomial factor than its quantum equivalent. Still, the goal of this paper is to highlight the differences between the theoretic implementation of an algorithm and its real-world implementation, and the Quantum Recommendation System algorithm is a good example for this purpose.

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. Indeed, the classical solutions required to compute a $m \times n$ matrix from which the recommendations were made [11], which means that their complexity was at least $O(m\,n)$. The algorithm described in [10] by Kerenidis and Prakash however, has a time complexity of $O(\log(k)\ \mathrm{polylog}(m\,n))$. Since it does not have to write down neither the entire matrix nor even a row vector from this matrix, it is not subject to the same linear complexity as its classical equivalent.

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. One problem couldn't be tackled though, which is the lack of an actual quantum ram implementation. As a consequence, our implementation's complexity matches the original algorithm' one of $O(\operatorname{polylog}(m\,n)\ \operatorname{poly}(k))$ if and only if the Quantum RAM can be accessed in a superposed state, and succeeds with only a probability of $1 - \frac{1}{\log(m\,n)}$, where the original algorithm succeeds with probability $1 - \frac{1}{\operatorname{poly}(m\,n)}$.

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 [4].

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 $|x\rangle$. A quantum register is defined as a group of qubits. 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:

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

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 invertible, 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:

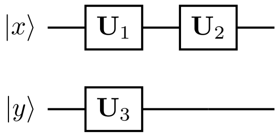

Figure 1: An example of a quantum circuit

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:

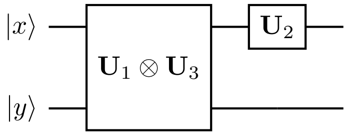

Figure 2: An example of a quantum circuit with a gate applied on several qubits

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:

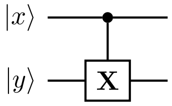

Figure 3: An example of a quantum circuit with a conditoned gate

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

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

$\mathbf{X}$ 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:

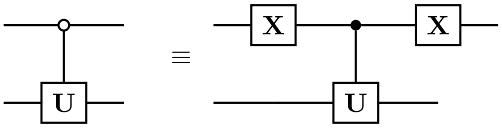

Figure 4: An example of a quantum circuit with a gate conditioned on the state $|0\rangle$

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. Indeed, 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:

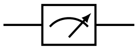

Figure 5: Representation of a measurement

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 [8] is often used as such.

Note that the way it is performed, measuring a state is not sensible to the global phase of a quantum state. For instance, let us consider a state $|\psi\rangle$:

$$|\psi\rangle = \alpha \; |0\rangle + \beta \; |1\rangle \; .$$

Then $|psi\rangle$ can be rewritten as follows:

$$|\psi\rangle = \mathrm{e}^{\mathrm{i}\,\gamma} \; \big(\cos{(\theta)} \; |0\rangle + \mathrm{e}^{\mathrm{i}\,\delta} \; \sin{(\theta)} \; |1\rangle\big)$$

with $(\gamma, \theta, \delta) \in \mathbf{R}^3$. The global phase shift $\mathrm{e}^{\mathrm{i}\,\gamma}$ is physically irrelevant, as unobservable [2].

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:

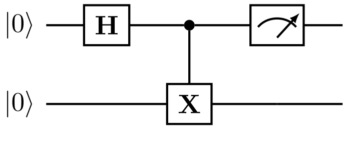

Figure 6: A quantum circuit for a demonstration of the impact of entanglement on measuring

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:

$$\mathbf{X} \overset{\text{def}}{=} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \mathrm{H} \overset{\text{def}}{=} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \mathrm{R}_z(\theta) \overset{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\,\theta} \end{pmatrix} \quad \mathrm{R}_y(\theta) \overset{\text{def}}{=} \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

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

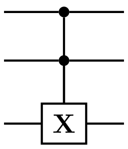

Figure 7: The Toffoli gate

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

$$\mathcal{E} \overset{\text{def}}{=} \{\mathbf{X}, \mathbf{R}_z(\theta), \mathbf{R}_y(\theta), \mathsf{C} - \mathbf{X}, \mathsf{C} - \mathbf{R}_z(\theta), \mathsf{C} - \mathbf{R}_y(\theta), \mathsf{CC} - \mathbf{X}\}\ .$$

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

Assumption 2.1 ensures that we can compute the complexity of a quantum algorithm by decomposing it in a circuit that contains only elementary gates. For instance, a quantum algorithm with time complexity $O(\mathrm{poly}(n))$ uses a polynomial number in n of consecutive gates. Note that gates can be applied in parallel. For instance, the circuit depicted in Figure 8 has a time complexity of $O(1)$ while the one in **??** has a time complexity of $O(n)$.

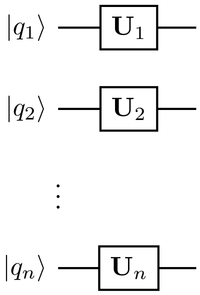

Figure 8: An example of parallel computation

Of course, these examples assume that all the $\mathbf{U}_i$ are elementary gates.
That said, the first gate we introduce is the SWAP gate, whose utility will be described in subsection 2.2.

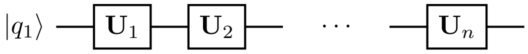

Figure 9: An example of a circuit with $O(n)$ time complexity

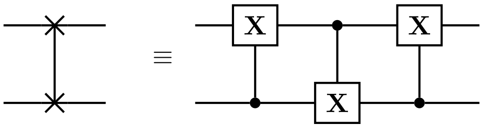

Figure 10: The SWAP gate

Definition 2.1 (SWAP gate). *The swap gate is used to exchange the value of two qubits. It is implemented as shown on Figure 10.*

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

We will need to add more control qubits to our gates. Hence, we describe how to apply a gate on a qubit controlled by n other qubits.

Definition 2.2 (n-controlled gate). *Let $\mathbf{U}$ be a unitary and $n \in \mathbf{N}$. Then it is possible to implement a gate $\mathsf{C}^n - \mathbf{U}$ that acts on $n+1$ qubits such that $\mathbf{U}$ is applied to the $n+1^{th}$ qubit if and only if the n other qubits are in state $|1\rangle$ in polynomial time. The proof and the decomposition is given in [3].*

We also introduce the Quantum Fourier Transform gate. Its only use is within the Quantum Phase Algorithm, but by decomposing things like this, the complexity analysis becomes clearer.

Definition 2.3 (Quantum Fourier Transform). *The Quantum Fourier Transform gate $\mathbf{F}$ matches the classical definition of the Fourier transform. Hence, we have, for n qubits:*

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

Its implementation is given in [14]:

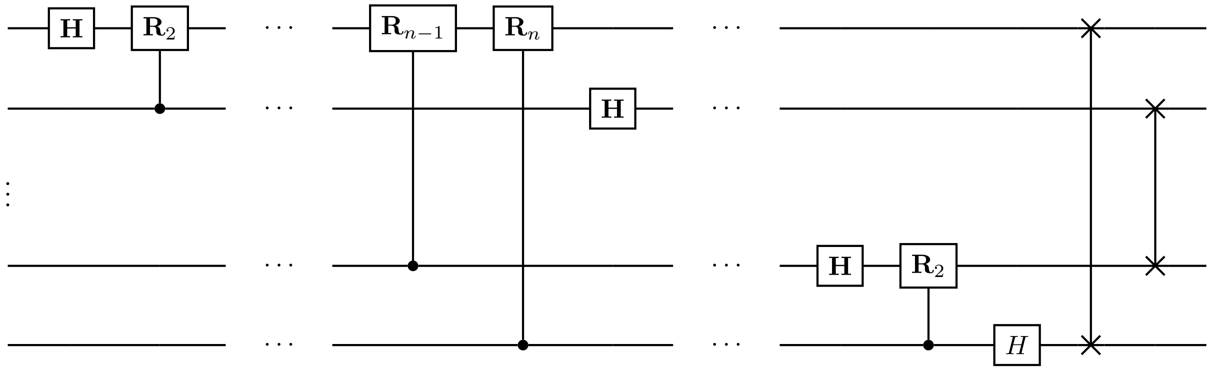

Figure 11: Implementation of the Quantum Fourier Transform

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 $O\left(n^2\right)$.

