INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

The Quantum Fourier Transform for Earth Observation

Aaron Söhnen

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

The Quantum Fourier Transform for Earth Observation

Aaron Söhnen

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Tobias Guggemos
Abgabetermin:	6. Oktober 2022

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 06. Oktober 2022

. . . . . .

(Unterschrift des Kandidaten)

Abstract

The Fourier transform algorithm is ubiquitously used and essential for applications like digital signal processing, as well as audio and video compression. Its quantum analogue, the quantum Fourier transform (QFT), first became famous as a part of Shor's Algorithm for factoring numbers in the nineties. Since then, many influential algorithms have been built on top of the QFT and the associated quantum phase estimation.

One of these is a quantum algorithm for solving systems of linear equations. It was developed by Harrow, Hassidim and Loyd in 2009 [HHL09] and is hence often called HHL for short. Under certain conditions, HHL is able to find the solution x of the system $Ax = b$ with a complexity of only $\mathcal{O}(\log n)$, where n is the number of variables. This stands in contrast to classical techniques which achieve a runtime of $\mathcal{O}(n^2)$ at best.

In this work, we develop the HHL algorithm and the components upon which it is built, including the quantum Fourier transform. While the theoretical speedup of HHL is exponential, the conditions which must be met can pose difficult problems for actual implementations. We give an in-depth overview of these conditions and the research aimed to improve them.

At DLR[1] large amounts of earth observation data from satellites needs to be processed. This presents a possible future use case for quantum algorithms. We evaluate the applicability of HHL for space-borne synthetic aperture radar tomography, which is a technique to reconstruct three-dimensional surface features from radar data.

[1] German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt)

Contents

1 Introduction and Motivation

After the discovery of quantum mechanical phenomena in the early 20th century, the studied effects suggested that the construction of a quantum computer could be possible. Such devices would utilize quantum mechanics to perform computations that are impossible on standard computers. Now, in the 21st century, the first functional versions of such devices are being built by multiple organizations and there is an active research community developing algorithms for applications.

Since its discovery in 1994, Shor's algorithm [Sho94] was for a long time the only widespread algorithm associated with quantum computers. Given a capable quantum machine, it would allow to factorize large numbers much faster than classical computers. This would break all common and established schemes of public key cryptography. With this threat lingering on the distant horizon, efforts of developing more robust classical cryptography schemes[1] have been the main impact of quantum computing to the research world.

In the past 15 years however, active research has produced more constructive and concrete results for applications of quantum computing. Coupled with promising progress in the hardware domain, it is warranted to explore concrete applications.

With this in mind, the German Aerospace Center dedicated resources towards evaluating possible applications of quantum computing. For example, in earth observation, satellites gather large amounts of data that needs computationally expense post-processing. In such cases, quantum computers could be able to provide a significant speed-up.

The quantum speed-up of many algorithms is powered by a subroutine that also lies at the heart Shor's algorithm. This subroutine is called quantum Fourier transform (QFT). Mathematically, it is directly related to the classical discrete Fourier transform (DFT), and hence also able to decompose a signal into its frequency components. In Shor's algorithm, the QFT is used for the closely related problem of order finding. In most of the other algorithms however, QFT is used in a different role than frequency analysis: On a suitably prepared quantum state, the inverse QFT is able to transform analog quantum amplitudes into a binary representation. This is an extremely useful tool, similar to a classical analog-to-digital converter (ADC).

The QFT is a key component in quantum phase estimation (QPE) [Kit96], quantum counting [BHT98], solving systems of linear equations [HHL09], and quantum machine learning [BWP+17] [WBL12]. Like many physical problems, applications in data processing related to earth observation can often be formulated as systems of linear equations. The algorithm for this is generally called HHL algorithm, after its authors Harrow, Hassidim and Loyd and achieves an exponential speed-up compared to the best classical algorithm.

The goal of this thesis is to develop the HHL algorithm and subsequently evaluate possible applications for Earth Observation at DLR. Of the currently discussed quantum algorithms, HHL is unique in the regard that it uses almost all of the important subroutines: It incorporates QFT, QPE, classical logic, postselection, and amplitude amplification [2]. Each of

[1] These new schemes are commonly grouped together under the term post-quantum cryptography.

[2] Very broadly speaking, there are two independent base routines that are at the heart of quantum speedups.

these subroutines will be introduced in a dedicated section of Chapter 4, but still be kept in context of their application in the HHL algorithm. This allows getting a better grasp of the otherwise often abstract algorithms that are fundamental to quantum computing. At the same time, detailed knowledge of the anatomy of HHL is built up. This knowledge enables to draw valid conclusions on the general applicability of HHL.

Chapter 5 focuses on the general applicability of HHL in real world scenarios and introduces a decision matrix to get an overview of possible use-cases. In Chapter 6, a concrete application from earth observation is evaluated in detail: In SAR tomography, three-dimensional images are reconstructed from two-dimensional ones provided by radar satellites. Also, an example run of HHL is analyzed mathematically.

This work begins with Chapter 2 and Chapter 3, which introduce the notation and some background on quantum mechanical effects that are beneficial to the understanding of the other chapters.

One is QFT, the other is the so-called amplitude amplification. It powers Grover's search algorithm [Gro96].

2 Notation and Convention

2.1 Variable Names

In quantum computing, especially when talking about complexities, there is a source of confusion: While in classical computing, the reference is the problem size (e.g. the dimension of a vector), in quantum computing, the number of qubits is used sometimes instead. This is because the number of quantum gates will generally depend on the number of qubits. On its own, this would be no problem, but in the literature, the associated variable names n, m, N, M are used inconsistently for these different quantities. This work adheres to the following convention:

- n is the dimension, or size, of the problem.
  Apart from the number of elements in a vector or dimension of a matrix, this could be the size of a database in which a search is performed.