As said earlier, the utility of defining the Quantum Fourier Transform is to use it within the Quantum Phase Estimation Algorithm, whose goal is to estimate the eigenvalue of a given unitary matrix associated to a given eigenvector of this matrix.

Definition 2.4 (Quantum phase estimation). *Let $\mathbf{U}$ be a unitary matrix. Let $|\psi\rangle$ be an eigenvector of $\mathbf{U}$ associated to the eigenvalue $\mathrm{e}^{2\,\mathrm{i}\,\pi\,\theta}$, with $\theta \in [0\,;\,1]$. Let be $\varepsilon \in \mathbf{R}_+^*$. Let $\overline{\theta}$ be the best approximation of θ on $n = \left\lceil \log_2(\frac{1}{\varepsilon}) \right\rceil$ bits, that is:*

$$\left\|\theta - \overline{\theta}\right\|_1 \leqslant \varepsilon\,.$$

Then with probability at least $\frac{4}{\pi^2}$, the Quantum Phase Estimation algorithm maps the state $|0\rangle^{\otimes n}\,|\psi\rangle$ to the state $\left|\overline{\theta}\right\rangle\,|\psi\rangle$ in time $O\left(2^n\,n^2\,T_{\mathbf{U}}\right)$, where $T_{\mathbf{U}}$ is the time taken for implementing $\mathbf{U}$.

The actual implementation of the $\mathbf{QPE}$ gate using $\mathbf{QFT}^\dagger$ is well-known in the literature, and presented for instance in [13]. Its quantum circuit is the following one:

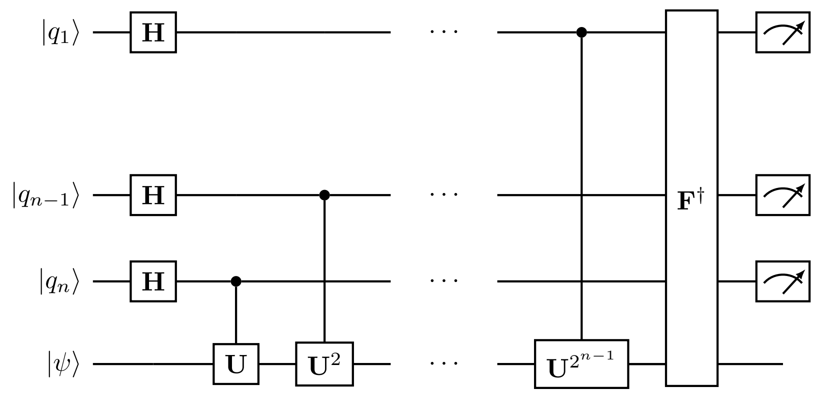

Figure 12: Quantum circuit for the Quantum Phase Estimation algorithm

The $\mathbf{H}$ gates are all applied in $O(1)$, and we know that $\mathbf{F}^\dagger$ is applied in time $O(n^2)$. However, the controlled gate $\mathbf{U}^{2^k}$ is implemented by concatenating $\mathbf{U}$ to itself 2^k times. As such, the total number of gates used for this step is $T_{\mathbf{U}(2^n-1)}$. Hence, the complexity of this circuit is $O\left(2^n\,n^2\,T_{\mathbf{U}}\right)$.

Finally, we will need to apply an arbitrary function on a quantum state. Since a quantum gate has to be reversible, there is a common trick to define such a gate.

Definition 2.5 (Gate of a real function). *Let $f : \{0\,;\,1\}^n \to \{0\,;\,1\}^m$ be an arbitrary function on bit strings. Then, the gate:*

$$\mathbf{U}_f : |x\rangle\,|y\rangle \mapsto |x\rangle\,|x \oplus y\rangle$$

is unitary. If f is efficiently computable, then so is $\mathbf{U}_f$. A way to perform this can be to decompose f using its Taylor expansion, or simply to translate the classical circuit for computing f into a quantum circuit.

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. These limitations are the difference between the theoretic algorithm and its actual implementation and as such, the reason of this paper. Three of them are particularly important.

First of all, only a limited set of gates can be implemented. For instance, Google's Xmon devices only supports the $\mathbf{Z}$ gate, the $\mathsf{C} - \mathbf{Z}$ gate and a more general version of the $\mathbf{X}$ gate. This is not of great importance, as long as we can define every gate in $\mathcal{E}$ from every allowed gate in time $O(1)$, to satisfy Assumption 2.1.

Then, only neighbours qubits can interact. Depending on how the qubits are ordered, like in a line or on a grid, their connectivity changes. To overcome this problem, we need to add SWAP gates whenever we desire to control a gate on several qubits. Since the number of SWAP gates is however in $O(n)$, it does not change the overall complexity of $\mathsf{C}^n - \mathbf{U}$. Hence, for simplicity and legibility, we decided not to include the SWAP gates in the circuit.

Finally, it is not possible to add gates dynamically on the quantum circuit. For instance, it is not possible to perform a measurement and add a gate dynamically as a function of the result. More precisely, there is no possibility to act on any qubit inside once it has begun to run. As a recall, the no-deleting theorem states:

Theorem 2.1 (No-deleting theorem). *There is no unitary* $\mathbf{U}$ *that maps an arbitrary state* $|\psi\rangle$ *to the state* $|0\rangle$.

The no-deleting theorem holds whenever the operations that we consider are unitary, which is the case whenever no measurement is performed. Here, we claim that this theorem still holds even whenever a measurement is performed.

One may think about performing a measurement applying $\mathbf{X}$ gates accordingly. Even if we were able to perform such a task, the coherence of the qubits wouldn't be guaranteed anymore. Hence, one would have to recreate the state on which it performed the measurement, and to apply measuring gates. Plus, it would imply that these measuring gates give two times in a row the same result, which has a probability of failure (otherwise there would be no point in performing a measurement in first place).

Other considerations are to be taken into account, like for instance the number of qubits available. Since there is no way to mitigate them, we discuss them in the conclusion.

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

Let us consider some platform with m different users and n different products. Each user has the possibility, once bought, to rate a product according to a binary marking scheme: either the user liked the product, or she did not. The goal of a recommendation system is to provide a user a recommendation from a list of already graded products. This list includes also products graded by other users.

Hence, we can represent this as a matrix with m rows and n columns, where each row represents a user and each column a product. Each coefficient of the matrix is either 0 or 1, indicating whether the user liked the product. This matrix is called the preference matrix, and we only have access to a portion of it (otherwise the problem would be trivial).

From this incomplete preference matrix, the recommendation system will compute an approximation of the true preference matrix and use it to make recommendation. Its goal is to discover somehow the relationships between users to discover how the fact that some user rated a product positively affects the probability that another user would do the same. For this reason, it is assumed that the preference matrix has a good approximation of rank k. In real-world terms, it means that it makes the assumption that there are k typical different users. Two users belonging to the same class will most likely rate positively the same products.

In classical approaches at that time, an SVD decomposition was performed on the reconstructed preference matrix. The, by keeping only the k largest singular values, a rank-k matrix was obtained and used for making predictions. This approach is polynomial in the parameters of the preference matrix m and n. The problem is that the reconstructed matrix has to be recomputed each time a user rates a new product. The problem is that the rank-k matrix had to be computed before getting a result from it. The principle of the algorithm of Kerenidis and Prakash is therefore to sample from this reconstructed matrix to get a recommendation once in polylogarithmic time.

3.2 A quantum algorithm as a recommendation system

The following describes the algorithm presented in [10]. This section intends to give a high-level overview of this algorithm. Its actual implementation is discussed in section 4. Let $\mathbf{P} \in \{0\,;\,1\}^{m \times n}$ a matrix that has a good rank-k approximation. That is, for a given approximation parameter ε:

$$\exists \hat{\mathbf{P}}, \|\mathbf{P} - \mathbf{P}_k\|_F \leqslant \varepsilon \|\mathbf{P}\|_F$$

where $\mathbf{P}_k$ is the matrix obtained by keeping only the k largest singular values in the SVD decomposition of $\mathbf{P}$.

Let $\eta \in \mathbf{R}_+^*$. Then we define the matrix $\hat{\mathbf{P}}$ as:

$$\forall (i,j) \in [\![1\,;\,m]\!] \times [\![1\,;\,n]\!], \hat{\mathbf{P}}_i,_j \overset{\text{def}}{=} \begin{cases} \mathbf{P}_i,_j \, \frac{\eta \, \|\mathbf{P}\|_F}{16\, n} & \text{with probability } \frac{16\, n}{\eta\, \|\mathbf{P}\|_F} \\ 0 & \text{with probability } 1 - \frac{16\, n}{(\eta\, \|\mathbf{P}\|_F)^2} \end{cases} .$$

Once $\hat{\mathbf{P}}$ has been built, the algorithm adds every new information it gets after every products it recommends, to it. Hence, the definition above is only used to initialize $\hat{\mathbf{P}}$.

Let j be the index of the row we want to sample. That is, we want a recommendation for user j. Then the algorithm operates in several steps, which we describe in the following.

3.2.1 Loading the row vector as a quantum state

The first thing to do is to load a quantum state that corresponds to $\hat{\mathbf{P}}_j$. That is, we want to create:

$$\left|\hat{\mathbf{P}}_j\right\rangle \overset{\text{def}}{=} \sum_{i=0}^{\lceil\log(n)\rceil-1} \hat{\mathbf{P}}_{j,i+1} \left|i\right\rangle .$$

The problem is hence to load in logarithmic time a quantum state corresponding to a vector stored in a classical data structure. This is done via the use of QRAM, as described in subsection 4.1.

We denote the SVD decomposition of $\hat{\mathbf{P}}$ as:

$$\hat{\mathbf{P}} = \sum_i \sigma_i \, \mathbf{u}_i \, \mathbf{v}_i^\dagger .$$

Even though we load $\left|\hat{\mathbf{P}}_i\right\rangle$ using its coefficients in the computational basis, we denote the quantum state in the left orthonormal basis in the SVD decomposition of $\hat{\mathbf{P}}$, that is:

$$\left|\hat{\mathbf{P}}_i\right\rangle \overset{\text{def}}{=} \sum_i \alpha_i \left|v_i\right\rangle .$$

3.2.2 Finding singular values

We now have the quantum state:

$$\sum_i \alpha_i \left|v_i\right\rangle$$

and we would like to find its singular values to get the state:

$$\sum_i \alpha_i \left|v_i\right\rangle \left|\sigma_i\right\rangle .$$

However, the Quantum Phase Estimation algorithm cannot be used right away, since $\hat{\mathbf{P}}$ is not unitary.

Hence, the goal is to define in logarithmic time a unitary $\mathbf{W}$ whose eigenvalues are the singular values of $\hat{\mathbf{P}}$. In order to define it, two matrices $\mathbf{M}_1 \in \mathbf{R}^{m\,n\times m}$ and $\mathbf{M}_2 \in \mathbf{R}^{m\,n\times n}$ are defined as follows:

$$\forall i \in [\![1\,;\,m]\!], \mathbf{M}_{1_i} \overset{\text{def}}{=} \mathbf{e}_i \otimes \frac{\hat{\mathbf{P}}_i}{\left\|\hat{\mathbf{P}}_i\right\|}$$

with $\mathbf{e}_i$ being the i^{th} vector of the computational basis of $\mathbf{R}^m$ and:

$$\forall i \in [\![1\,;\,n]\!], \mathbf{M}_{2_i} \overset{\text{def}}{=} \frac{\widetilde{\mathbf{P}}}{\left\|\hat{\mathbf{P}}\right\|_F} \otimes \mathrm{e}'_i$$

with $\mathbf{e}'_i$ being the i^{th} vector of the computational basis of $\mathbf{R}^n$ and $\widetilde{\mathbf{P}}$ being defined as:

$$\forall i \in [\![1\,;\,m]\!], \widetilde{\mathbf{P}}_i \overset{\text{def}}{=} \left\|\hat{\mathbf{P}}_i\right\| .$$

Then, by naming $\mathbf{U}$ the reflection on the vector space spanned by the columns of $\mathbf{M}_1$ and $\mathbf{V}$ the reflection on the vector space spanned by the columns of $\mathbf{M}_2$, then the unitary $\mathbf{W} = \mathbf{U}\,\mathbf{V}$ has as eigenvalues the singular values of $\hat{\mathbf{P}}_i$.

Hence, the task is to implement in logarithmic time $\mathbf{W}$ and to apply the Quantum Phase Estimation algorithm on it to get the state:

$$\sum_i \alpha_i \left|v_i\right\rangle \left|\sigma_i\right\rangle .$$

which is discussed in subsection 4.2.

3.2.3 Applying a gate according to a threshold

The previous part of the algorithm left us with the state:

$$\sum_i \alpha_i \left|v_i\right\rangle \left|\sigma_i\right\rangle .$$

In order to do a projection, we want to filter what singular values we want to keep. Hence, we define a threshold τ and want to get the state:

$$\sum_{i \in \mathcal{S}} \alpha_i \left|v_i\right\rangle \left|\sigma_i\right\rangle \left|0\right\rangle + \sum_{i \in \overline{\mathcal{S}}} \alpha_i \left|v_i\right\rangle \left|\sigma_i\right\rangle \left|1\right\rangle$$

where $\mathcal{S}$ is the set of indexes such that the corresponding eigenvector is associated with a eigenvalue $\sigma_i \geqslant \tau$. In the algorithm of Kerenidis and Prakash, τ is defined to be $\sigma\left(1 - \frac{\kappa}{2}\right)$.

Hence, the goal now is to define a unitary $\mathbf{T}_\sigma$ such that:

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

Hence, we need to be able to compare two quantum states. This is discussed in subsection 4.3.

After having applied T_σ, we are left with the state:

$$\sum_{i \in \mathcal{S}} \alpha_i \left|v_i\right\rangle \left|\sigma_i\right\rangle \left|0\right\rangle + \sum_{i \in \overline{\mathcal{S}}} \alpha_i \left|v_i\right\rangle \left|\sigma_i\right\rangle \left|1\right\rangle$$

with $\mathcal{S}$ being defined as above.

3.2.4 Concluding

Once the previous state obtained, we first need to uncompute the output of the Quantum Phase Estimation algorithm, so that the final state is not entangled. Hence, we are left with:

$$\sum_{i \in \mathcal{S}} \alpha_i \left|v_i\right\rangle \left|0\right\rangle + \sum_{i \in \overline{\mathcal{S}}} \alpha_i \left|v_i\right\rangle \left|1\right\rangle .$$

We then measure the second register in the computational basis. If the outcome is $\left|1\right\rangle$, then we redo everything. If it is $\left|0\right\rangle$, then we measure the first register in the computational basis to get a product for user j.

4 Designing a quantum circuit for a recommendation system

We now design a quantum circuit to perform the algorithm of Kerenidis and Prakash. Let $\mathbf{P}$ be the black box preference matrix. As a recall, we define $\hat{\mathbf{P}}$ as:

$$\forall (i, j) \in [\![1 \,;\, m]\!] \times [\![1 \,;\, n]\!], \hat{\mathbf{P}}_i, _j \overset{\text{def}}{=} \begin{cases} \mathbf{P}_i, _j \, \frac{\eta \left\|\mathbf{P}\right\|_F}{16\,n} & \text{with probability } \frac{16\,n}{\eta \left\|\mathbf{P}\right\|_F} \\ 0 & \text{with probability } 1 - \frac{16\,n}{(\eta \left\|\mathbf{P}\right\|_F)^2} \end{cases} .$$

$\hat{\mathbf{P}}$ is assumed to be known at the beginning of the algorithm. It is also assumed that, by denoting $\mathbf{P}_k$ the matrix obtained by keeping only the k largest singular values in the SVD decomposition of $\mathbf{P}$:

$$\left\|\mathbf{P} - \mathbf{P}_k\right\|_F \leqslant \varepsilon \left\|\mathbf{P}\right\|_F$$

for a small rank k (generally lower than 100) and an approximation parameter ε, both known beforehand. We also consider an index j to get a recommendation from.