- m represents the number of qubits used by an algorithm. Due to the nature of quantum encoding, we commonly have $m \approx \log n$.

These were chosen in part because lower- and upper case letters are cumbersome to distinguish vocally, and both frequently appear together. While j and k are used as indices in varying contexts, all variable names in Table 2.1 are used consistently.

2.2 Linear Algebra

When not otherwise specified, systems of linear equations $Ax = b$, are defined by

$$A \in C^{n \times n}; \quad x, b \in \mathbb{C}^n.$$

Non-square matrices of underdetermined systems are $A \in \mathbb{C}^{l \times n}$ with $l < n$. For any matrix, $A^\dagger$ is defined as the conjugate transpose of the matrix.[1] As the HHL algorithm will be mainly concerned about hermitian and unitary matrices, some facts are briefly stated to assist reasoning about them.

- A matrix is hermitian if $H = H^\dagger$. The diagonal can hence only contain real values. Also, all the eigenvalues of Hermitian matrices are real.

- A matrix is unitary if $UU^\dagger = 1$. If the matrix as only real entries, this property is also called orthogonal. In both cases, all the column vectors are of norm 1 and orthogonal to each other. The same holds for the row vectors.

[1] Outside of quantum physics, the notation A^* is more prevalent.

Variable	is used for
s	Matrix sparsity (in the literature, d is often used.)
t	Hamiltonian simulation time
ε	Final additive error
κ	Condition number
n	Vector and matrix dimension (in the literature, N is often used.)
l	Secondary matrix dimension (if the matrix is non-square)
m	Number of qubits
d	Number of qubits used for kickback in phase estimation
U	Unitary matrix
A	HHL input matrix
H	Hamiltonian matrix
α, β	Complex amplitudes
λ	Eigenvalues
ϕ	Complex phase of the eigenvalue $\lambda = e^{2\pi i\phi}$
u	Eigenvectors

Table 2.1: An overview of variable names used throughout this work

- Both unitary and hermitian matrices are diagonalizable.[2] Eigenvectors of diagonalizable matrices form a basis of the underlying vector space (e.g. $\mathbb{C}^n$).
- Eigenvectors of hermitian and unitary matrices are orthogonal for different eigenvalues.
- Unitary matrices only perform rotations and reflections; they cannot scale any vector as they always preserve norms. Hence, the eigenvalues can only be 1 and -1. (Actually, in the unitary case, all complex numbers with absolute value 1 are possible.)
- The action of hermitian matrices cannot be described as intuitively, but at least in the real case, the following holds for symmetrical matrices: For a square invertible matrix, the transpose A^T of a matrix will scale vectors like A, but rotate them like A^{-1}. If the matrix is symmetric, we get that A has to rotate vectors the same way as A^{-1}. As AA^{-1} has to be the identity, only rotations that are self-inverse (like rotating by 180° on an axis) are possible.

Even though this is only the real case, a rough conclusion can be drawn: Unitary matrices rotate but cannot scale, while hermitian matrices can do both, but are somewhat constrained in the possible rotations. The two classes of matrices hence seem mostly disjunct, however there are matrices that are both hermitian:

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

hermitian but not unitary | hermitian and unitary | unitary but not hermitian

[2] A matrix is diagonalizable if there exists an invertible matrix P such that $P^{-1}AP$ is a diagonal matrix. Diagonal matrices are zero everywhere except on the diagonal. Most matrices are diagonalizable, but one counterexample are nilpotent matrices.

2.3 Quantum Information

The notation of quantum states is especially inconsistent in the literature, maybe because quantum computing is still a relatively young field. This work uses the following conventions:

Binary encoding: For a binary encoded value $j = (j_{n-1} \dots j_0)_2 \in \{0,1\}^n$; the corresponding quantum binary encoding is $|j\rangle = |j_{n-1} \dots j_0\rangle$. Both represent the value

$$j = \sum_{k=0}^{n-1} j_k 2^k.$$

Hence, the left-most qubit is the most significant bit (MSB).

Basis states: The computational basis states $|k\rangle$ are defined by the quantum binary encoding $|k_{m-1} \dots k_0\rangle$ of k.

Amplitude encoding: For a vector $b = (b_0, \dots, b_{n-1})^T \in \mathbb{C}^n$ with $n = 2^m$ for some $m \in \mathbb{N}$ the corresponding quantum amplitude encoding is the state $\sum_{k=0}^{n-1} b_k |k\rangle$ and requires $\log n = m$ qubits. If n is not exactly a power of 2, this can still be done with $m = \lceil \log n \rceil$ and setting the additional values to 0. For simplicity, we generally assume $n = 2^m$.

Quantum Circuits: In a quantum circuit, the left-most (and most significant) qubit is placed on the top of the diagram.

A summary and example circuit of the conventions used in this work is depicted in Figure 2.1. Since there are two other common conventions that differ from this one, they are provided for comparison, as well.

The first is used in the book "Quantum Computation and Quantum Information" by Nielsen and Chuang and depicted in Figure 2.2. It is very similar, but uses a more mathematical style for indexing: Indices begin at 1 instead of 0, and binary states indexed from left to right, like vectors usually are. However, the most significant bit is still on the left [NC16, p. 218].

The second is used by IBM Qiskit and depicted in Figure 2.3. It uses a computer science indexing style again, but the left-most bit in Ket-notation is the lowermost bit in the quantum circuit. This can be confusing and must be kept in mind when working with Qiskit implementations of HHL.