As explained in subsubsection 3.2.1, the very first step is to load in logarithmic time the state:

$$\left|\hat{\mathbf{P}}_j\right\rangle \overset{\text{def}}{=} \sum_{i=0}^{\lceil \log(n) \rceil - 1} \hat{\mathbf{P}}_{j,i+1} \left|i\right\rangle .$$

4.1 Loading from QRAM

In order to load a quantum state from a real vector, Kerenidis and Prakash make use of Quantum RAM, also denoted QRAM. The principle is to have a classical data structure with quantum access, that is, a data structure from which we can get information by querying it with a quantum state, potentially in superposition.

While some general-purpose QRAM circuit has been stud, for instance in [12], we will describe a design specially designed for state creation, described by Prakash in [16].

4.1.1 Designing Quantum RAM

Let $\mathbf{x} \in \mathbf{R}^p$ be a real vector. The QRAM structure is a binary tree with $2^{\lceil \log_2(p) \rceil}$ leaves with the following properties:

1. The leaf number i stores the value x_i^2, along with the sign of x_i. Potential additional leaves stores the value 0.
2. Each node stores the sum of the values of its two children.

Hence, the root stores $\sum_i x_i^2 = \|\mathbf{x}\|^2$. For instance, with:

$$\mathbf{x} = \begin{pmatrix} 0.2 \\ -0.15 \\ 0.1 \\ -0.1 \\ 0 \\ 0.3 \end{pmatrix}$$

the QRAM tree would be:

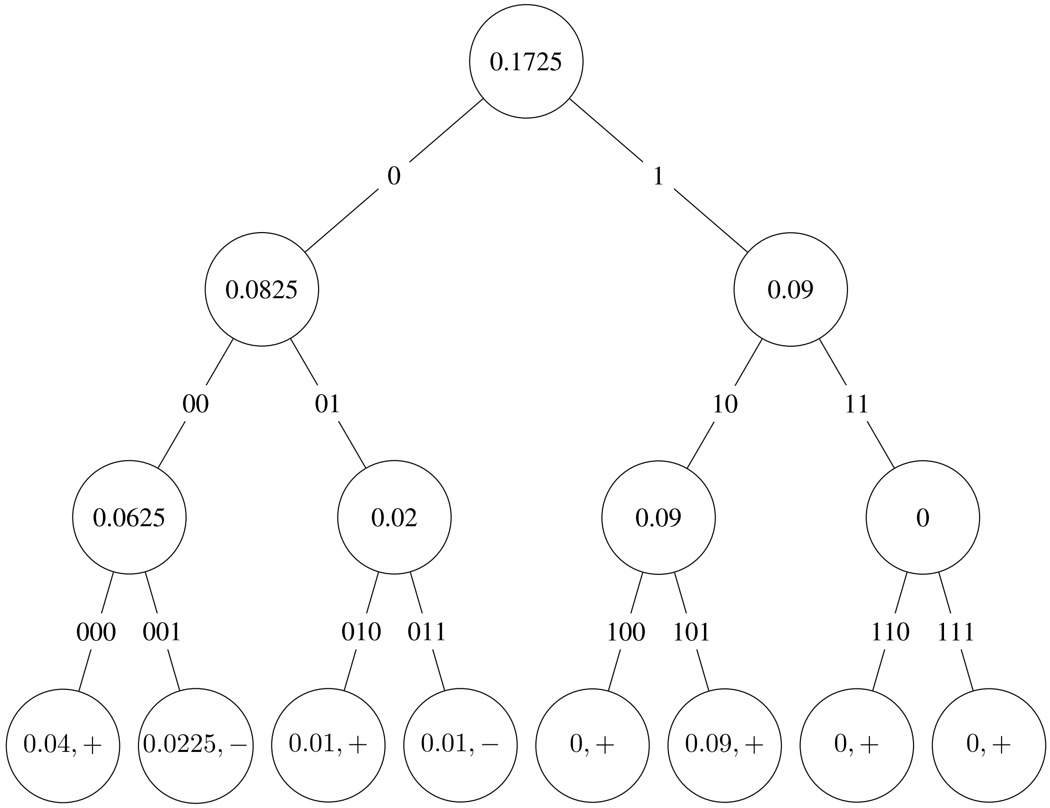

Figure 13: An example of a QRAM tree

We now define the gate $\mathbf{L}_\mathrm{x}$ with the following property:

$$\mathbf{L}_\mathrm{x} \left|0\right\rangle^{\otimes \lceil \log_2(p) \rceil} = \left|x\right\rangle \ .$$

Hence, our goal is to decompose $\mathbf{L}_{\mathrm{x}}$ into elementary gates to ensure that it can be implemented in logarithmic time. In order to do so, we use the algorithm presented in [6].

Algorithm 1: Loading a vector from QRAM [6]

```
Input  : A vector $\mathbf{x} \in \mathbf{R}^p$ stored in QRAM, a quantum state $\left|q_1\, q_2\, \cdots\, q_{\lceil \log_2(p) \rceil}\right\rangle = |0\rangle^{\otimes \lceil \log_2(p) \rceil}$.
Output: The quantum state $\left|q_1\, q_2\, \cdots\, q_{\lceil \log_2(p) \rceil}\right\rangle = |x\rangle$.
function processNode(Node $u$):
    Let $u_l$ be the left child of $u$, $u_r$ be its right child and let $k$ be the level of the tree at which $u$ is. For
      instance, the level of the root is 1, while the level of a leaf is $\lceil \log_2(p) \rceil$. Note that we can access the
      values of these nodes in time $O(k)$ thanks to the binary tree structure. Compute $\theta$ being defined as:
          $$\theta \overset{\text{def}}{=} \arccos\left(\sqrt{\frac{\text{value}(u_l)}{\text{value}(u)}}\right).$$
    Apply on $|q_k\rangle$ the gate $\mathbf{R}_y(2\,\theta)$ controlled by $|q_1\, \cdots\, q_{k-1}\rangle$ being equal to the binary expansion of $u$.
    if $u_l$ is a leaf then
        if $\operatorname{sgn}(u_l) = +$ and $\operatorname{sgn}(u_r) = -$ then
            Apply $\mathbf{R}_z(\pi)$ controlled on $|q_1\, \cdots\, q_{k-1}\rangle$ as previously on $|q_k\rangle$.
        else if $\operatorname{sgn}(u_l) = -$ and $\operatorname{sgn}(u_r) = +$ then
            Apply $\mathbf{R}_y(2\,\pi)\,\mathbf{R}_z(\pi)$ controlled on $|q_1\, \cdots\, q_{k-1}\rangle$ as previously on $|q_k\rangle$.
        else if $\operatorname{sgn}(u_l) = -$ and $\operatorname{sgn}(u_r) = -$ then
            Apply $\mathbf{R}_y(2\,\pi)$ controlled on $|q_1\, \cdots\, q_{k-1}\rangle$ as previously on $|q_k\rangle$.
    else
        processNode($u_l$)
        processNode($u_r$)
    end

Apply processNode on the root of the QRAM tree.
```

The correctness of this algorithm is proved in [6]. Taking back our previous example, this algorithm will generate the following circuit:

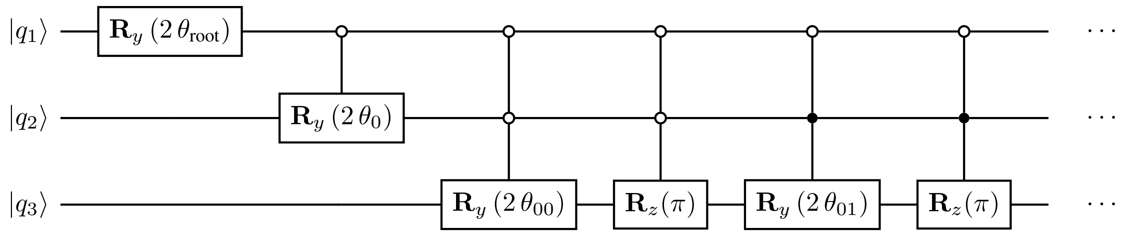
$|q_1\rangle$
$|q_2\rangle$
$|q_3\rangle$
$\mathbf{R}_y\,(2\,\theta_{\text{root}})$
$\mathbf{R}_y\,(2\,\theta_0)$
$\mathbf{R}_y\,(2\,\theta_{00})$
$\mathbf{R}_z(\pi)$
$\mathbf{R}_y\,(2\,\theta_{01})$
$\mathbf{R}_z(\pi)$
$\cdots$
$\cdots$
$\cdots$

Figure 14: An example of a circuit generated by the loading from QRAM algorithm

Several things are to be noted with this implementation:

- The algorithm can be slightly improved by not applying gates for nodes whose value is equal to their left children's.
- The computation of θ_{11} implies a division by zero. Since this can only happen when a leaf and its parent share the same value, the previous point would apply.
- Qubit $|q_k\rangle$ undergoes 2^k rotations. More are actually applied on the last one, with a limit of $3 \times 2^{\lceil \log_2(p) \rceil}$. Hence, the total number of consecutive gates applied with this algorithm is:

$$3 \times 2^{\lceil \log_2(p) \rceil} + \sum_{k=1}^{\lceil \log_2(p) \rceil - 1} 2^k = 2^{2 + \lceil \log_2(p) \rceil} - 1$$

which makes this algorithm linear in p, while we want to have a logarithmic complexity.

4.1.2 Parallel execution of rotations

The thing is that we can parallelize the rotations applied. Since our goal is to get data from the QRAM from a superposed state, let us define a function f such that:

$$f : \left|\begin{array}{ccc}\mathbf{R}^p & \to & \mathbf{R}^t \\ \mathbf{x} & \mapsto & \theta_{\mathbf{x}}\end{array}\right. .$$

The goal of f is to take a vector from the computational basis of $\mathbf{R}^p$ and to map it to the corresponding rotation angle to be performed at a certain level. Hence, the associated matrix of f is a permutation matrix.

Using $\mathbf{U}_f$, we can now have access to $|\theta_x\rangle$. As a recall, since we assume f is efficiently computable classically, so is $\mathbf{U}_f$. We now want to perform a rotation around the y-axis of angle $2\,\theta_x$.

In order to give the intuition on the way to perform this, let us consider the following circuit:

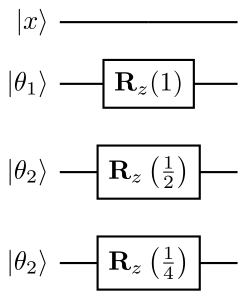

Figure 15: A rotation around the z-axis using a quantum state

The final quantum state is given by:

$$|x\rangle \otimes \mathrm{e}^{\mathrm{i}\,\theta_1}\,|\theta_1\rangle \otimes \mathrm{e}^{\mathrm{i}\,\theta_2\,2^{-1}}\,|\theta_2\rangle \otimes \mathrm{e}^{\mathrm{i}\,\theta_3\,2^{-2}}\,|\theta_3\rangle = \mathrm{e}^{\mathrm{i}\,\theta}\,|x\rangle\,|\theta\rangle\,.$$

Hence, we are able using this to perform a rotation around the z-axis from a quantum state. Quite similarly to what happened in the previous circuit, we will now apply phase gates on the $|\theta_k\rangle$ qubits, but this time controlled on the target qubit. The reason why we perform this will appear in the state description. Let us consider the following circuit, where we only took two qubits for approximating θ for legibility:

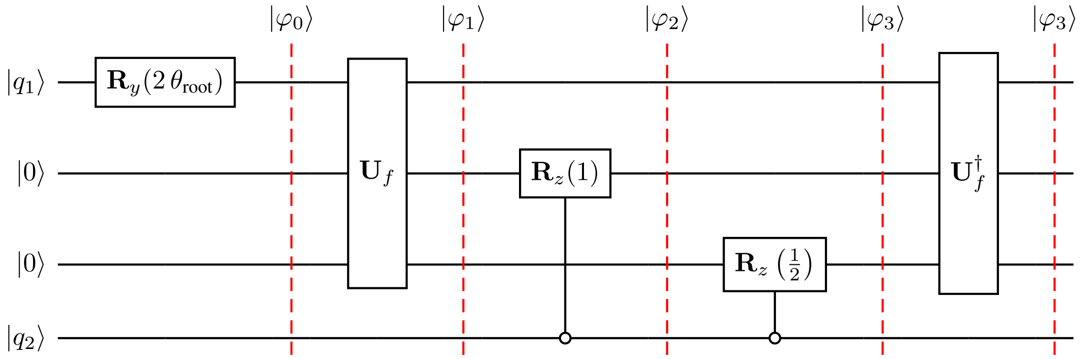

Figure 16: Rotating around the z-axis using parallel computation

In order to see that this circuit almost performs the operation we want, let us describe the quantum state at every step. First of all, we have:

$$|\varphi_0\rangle = (\alpha\,|000\rangle + \beta\,|100\rangle)\,(\gamma\,|0\rangle + \delta\,|1\rangle)\,.$$

We then apply the $\mathbf{U}_f$ gate on the first three qubits. As a recall, since $\mathbf{U}_f$ implements f, this will leave the first qubit unchanged, while it will modify the two others qubits to the actual value of θ_x. Hence, we got the state:

$$|\varphi_1\rangle = (\alpha\,|0\theta_{0,0}\theta_{0,1}\rangle + \beta\,|1\theta 1, 0\theta 1, 1\rangle)\,(\gamma\,|0\rangle + \delta\,|1\rangle)\,.$$

Note that we denoted $\theta_{x,n}$ as the $n+1^{\text{th}}$ qubit of θ_x. It will be easier for the following to develop this expression:

$$|\varphi_1\rangle = \alpha\,\gamma\;|0\theta_{0,0}\theta_{0,1}0\rangle + \alpha\,\delta\;|0\theta_{0,0}\theta_{0,1}1\rangle + \beta\,\gamma\;|1\theta_{1,0}\theta_{1,1}0\rangle + \beta\,\delta\;|1\theta_{1,0}\theta_{1,1}1\rangle\;.$$

We now apply the $\mathbf{R}_z(1)$ gate controlled on the last qubit:

$$|\varphi_2\rangle = \alpha\,\gamma\;|0\theta_{0,0}\theta_{0,1}0\rangle + \alpha\,\delta\,\mathrm{e}^{\mathrm{i}\,\theta_{0,0}}\;|0\theta_{0,0}\theta_{0,1}1\rangle + \beta\,\gamma\;|1\theta_{1,0}\theta_{1,1}0\rangle + \beta\,\delta\,\mathrm{e}^{\mathrm{i}\,\theta_{1,0}}\;|1\theta_{1,0}\theta_{1,1}1\rangle\;.$$

Similarly, we apply the $\mathbf{R}_z\left(\frac{1}{2}\right)$ gate on the third qubit controlled on the last qubit:

$$|\varphi_3\rangle = \alpha\,\gamma\;|0\theta_{0,0}\theta_{0,1}0\rangle + \alpha\,\delta\,\mathrm{e}^{\mathrm{i}\,\theta_{0}}\;|0\theta_{0,0}\theta_{0,1}1\rangle + \beta\,\gamma\;|1\theta_{1,0}\theta_{1,1}0\rangle + \beta\,\delta\,\mathrm{e}^{\mathrm{i}\,\theta_{1}}\;|1\theta_{1,0}\theta_{1,1}1\rangle\;.$$

Finally, we uncompute the first three qubits to be left with:

$$|\varphi_4\rangle = \alpha\,\gamma\;|000\rangle + \alpha\,\delta\,\mathrm{e}^{\mathrm{i}\,\theta_{0}}\;|0001\rangle + \beta\,\gamma\;|1000\rangle + \beta\,\delta\,\mathrm{e}^{\mathrm{i}\,\theta_{1}}\;|1001\rangle\;.$$