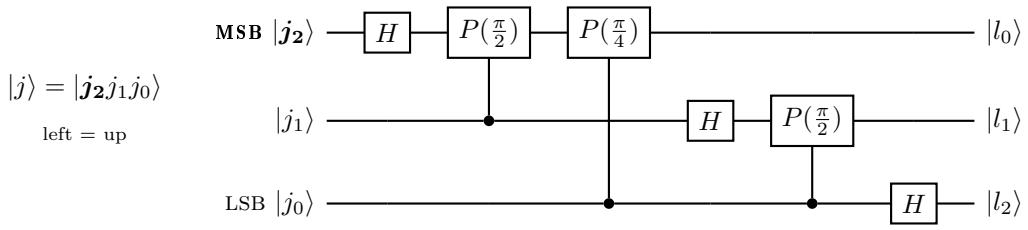


Figure 2.1: The convention used in this work and by [Hom05]

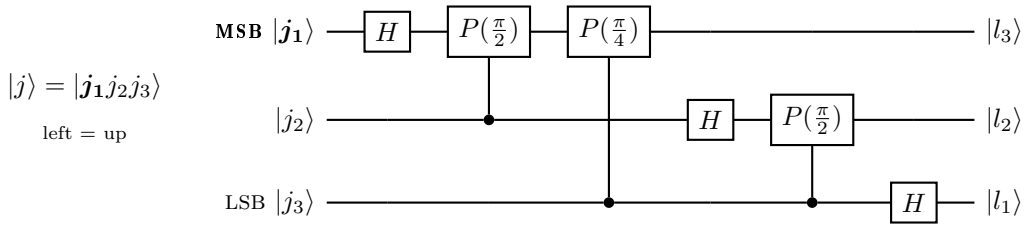


Figure 2.2: The convention used by Nielsen and Chuang [NC16]

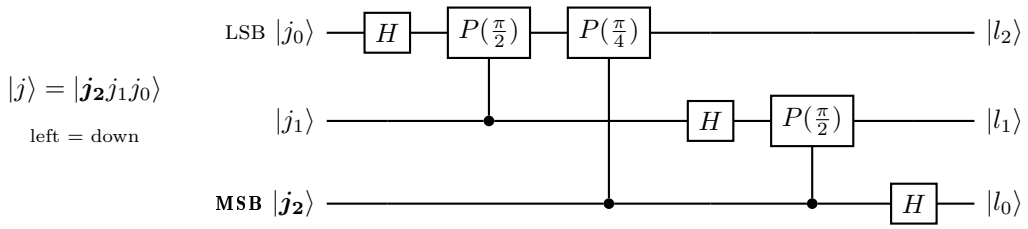


Figure 2.3: The convention used by IBM Qiskit

3 Rotations and the Bloch Sphere

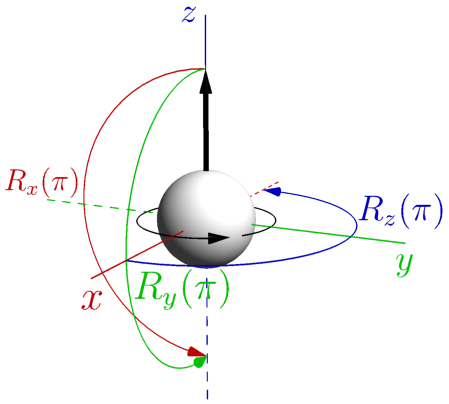
The HHL algorithm uses rotation gates in a subroutine to prepare a specific quantum amplitude. These gates, generally called $R_x(\theta), R_y(\theta)$, and $R_z(\theta)$ are reminiscent of the Euclidean rotations, but this is deceiving. They are different operations on quantum states and only called this way due to their apparent effect on vectors on the Bloch sphere. For example, they have a period of $\theta = 4\pi$ instead of 2π. To reduce confusion and aid understanding of the subroutine, this chapter develops the similarities and differences of classical and quantum rotations in detail.

3.1 Euclidean Rotations

Let's start by looking at rotations in the standard euclidean space. Imagine a spinning billiard ball with its angular momentum vector determined by the right-hand rule. Rotating the spinning ball can be described mathematically by rotating its angular momentum vector. The rotations around the axes are described by the three fundamental rotation matrices. A full rotation corresponds to an angle of $\theta = 2\pi$.

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \quad R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \quad R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

By combining different fundamental rotations, any rotation in 3D space can be achieved. The effect of applying each with angle π to the vector $(0, 0, 1)$ can be seen in this picture:



If you do some playing around with these matrices, you might come across identities like the following:

$$R_x(\pi)\, R_y(\pi) = R_z(\pi)$$

Thus, there are different combinations of the fundamental rotations producing the same result. One can prove that there are patterns in the identities that are characteristic for the rotations in 3D space. These patterns are formalized mathematically by the special orthogonal group $SO(3)$. [1]

3.2 The origin of the name spin

Rotations in 3D space are very intuitive to us humans, and hence were a natural choice for physicists when they first tried to find analogies to explain quantum phenomena in the early 1920s. In this time, when the fundamentals of quantum dynamics were actively being worked out, George Uhlenbeck and Samuel Goudsmit proposed a theory to explain the Zeeman effect. They assigned an additional quantum number to electrons and called it *spin*, suggesting electrons to be actually spinning in the classical sense like a billiard ball.

As the legend goes, Uhlenbeck hastily tried to pull back the publication after correspondence with Lorentz[2] left him convinced that his theory was false. His supervisor, Paul Ehrenfest, famously responded "Both of you are young and can afford to do something stupid.", and had the paper ([UG26]) published. Although it turned out that electrons do not actually spin, the principal idea was correct and brought the two young scientists much fame. The name spin stuck and still influences the notation and intuition we have about quantum states. As quantum particles really do have some kind of "intrinsic" angular momentum, this intuition might be helpful, but it also can cause confusion when working with the formal definitions.

3.3 Quantum Rotations

The state of a single qubit is defined by

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

where $\alpha, \beta \in \mathbb{C}$ such that $|\alpha|^2 + |\beta|^2 = 1$. The vectors are elements of $\mathbb{C}^2$. The quantum rotation gates $R_x(\theta), R_y(\theta), R_z(\theta)$ acting on such a single qubit state are defined for $\theta \in [0, 4\pi)$:

$$R_x(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} R_z(\theta) = \begin{pmatrix} e^{-\frac{i\theta}{2}} & 0 \\ 0 & e^{\frac{i\theta}{2}} \end{pmatrix}$$

It seems quite arbitrary at first that the maximum angle θ is 4π, but these modern definitions were established through a series of careful and systematic experiments over time. The first and most famous of which being the Stern-Gerlach experiment which shows that the quantum spin has to be quantized (when observed). This means that in contrast to a spinning billiard ball, whose rotation vector changes size depending on the rotation speed, electrons can only "spin" at fixed "speeds".

[1] The name comes from the special orthogonal matrices that form the group elements. Orthogonal matrices cannot scale vectors and only perform rotations or reflections. Special means that not even reflections are allowed.

[2] Hendrik Lorentz after whom the Lorentz force is named

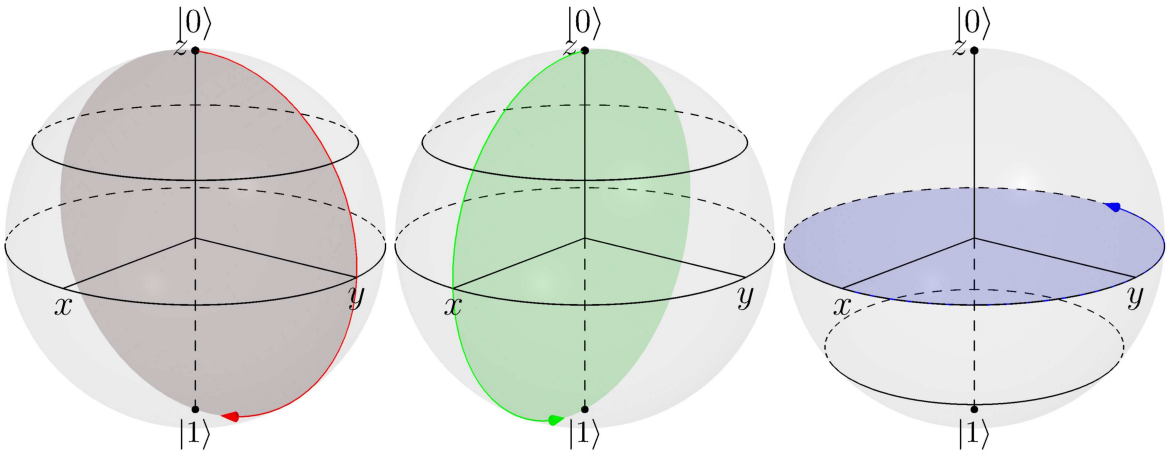


Figure 3.1: The three rotations R_x, R_y, R_z on the Bloch Sphere

Other experiments gave insights on the effect of the rotation of quantum particles: To coincide with the physical observations the mathematical structure of quantum rotations had to be different from $SO(3)$; instead, the matching structure is $SU(2)$, the special unitary group of two (complex) dimensions. However, $SU(2)$ and $SO(3)$ are quite similar. In particular, it is possible to map the elements of $SU(2)$ to elements $SO(3)$ while at the same time "preserving" the structure. In this so-called covering, precisely two elements of $SU(2)$ get mapped to each one element of $SO(3)$. This suggests that $SU(2)$ contains in some way twice as many distinct rotations. Additionally, each "half" of $SU(2)$ has the same structure as $SO(3)$.

Hence, it makes sense to use the range from 0 to 2π for one such half to highlight the similarity in structure. For the second half, the range is extended from 2π to 4π. A simple example is the rotation of the state $|0\rangle$ around the y axis:

$$\begin{aligned} R_y(0)\,|0\rangle = R_y(4\pi)\,|0\rangle &= \;\;|0\rangle & R_y(\pi)\,|0\rangle &= \;\;|1\rangle \\ R_y(2\pi)\,|0\rangle &= -\,|0\rangle & R_y(3\pi)\,|0\rangle &= -\,|1\rangle \end{aligned}$$

A rotation of 2π is "almost" the identity, but a complex phase of $e^{-i\pi}$ causes the factor -1. Only after two full rotations, the true identity is reached. The concept of the double covering also appears on Bloch sphere. Figure 3.1 shows the three rotation planes, each arrow drawing the trace of a rotation with angle π. This angle is perceived as 90° on the Bloch sphere, which is a useful side effect of extending the range in the way described above. Rotations of 2π arrive at the same vector and the same spot on the Bloch sphere, only the hidden phase has changed. The trace of a full 4π rotation travels around the Bloch sphere two times, which can be counterintuitive.

One must also take care not to confuse the quantum state vectors on the Bloch sphere with actual vectors in the three-dimensional Euclidean space. Quantum state vectors live in the different space $\mathbb{C}^2$, and the quantum rotation R_x, R_y, R_z are different from their euclidean counterparts. Their similarity in names is because of the apparent effect on the Bloch sphere. The rotation R_η seems to rotate quantum state vectors about the axis η (for $\eta \in \{x, y, z\}$).

4 The HHL Algorithm

The HHL Algorithm is usually introduced the following way: Let

$$Ax = b$$

be a system of linear equations, where $b \in \mathbb{C}^n$ and $A \in \mathbb{C}^{n \times n}$ is a hermitian matrix. Let s be the sparsity of A, i.e. the number of non-zero elements per row or column. Finally, let κ be the condition number of A. Under these conditions, HHL is capable of producing a quantum state $|x\rangle$ proportional to the solution x with an additive error of ε. The complexity is

$$\mathcal{O}(\log(n)\, s^2\, \kappa^2\, \frac{1}{\varepsilon}).$$

The vector b is provided to the algorithm normalized and amplitude encoded as $|b\rangle$ in the input register $\mathcal{I}$. The matrix A is implemented as a quantum logic gate which acts on quantum states during the execution of the algorithm.