Which we can rewrite as:

$$|\varphi_4\rangle = \left(\alpha\;|000\rangle + \beta\;|001\rangle\right)\gamma\;|0\rangle + \left(\alpha\,\mathrm{e}^{\mathrm{i}\,\theta_{0}}\;|000\rangle + \beta\,\mathrm{e}^{\mathrm{i}\,\theta_{1}}\;|100\rangle\right)\;\delta\;|1\rangle$$

which is exactly what we wanted to get. Indeed, we left the three first qubits unchanged, and applied rotations around the z-axis on the last qubit according to the state $|x\rangle$. The only thing left to do now is to convert rotations around the z-axis to rotations around the y-axis. In order to do this, we want to apply basis change matrices before and after having applied rotations around the z-axis. Let us show that the gate $\frac{1}{\sqrt{2}}\left(\mathrm{e}^{\mathrm{i}\,\frac{\pi}{2}}\,\mathbf{R}_y(\pi) + \mathbf{R}_z(\pi)\right)$ is the gate we are looking for. Indeed, we have:

$$\begin{aligned}
\frac{1}{\sqrt{2}}\begin{pmatrix}1 & -\mathrm{i}\\ \mathrm{i} & -1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & \mathrm{e}^{\mathrm{i}\,\varphi}\end{pmatrix}\left(\frac{1}{\sqrt{2}}\begin{pmatrix}1 & -\mathrm{i}\\ \mathrm{i} & -1\end{pmatrix}\right)^{-1} &= \frac{1}{2}\begin{pmatrix}1 & -\mathrm{i}\\ \mathrm{i} & -1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & \mathrm{e}^{\mathrm{i}\,\varphi}\end{pmatrix}\begin{pmatrix}1 & -\mathrm{i}\\ \mathrm{i} & -1\end{pmatrix}\\
&= \frac{1}{2}\begin{pmatrix}1 & -\mathrm{i}\,\mathrm{e}^{\mathrm{i}\,\varphi}\\ \mathrm{i} & -\mathrm{e}^{\mathrm{i}\,\varphi}\end{pmatrix}\begin{pmatrix}1 & -\mathrm{i}\\ \mathrm{i} & -1\end{pmatrix}\\
&= \frac{1}{2}\begin{pmatrix}1+\mathrm{e}^{\mathrm{i}\,\varphi} & -\mathrm{i}\left(1-\mathrm{e}^{\mathrm{i}\,\varphi}\right)\\ \mathrm{i}\left(1-\mathrm{e}^{\mathrm{i}\,\varphi}\right) & 1+\mathrm{e}^{\mathrm{i}\,\varphi}\end{pmatrix}\\
&= \frac{1}{2}\begin{pmatrix}2\,\cos\left(\frac{\varphi}{2}\right)\,\mathrm{e}^{\mathrm{i}\,\frac{\varphi}{2}} & -2\,\sin\left(\frac{\varphi}{2}\right)\,\mathrm{e}^{\mathrm{i}\,\frac{\varphi}{2}}\\ 2\,\sin\left(\frac{\varphi}{2}\right)\,\mathrm{e}^{\mathrm{i}\,\frac{\varphi}{2}} & 2\,\cos\left(\frac{\varphi}{2}\right)\,\mathrm{e}^{\mathrm{i}\,\frac{\varphi}{2}}\end{pmatrix}\\
&= \mathrm{e}^{\mathrm{i}\,\frac{\varphi}{2}}\,\mathbf{R}_y(\varphi)
\end{aligned}$$

Hence, up to a global phase which is irrelevant, we can using these basis change matrices use rotations around the z-axis to perform rotations around the y-axis. Hence, the final circuit to perform such a task is as follows:

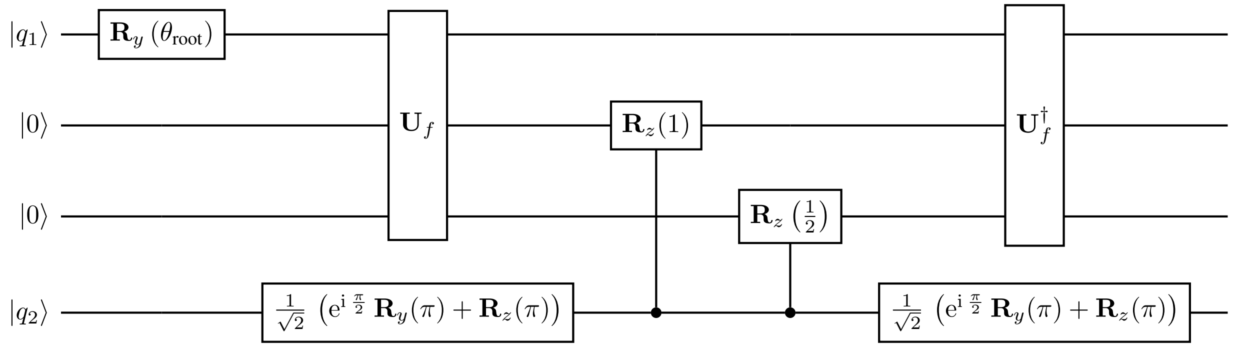
$|q_1\rangle$
$\mathbf{R}_y\left(\theta_{\text{root}}\right)$
$|0\rangle$
$|0\rangle$
$|q_2\rangle$
$\mathbf{U}_f$
$\mathbf{R}_z(1)$
$\mathbf{R}_z\left(\frac{1}{2}\right)$
$\mathbf{U}_f^\dagger$
$\frac{1}{\sqrt{2}}\left(\mathrm{e}^{\mathrm{i}\,\frac{\pi}{2}}\,\mathbf{R}_y(\pi) + \mathbf{R}_z(\pi)\right)$
$\frac{1}{\sqrt{2}}\left(\mathrm{e}^{\mathrm{i}\,\frac{\pi}{2}}\,\mathbf{R}_y(\pi) + \mathbf{R}_z(\pi)\right)$

Figure 17: Circuit to perform rotations around the y-axis efficiently

This circuit can be generalised for an arbitrary number of qubits for approximating θ and for any level of the tree k.

We now have defined the loading gate $\mathbf{L}_{\mathrm{x}}$, which we can apply to get the state $|x\rangle = \left|\hat{\mathrm{P}}_j\right\rangle$.

4.2 Applying the Quantum Phase Estimation algorithm

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

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

Since $\widetilde{\mathbf{P}}$ is easily loadable via QRAM, we only have to apply the loading gate $\mathbf{L}_{\widetilde{\mathbf{P}}}$ to the first register to get the desired state. As a recall, $\widetilde{\mathbf{P}}$ is defined as:

$$\forall i \in [\![1\,;\,m]\!], \widetilde{\mathbf{P}}_i = \|\hat{\mathbf{P}}_i\| .$$

Let us assume that $\hat{\mathbf{P}}$ is stored in QRAM. Storing a matrix in QRAM is done similarly to storing a vector in QRAM: a binary tree is built just like the row vectors of $\hat{\mathbf{P}}$ were real numbers, then each leaf is the root of a QRAM tree that stores the corresponding vector.

Two cases are now possible: either the structure we possess has a quantum access, or not. Since we did not manage to code such a structure, our code does not match the original algorithm's complexity. Indeed, the goal is now to define a unitary $\widetilde{\mathbf{U}}$ such that:

$$\widetilde{\mathbf{U}}\;|i\rangle\;|0\rangle^{\otimes\lceil\log_2(n)\rceil} = |i\rangle\;\left|\hat{\mathbf{P}}_i\right\rangle .$$

In the case where the data structure lacks a quantum access, the way to implement it is to use m different loading gates, which leads to a time complexity for this gate of $O(m\;\log(n))$. If the data structure has a quantum access, then by definition it will implement $\widetilde{\mathbf{U}}$.

The other unitary that we need, called $\widetilde{\mathbf{V}}$ in the original algorithm, is simply the $\mathbf{L}_{\widetilde{\mathbf{P}}}$ gate we used before.

Assuming that the data structure we are using has a quantum access, both $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ can be implemented in time $\mathrm{polylog}(m\,n)$.