A simplified schematic of the algorithm is depicted in Figure 4.1. It can be divided into three main parts: Phase estimation, eigenvalue inversion, and uncomputation. A HHL execution is successful if the measurement of the ancilla register $\mathcal{A}$ returns 1.

The computational complexity is dominated by the phase estimation, where the eigenvalues of A are calculated. In the next step, the eigenvalues are inverted and added to the quantum state. The result is a representation of $x = A^{-1}b$, and the problem is solved. The uncomputation step is the phase estimation in reverse. Its purpose is to reset register C back to zero, which is needed for repeated HHL executions during amplitude amplification.

The dependence on n is exponentially better than the best classical algorithm, the conjugate gradient method, which has a complexity of

$$\mathcal{O}(n\, s\, \kappa\, \log \tfrac{1}{\varepsilon}).$$

However, the complexity of HHL also depends on s, κ, and $1/\varepsilon$. If one of these factors is significantly larger than $\log n$, they dominate the overall complexity and nullify the performance gains of HHL. Conjugate gradient essentially has the same dependency on the other factors, but the play a less significant role compared to the huge n factor.

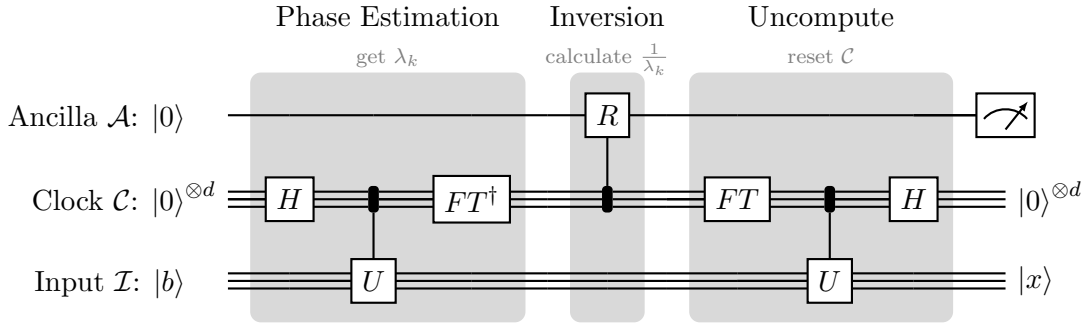


Figure 4.1: HHL praeceptum est omnis divisum in partes tres (freely quoted from [CaeBC])

The stated complexity also ignores the cost of preparing $|b\rangle$ and reading out $|x\rangle$. Doing this generally requires $\mathcal{O}(n)$ steps, limiting the practical usability. There are also some other, more concealed problems that arise from some internals. As a result, there has been a research effort towards improving the constraints and general applicability of HHL.

The main parts of HHL, phase estimation and inversion, are covered in Section 4.5 and Section 4.6, respectively. As these subroutines are of considerable complexity, several aspects are covered in advance: We start with an overview of the mathematical concept behind HHL in Section 4.1. Then, the different ways of defining computational complexities in the quantum context are introduced in Section 4.2. Literature on HHL is not consistent in this regard, so it helps greatly to keep in mind what is meant when the word complexity is used, depending on the context. Section 4.3 is concerned with a step that is often excluded from HHL, but very relevant in practice: How can the state $|b\rangle$ and the matrix A be efficiently prepared if they have to be read from classical data? Finally, after phase estimation and eigenvalues inversion, the whole algorithm is tied together by the final steps in Section 4.7.

4.1 Mathematical Overview

The most straight-forward way of solving a linear system is to calculate the inverse matrix A^{-1} which yields

$$Ax = b \iff x = A^{-1}b.$$

However, we cannot apply A^{-1} to the quantum state $|b\rangle$ since A is not unitary in general (not every invertible matrix is unitary). Instead, we use a decomposition of b into eigenvectors of A to sidestep the inversion while still achieving the desired result. This mathematical consideration will lay out the fundamental steps of the HHL algorithm. We begin by highlighting two facts:

Fact 4.1. Let $A \in C^{n\times n}$ a hermitian matrix. Then there exists an orthonormal basis $u_0, \ldots, u_{n-1}$ of C^n such that the u_k are eigenvectors of A.

Fact 4.2. Quantum states can be described with respect to any orthonormal basis. This basis is usually and implicitly assumed to be the computational basis (e.g. $|0\rangle, |1\rangle$) of standard unit vectors. The coefficients of amplitude encoded values (like $|b\rangle$ in our case) refer to these basis vectors.

The first fact allows us to write an equivalent definition of b that does not use the standard basis vectors e_k. For some $\beta_k \in \mathbb{C}$ and $u_k \in \mathbb{C}^n$ defined as above we have:

$$\sum_{k=0}^{n-1} b_k e_k = b = \sum_{k=0}^{n-1} \beta_k u_k \tag{4.1}$$

From the second fact we know that the associated quantum state can be described in an equivalent way:

$$\sum_{k=0}^{n-1} b_k |k\rangle_{\text{comp}} = |b\rangle = \sum_{k=0}^{n-1} \beta_k |u_k\rangle \tag{4.2}$$

For now, we assume that A is invertible, i.e. the linear system has precisely one solution.[1] For any invertible A with eigenvalues $\lambda_0, \ldots, \lambda_{n-1}$, the inverse matrix A^{-1} has eigenvalues $1/\lambda_0, \ldots, 1/\lambda_{n-1}$ which correspond to exactly the same eigenvectors. Since the u_k are the eigenvectors of A, we can construct

$$x = A^{-1}b = \sum_{k=0}^{n-1} \beta_k A^{-1} u_k = \sum_{k=0}^{n-1} \frac{\beta_k}{\lambda_k} u_k. \tag{4.3}$$

Given the eigenvectors of A and their inverted eigenvalues, this yields a solution to the linear system without explicitly inverting A. This has no real benefit for classical computing, though. Given eigenvectors and -values, A^{-1} can be directly calculated via reverse eigendecomposition.

The situation is different in the quantum realm: Just by loading $|b\rangle$, the decomposition into eigenvectors is already there. We would only need to transform $|b\rangle$ into a state that, when viewed in the eigenbasis, contains the inverse eigenvalues in the amplitude:

$$|b\rangle = \sum_{k=0}^{n-1} \beta_k |u_k\rangle \quad \xrightarrow{\text{transform}} \quad \sum_{k=0}^{n-1} \frac{\beta_k}{\lambda_k} |u_k\rangle = |x\rangle$$

The transformed state indeed encodes the solution vector x: With the same principle from (4.1) and (4.2), we get a decomposition $x = \sum \chi_k |u_k\rangle$ for some χ_k. Plugging this into (4.3) yields $\chi_k = \beta_k/\lambda_k$, establishing the result.

HHL relies on the quantum phase estimation algorithm (QPE) to perform this transformation, which works with unitary matrices U. The eigenvalues of any such U lie on the complex unit circle and are hence of the form $\lambda = e^{2\pi i\phi}$ for $\phi \in [0, 1)$. QPE essentially computes the phase ϕ of all eigenvalues of a given U. While A is not unitary, the matrix exponential e^{iA} is. Even more conveniently, if λ_k are the eigenvalues of A, the eigenvalues of e^{iA} are $e^{i\lambda_k}$ with the same eigenvectors. Since A is hermitian, all λ_k are in $\mathbb{R}$. This matches perfectly: The phases of $e^{2\pi iA}$ returned by QPE are exactly the eigenvalues of A. This resembles the first and main step of HHL. A second step is then needed to invert the calculated eigenvalues and correctly merge them into the overall quantum amplitudes. Further structure is needed to keep the calculations efficient, but the overall idea stays the same.

4.2 Quantum complexities

As mentioned, the HHL algorithm has a "complexity" of $\mathcal{O}(\log(n)\, s^2\, \kappa^2\, \frac{1}{\varepsilon})$. This is in contrast to $\mathcal{O}(n\, s\, \kappa\, \log(\frac{1}{\varepsilon}))$ of the conjugate gradient (CG) method [She94], the best classical algorithm for sparse matrices. The CG algorithm has an exponentially worse dependence on the dimension while being less sensitive on the desired error.

For the classical algorithm, the complexity (or run time) is well-defined: It describes the number of elementary computation steps like additions and multiplications, which are executed in (roughly) one CPU cycle each. Quantum computers on the other hand are more akin to analog devices without any state machine or clear steps upon which the complexity could be defined. As a result, there are different ways to evaluate quantum complexity: Gate count, depth complexity and query complexity.

[1] Of course, this is not generally the case, but the aim is to develop an idea for the algorithm. We will see later how to deal with non-invertible matrices.

4.2.1 Gate Count

One simple way is to count the total number of elementary gates used by an algorithm. The set of elementary gates could for example contain arbitrary rotations and the CNOT-operator [IRM+19]. Much like universal classical logic gates, different sets can be chosen. Many of the currently competing quantum architectures have different gates that can be directly implemented in hardware, making it impossible to choose a gate set that works with any architecture.

Additionally, any non-trivial unitaries appearing in the algorithm have to be decomposed into an equivalent set of elementary gates first. Doing this with the optimal number of elementary gates is an unsolved problem in the general case. If the unitary depends on input data or is somehow else chosen at runtime, calculating the number of gates gets even more difficult.

Despite these downsides, the gate count is conceptually simple and likely has high predictive power on the real-world feasibility of an algorithm: The major problems of current quantum hardware are keeping the states coherent for long enough and suppressing errors. Both errors and the likelihood for decoherence are increased by each gate, so it makes sense to use this number for evaluating an algorithm's complexity.

4.2.2 Depth complexity

The gate count is refined by the depth complexity: Since quantum gates can be performed on different qubits in parallel, the main factor for errors and decoherence is actually the longest "chain" of gates that cannot be executed in parallel. An example for this would be the controlled rotation part of HHL (Section 4.6) because all rotations act on the same qubit and hence cannot be performed in parallel.

In this context a special depth is often used, the T-depth, which counts only the number of T-gates. T-gates are often used with X, Y, and Z gates to form the Clifford+T gate set. The execution of T-gates takes considerably longer and introduces more error compared to "basic" Clifford gates, so it is the most limiting factor.

4.2.3 Query complexity

The difficulties of computing the gate count of general unitary matrices make proofs on runtimes or lower bounds very hard. [Chi21, 20.1] Because of this, there are few results that make direct statements about gate counts and depth complexity. Especially in the subject surrounding HHL, another notion of complexity is predominantly used, the query complexity. It counts the number of queries to a black box during the computation. This black box is generally the most costly part of an algorithm, so it is justified that the complexity of all other parts is simply ignored in this formalism. Although such black boxes cannot be directly implemented, the formalism is much better suited for reasoning about and comparing algorithms. The black box is formalized as a so-called quantum oracle.

Definition 4.3 [Chi21, Eqn. (20.2)]**.** Let $q, o \in \mathbb{N}$ and $f : \{0,1\}^q \to \{0,1\}^o$ be a function. A (binary) *quantum oracle* is a unitary operator O defined by its action on the computational basis states

$$|x\rangle |0\rangle^{\otimes o} \to |x\rangle |0 \oplus f(x)\rangle$$

The first (query) register with state $|x\rangle$ contains q qubits, the second (oracle) register contains o qubits and is initialized with state $|0\rangle^{\otimes o}$.