The Quantum Recommendation System algorithm uses two unitary matrices $\mathbf{U}$ and $\mathbf{V}$ to perform the Quantum Phase Estimation algorithm. Both these gates are reflections on certain vector fields and both can be defined using $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$:

$$\mathbf{U} = \widetilde{\mathbf{U}}\,\mathbf{R}_1\,\left(\widetilde{\mathbf{U}}\right)^{-1}$$

and:

$$\mathbf{U} = \widetilde{\mathbf{V}}\,\mathbf{R}_0\,\left(\widetilde{\mathbf{V}}\right)^{-1}$$

where $\mathbf{R}_1$ is the reflection on the vector field $|y\rangle\;|0\rangle^{\otimes\lceil\log(n)\rceil}$ for $|y\rangle\in\mathbf{R}^m$ and where $\mathbf{R}_0$ is the reflection on the vector field $|0\rangle^{\otimes\lceil\log(m)\rceil}\;|x\rangle$ for $|x\rangle\in\mathbf{R}^n$. It is easy to implement both $\mathbf{R}_0$ and $\mathbf{R}_1$ since they are both diagonal matrices with -1 coefficients everywhere except on the vector they perform a reflection on. For instance, we have:

$$\mathbf{R}_0 = \begin{pmatrix}\mathbf{I}_n & \mathbf{0}\\ \mathbf{0} & -\mathbf{I}_{m\,n-n}\end{pmatrix} .$$

$-\mathbf{I}_{m\,n-n}$ can be implemented using a tensor product of $O(\log(m\,n))$ $\mathbf{R}_y(2\,\pi)$ gates. Hence, we can implement $\mathbf{R}_0$ as a single gate $-\mathbf{I}_{m\,n-n}$ that acts on the second register and controlled by the first register being in state $|0\rangle^{\otimes\lceil\log(n)\rceil}$. Each of these controlled gates will be implemented in time $O(\log(n))$ because of the numerous control qubits. As a consequence, $\mathbf{R}_0$ can be implemented in time $O(\mathrm{polylog}(m\,n))$, which matches the complexity of the original algorithm. A similar reasoning holds for $\mathbf{R}_1$.

Hence, we are able to implement $\mathbf{U}\,\mathbf{V}$ in polylogarithmic time. We can finally apply the Quantum Phase Estimation algorithm using this unitary. However, it is not possible to do it as originally claimed by Kerenidis and Prakash. Indeed, as mentioned in subsection 2.2, it is not possible to measure a state and then to apply $\mathbf{X}$ gates accordingly to uncompute it. Hence, it is not possible to increase the probability of success of the Quantum Phase Estimation algorithm using this way.

However, it is stated in [14], that one can increase the number of qubits used in order to increase the probability of success of the algorithm. Indeed, if we want the algorithm to output a n-bit approximation of the phase of an eigenvalue, one can use instead $n+p$ qubits. Nielsen and Chuang show then that the probability of success, that is the probability to obtain the desired state in the first n qubits, is at least $1-\frac{1}{2\,(2^p-2)}$.

Our main objective is to match the original complexity of the algorithm. Since we cannot repeat the QPE $\log(n)$ times as wanted by the original algorithm, we are allowed to add p qubits, with p verifying:

$$2^p = \log(n)\,.$$

Indeed, adding p qubits multiply by $2^p\,p^2$ the time taken by the algorithm. Neglecting p^2 in comparison with 2^p, and because we consider complexities in $\operatorname{polylog}(m\,n)$ anyway, we find that we can use an additional number of $p=\lceil \log_2(\log(n))\rceil$ qubits. The probability of success is now at least $1-\frac{1}{2\,(\log(n)-2)}$, which is lower than the original probability of success given in [10].

Still, it is necessary to mention that this cannot be implemented currently. Indeed, it is later needed in the algorithm to uncompute the output of the Quantum Phase Estimation algorithm, so that the state is not entangled. Yet, as long as we perform a measurement, which is necessary to get the approximation, we cannot uncompute this output. It is not possible either to run the Quantum Phase Estimation once to get the estimation, since the output of the Quantum Phase estimation is entangled with the rest of the qubits. In the following, we will assume that this problem has been tackled.

The QPE is used to get an estimation of the singular values of $\hat{\mathbf{P}}$. However, what the QPE outputs is an estimation of the phase of the corresponding eigenvalue. In order not to have to create the state $|\sigma_i\rangle$ obtained after the measurement (which would be entangled anyway), we keep this output from the Phase Estimation and work with it, rather than with $|\sigma_i\rangle$. Hence, the state after having applied the QPE and having uncomputed $\mathbf{Q}$ is:

$$\sum_i \alpha_i\ |v_i\rangle\ |a_i\rangle\ .$$

4.3 Separating states according to a threshold

4.3.1 Defining the condition

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 σ_i of $\mathbf{A}$. By definition:

$$\exists \theta_i \in [-\pi\,;\,\pi[, \overline{\sigma_i} = \cos\left(\frac{\theta_i}{2}\right)\ \|\mathbf{A}\|_F\ .$$

What we want to do now is to perform a mapping between a_i and θ_i, since we only have access to a_i. As a recall, a_i is the output of the Quantum Phase Estimation algorithm. Hence, the phase of the eigenvalue is $2\,\pi\,a_i$. The problem is that θ_i is defined on $[-\pi\,;\,\pi[$. Hence, θ_i and $2\,\pi\,a_i$ corresponds to the same angle for $a_i \in [0\,;\,\frac{1}{2}[$. For $a_i \geqslant \frac{1}{2}$, θ_i grows linearly again with a_i, and we have:

$$\exists (m,p) \in \mathbf{R}^2, \begin{cases} m \times \frac{1}{2} + p &=& -\pi \\ \\ m \times 1 + p &=& 0 \end{cases}$$

Hence, we have:

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

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:

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

4.3.2 Comparing a quantum state and a real number bitwise

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 $b_2 = 1 - b_1$. Let b represent either b_1 or b_2.

What we want to do is to determine whether $b < a_i$ by looking at their binary expansion. As a recall, the following holds:

$$\forall N \in \mathbf{N}, 2^{-}N = \sum_{k=N+1}^{+\infty} 2^{-k} .$$

In particular:

$$\forall N \in \mathbf{N}, \forall M \geqslant N, \forall \lambda \in \{0\,;\,1\}^{M-N}, 2^{-N} > \sum_{k=N+1}^{M} \lambda_k\, 2^{-k} .$$

We also know that we can write a_i using its binary expansion:

$$a_i = \sum_{k=1}^{\lceil \log(n) \rceil} a_{i,k} 2^{-k}$$

and the same holds for b:

$$b = \sum_{k=1}^{\lceil \log(n) \rceil} b_k 2^{-k} .$$

Using this, we know that the first bit j at which $b_j \neq a_{i,j}$ determines whether $a_i \geqslant b$. If $a_{i,j} = 1$, then $a_i > b$. If $b_j = 1$, then $b > a_i$.

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 $b_{1,2} = 1$. Hence, 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 [15]. 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.

4.3.3 Detecting the inequality

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

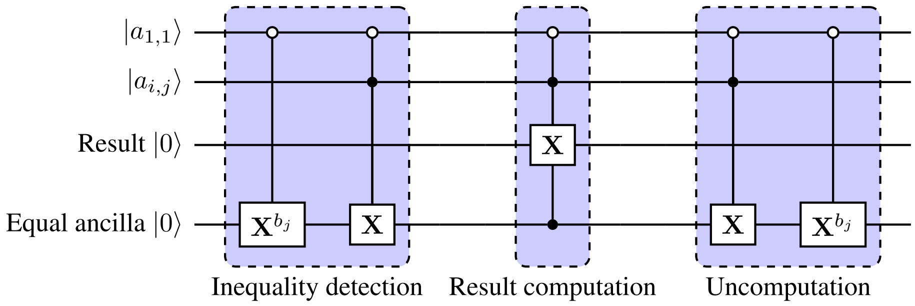

Figure 18: First idea of a partial implementation of $\mathbf{T}_\sigma$ using one ancilla qubit

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.