Example 4.4. A quantum oracle O_A for access to elements of the matrix A is given by its action on the basis states

$$|j,k\rangle |0\rangle \to |j,k\rangle |A_{jk}\rangle .$$

The matrix indices are binary encoded and concatenated on s (qu)bits, and it is assumed that every matrix element can be encoded in t (qu)bits.

There are other types of oracles, for example for preparing a continuous state, see [Chi21, 20.2]. Their power comes from the ability to query function values in superposition: If we queried O_A with a superposition of four different matrix coordinates, it would return a superposition of the values. With our oracle defined, we can proceed to the definition of query complexity.

Definition 4.5. The *quantum query complexity* of an algorithm is defined as the number of queries of a quantum oracle O. [Amb17] A query is one application of the unitary O; it does not matter if the input is in superposition or not.

Example 4.6. Ignoring costs for preparation of the state $|b\rangle$, the HHL algorithm has a query complexity of $\mathcal{O}(\kappa^2 s^2 \frac{1}{\varepsilon})$ by using queries to an oracle similar to the one given in Example 4.4.
In contrast to the original complexity, the query complexity given here does not depend on n. This is because all calculations that depend on n are performed by the oracle.

Note that in the query complexity setting, looking up all matrix entries at once (through supplying an equally-weighted superposition of all combinations) would only count as one query. Of course, the output superposition is not useful in most cases, but sometimes conclusions can be made, e.g. when all matrix elements are the same.

Finally, there is the concept of Quantum Turing Machines, but this type of "computation steps" is used predominantly in pure complexity theory and not for runtime analysis of specific algorithms. Most of the improvements HHL are formulated through query complexity, as we will later see.

4.3 Input Encoding

HHL requires two oracles: One for preparing the state $|b\rangle$, called O_b, and one for the matrix access as described in Example 4.4, called O_A. While it is convenient to treat them as a

black box when optimizing the overall algorithm, the problem of actually implementing them cannot be ignored either: To achieve a real-world quantum speedup at some point, there has to be a complete realization of the whole algorithm.

In the use case of processing classical linear systems with HHL, implementing O_b with a sublinear run-time seems impossible: Writing b to a classical RAM has complexity $\mathcal{O}(n)$ on its own, since b has n entries. The only exception to this is if we somehow already have b in classical RAM and "just" need to prepare the amplitude encoding $|b\rangle$. This problem is known as quantum state preparation, but as of now, there is no efficient implementation that can prepare any state, either.

Nonetheless, there are proposed solutions: For example, in a machine learning context, $|b\rangle$ does not need to be very precise, opening the possibility for an efficient algorithm that produces a "smoothed" version of b [ZFR+21]. The concept with the most traction and general applicability is called quantum RAM (QRAM). A basic QRAM implements a *binary lookup oracle*, a generalization of the one in Example 4.4:

Definition 4.7. A binary lookup oracle or data lookup oracle is a unitary operator O_{DL} defined by its action on the computational basis states

$$|i\rangle |0\rangle \to |i\rangle |w_i\rangle$$

w_i is the (binary encoded) data word at address i. The register $|i\rangle$ is wide enough to encode all addresses, and the number of qubits in $|w_i\rangle$ is the same as the word length.

Hence, a QRAM is able to perform lookups in superposition:

$$\sum_{i=0}^{n-1} \alpha_i |i\rangle |0\rangle^{\otimes d} \xrightarrow{QRAM} \sum_{i=0}^{n-1} \alpha_i |i\rangle |w_i\rangle$$

Since QRAM is its generalization, the matrix oracle O_A can be directly implemented in this way. For O_b this is not the case: we only get a binary encoding where an amplitude encoding is needed. Converting from one to the other is possible, but only in some cases efficient. Note that an implementing oracle like O_b is essentially the same as quantum state preparation, and the efficient general realization of the latter is an unsolved problem. In the following we will look at one sometimes-efficient way of implementing O_b. It is adapted from [Han21, 2.1.2].

Let n be the number of entries of b, and let $m \approx \log n$ be the number of qubits needed for the amplitude encoding. Let w be the word width of a binary lookup oracle O_{DL} which returns binary encoded entries of b. Lastly, assume that $\|b\|_\infty = 1$. This means that b has already been scaled in such a way that it is as large as possible, but no entry b_i is larger than 1, so it still can be represented as a quantum amplitude. Start with three registers:

- $\mathcal{B}$ with m qubits initialized to $\frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} |i\rangle$, an equally-weighted superposition of all basis states.
- $\mathcal{W}$ with w qubits initialized to $|0\rangle^{\otimes w}$.
- An ancillary register $\mathcal{A}$ with 1 qubit initialized to $|0\rangle$.

O_{DL} is then queried with B and returns all entries of b in superposition:

$$|0\rangle_{\mathcal{A}} \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} |i\rangle_{\mathcal{B}} |0\rangle_{\mathcal{W}}^{\otimes s} \xrightarrow{O_{DL}} |0\rangle_{\mathcal{A}} \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} |i\rangle_{\mathcal{B}} |b_i\rangle_{\mathcal{W}}$$

Next, the qubits of $\mathcal{B}$ are used to control multiple rotation gates acting on $\mathcal{A}$. With a binary value b_i as control, they are chosen such combined effect rotates $\mathcal{A}$ from $|0\rangle$ to

$$\sqrt{1 - b_i^2} |0\rangle + b_i |1\rangle .$$

The goal is to add the amplitude on $|1\rangle$ to the overall state; the square root on $|0\rangle$ is unavoidable to keep the transformation unitary. The complete operation with superposition looks like this:

$$|0\rangle_{\mathcal{A}} \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} |i\rangle_{\mathcal{B}} |b_i\rangle_{\mathcal{W}} \xrightarrow{\text{controlled rot.}} \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} \left(\sqrt{1 - b_i^2} |0\rangle + b_i |1\rangle\right)_{\mathcal{A}} |i\rangle_{\mathcal{B}} |b_i\rangle_{\mathcal{W}}$$