4.3.4 Maintaining the result qubit in its state once the result has been found

Hence, we need another ancilla qubit to know whether the circuit already has determined that $a_i > b$. By controlling the operations on the result qubit by this new ancilla qubit, we avoid applying again a gate on it if the result is already known, as shown on the following:

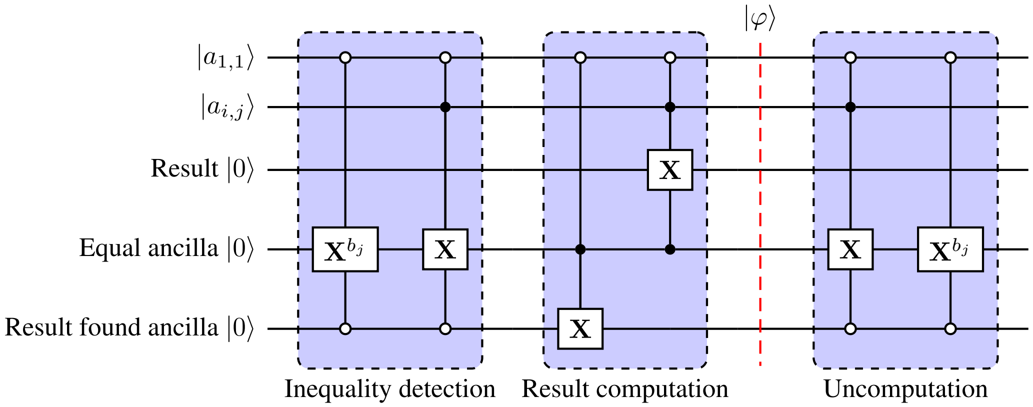

Figure 19: Second idea of a partial implementation of $\mathbf{T}_\sigma$ using two ancilla qubits

The idea is to test for the equality between the bits only when the circuit 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 $|1\rangle$.

4.3.5 Uncomputing the equal ancilla

Hence, we need to add another ancilla qubit whose goal is to store whether the result is set at the beginning of a step. By controlling the uncomputation gates on this qubit, we ensure to apply them only when it is necessary. This leads to another version of the partial implementation of $\mathbf{T}_\sigma$:

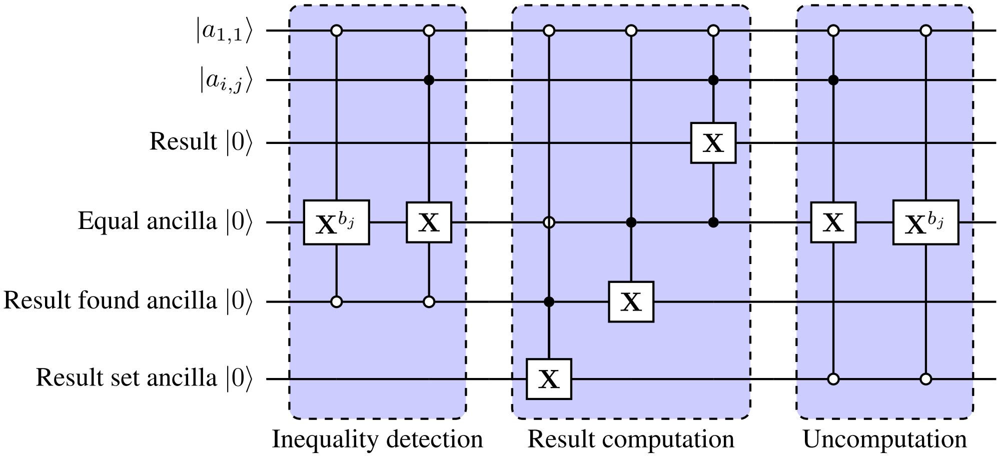

Figure 20: Third idea of a partial implementation of $\mathbf{T}_\sigma$ using three ancilla qubits

But there is still one problem: at the $j+1$ step, the ancilla will change its value again, eventually changing other registers.

4.3.6 Uncompute the equal qubit using more ancillas

To solve this problem, we can define an ancilla for every step and let them be thereafter. Then, the only need to take into account is to prove that the result qubit is not entangled with these ancillas, to ensure to have a correct measurement. Hence, we need an ancilla for every step, which can by symbolised with the following circuit:

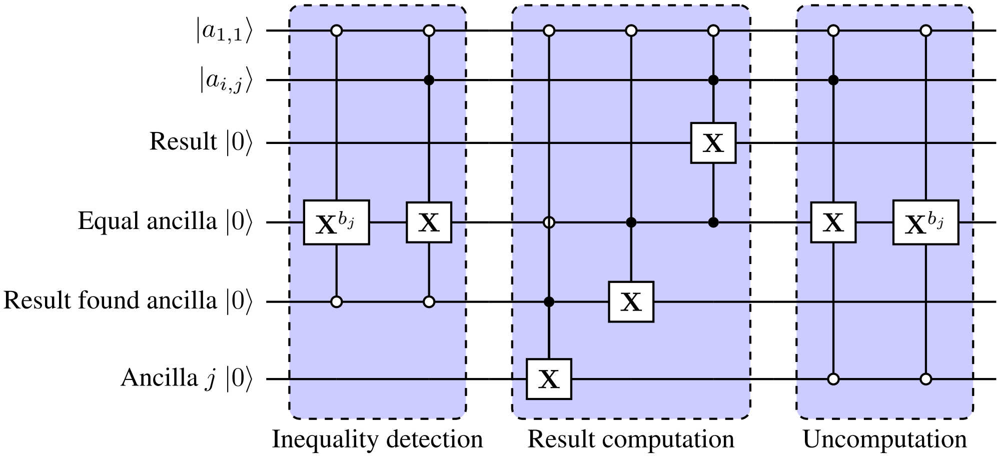

Figure 21: Final version of the implementation of $\mathbf{T}_\sigma$ using $\lceil\log(n)\rceil + 1$ ancilla qubits

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

In this paper, we tried to highlight the differences between a theoretical implementation of a quantum algorithm and its real-world implementation. We showed that even given some compromises, like cutting down the probability of success, some operations are for now impossible to perform.

In particular, we discussed two useful concepts used in quantum subroutines: QRAM and Quantum Phase Estimation. While the former shows its true potential when implemented with a quantum access, the latter is, at our knowledge, not conveniently usable as a quantum subroutine because of the measurement it performs.

We believe it is feasible to load data in the QRAM using circuits as mentioned in [12] but have chosen to let it for a future work. We also believe that it is possible to use the QPE algorithm without either performing the measurement or having to uncompute it, which would solve the problem.

Still, it is important to keep in mind that the algorithm described has a classical equivalent described by Tang in [18]. As such, it is very likely that this classical algorithm will be used in lieu of the quantum one, for performance reasons.

Plus, we did not discuss the actual number of qubits needed to run this algorithm, nor the performance of the gates we consider. Indeed, quantum computers are known to be noisy, and quantum error correction is now both an art and a science. In order to measure the performance of a quantum computer, Cross *et al.* recently introduced the notion of Quantum Volume in [5]. This metric given, one can use it to objectively evaluate a quantum computer. As such, it is useful to track whenever the quantum supremacy will be observed on several other algorithms than the one used by Google in 2019 [1].

6 Acknowledgments

I would like to thank Mario BERTA, both for having me introduced to the Quantum Computing field and for having been such a great supervisor in this Individual Study Option.

References

[1] Frank Arute *et al.* “Quantum supremacy using a programmable superconducting processor”. In: *Nature* 574.7779 (2019), pp. 505–510. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5. URL: https://doi.org/10.1038/s41586-019-1666-5.

[2] Mario Berta. *Quantum Computing*. 2019.

[3] Arthur Braida. “Learning Quantum States and Implementation with Cirq”. MA thesis. Imperial College London, 2019.

[4] 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.

[5] Andrew W. Cross *et al.* “Validating quantum computers using randomized model circuits”. In: *Physical Review A* 100.3 (2019). ISSN: 2469-9934. DOI: 10.1103/physreva.100.032328. URL: http://dx.doi.org/10.1103/PhysRevA.100.032328.

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

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

[8] 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.

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

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

[11] G. Linden, B. Smith, and J. York. "Amazon.com recommendations: item-to-item collaborative filtering". In: *IEEE Internet Computing* 7.1 (2003), pp. 76–80.

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

[13] Hamed Mohammadbagherpoor *et al. An Improved Implementation Approach for Quantum Phase Estimation on Quantum Computers*. 2019. arXiv: 1910.11696 [quant-ph].

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

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

[16] 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.

[17] 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.

[18] Ewin Tang. "A quantum-inspired classical algorithm for recommendation systems". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing - STOC 2019* (2019). DOI: 10.1145/3313276.3316310. URL: http://dx.doi.org/10.1145/3313276.3316310.