Now a measurement is performed on $\mathcal{A}$. If the result is 0, the operation failed and has to be started over. If the result is 1, the quantum state collapses to

$$\frac{1}{\|b\|_2} \sum_{i=0}^{n-1} b_i |1\rangle_{\mathcal{A}} |i\rangle_{\mathcal{B}} |b_i\rangle_{\mathcal{W}} .$$

Note that the normalization constant changed from

$$\frac{1}{\sqrt{\sum_{i=0}^{n-1} |1 - b_i^2 + b_i^2|}} = \frac{1}{\sqrt{n}} \quad \text{to} \quad \frac{1}{\sqrt{\sum_{i=0}^{n-1} |b_i|^2}} = \frac{1}{\|b\|_2}.$$

The desired amplitude encoding is now contained in register $\mathcal{B}$. This trick of measuring to convert from binary to amplitude encoding is called *postselection* and also used in HHL. It is covered in Section 4.7, including how to actually implement the controlled rotations.

The chance for measuring $|1\rangle$ is $\frac{1}{n} \sum_{i=0}^{n-1} |b_i|^2$. If the elements of b are very uniform, the $\|\cdot\|_\infty$-normalization causes the values to be close to 1. In this case, the success probability approaches 1, making the state preparation quite efficient. Unfortunately, if b is not uniform, the prospects are bleak. There are other conversion techniques, but they have drawbacks, too: One approach can trade oracle queries for increased success probabilities. Another one requires knowledge of "sub-norms" of parts of b to achieve an efficient deterministic preparation [Han21, p. 14-15]. For yet another approach, see [ZFR+21].

4.3.1 QRAM implementation

There are currently a few proposed architectures for implementing a quantum RAM. They are called Fan-out, Bucket-Brigade [GLM08] and Flip-Flop [PPR19]. There is also the Kerenidis-Prakash (KP) hybrid model with the sub-norm technique mentioned above [KP16].

The Fan-Out model is derived from classical memory architecture, so a brief overview is given: Modern RAM is laid out in rows (one for each address) and columns (e.g. $w = 64$, the word width), forming a grid. Each row line controls w single-bit memory cells. If the

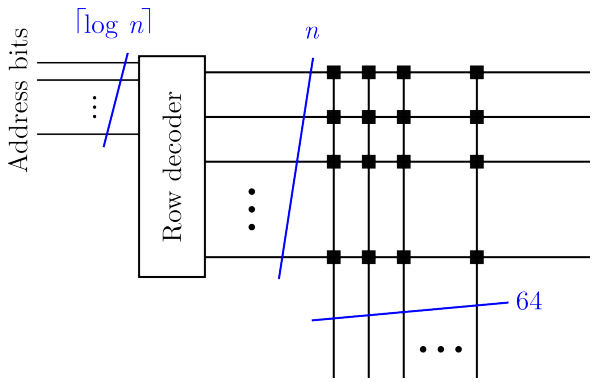


Figure 4.2: Schematic of a modern RAM with a word width of 64 bit and n memory cells

row line is turned on, the cells write their values to the column lines to be transported back to the CPU. For decoding the binary address and turning on the corresponding row line, a decoder is used. This component needs at least $\mathcal{O}(n)$ transistors arranged into a binary-tree pattern that get turned on and off by the address bits [GLM08, Fig. 1, 2]. Like any classical circuit, this could be implemented in a quantum computer, but since all $\mathcal{O}(n)$ quantum logic gates are active at the same time, an extremely large state must be kept coherent. This seems to be a major hurdle for an actual practical implementation.

As an improvement, [GLM08] introduced the Bucket-Brigade architecture. Instead of turning all the routing transistors on or off based on the address qubits, they use a three-phase hardware element that is able to both route and transfer the data. They start in a special waiting state and "lock in" their routing behavior depending on the first arriving signal. After that, all further signals are passed to the locked-in route. As a result, only the logic directly on the selected route is active, reducing introduced errors and the chance of decoherence. The three-phase hardware elements are called *qutrits* and could theoretically be realized through a number of physical phenomena. All of them require a purpose-built memory structure different from general-purpose quantum processors.

[PPR19] introduced a different method which uses standard quantum gates. Here, classical memory bits are used as input to CNOT gates in a quantum register that has as many qubits as a classical memory word. These selectively flipped qubits are themselves input to controlled rotations acting on a register for the final amplitude-encoded state. After this operation, the same classical memory bits are used to flip back, or "flop", the first quantum register. This process can then be repeated on the same registers with the other memory words, building up the final quantum state. This approach was combined with the probabilistic post-selection routine covered earlier to improve an issue discovered in the original Flip-Flop QRAM design in [DVDAPDS20].

Finally, the approach of Kerenidis and Prakash [KP16] was first developed for encoding matrices in a quantum recommendation system, which is an application related to HHL. It stores the amplitudes of different parts $\sum b_i^2$ of b in a binary tree structure. The amplitude encoding is constructed by subsequent rotations which depend on the values stored in the nodes of the tree.

With n vector entries in classical memory, the classical complexity to save it is $\mathcal{O}(n)$ and the quantum circuit cannot involve fewer than $\mathcal{O}(n)$ logical elements - even if the lookup itself is more efficient when not all of them are used.

4.4 Output Decoding

The only possibility to convert data saved in a quantum register back to classical data is to perform a measurement. Any measurement has to be performed with respect to a basis. Current quantum computers are generally only capable of performing measurements in the standard basis, which is also called computational, or Z basis. For one qubit, the Basis vectors are $|0\rangle$ and $|1\rangle$. With 5 qubits, $|15\rangle = |01111\rangle$ would be one of the 2^5 basis vectors between $|0\rangle$ and $|31\rangle$. Note that the $|0\rangle$ here implicitly refers to $|00000\rangle$.

With an arbitrary basis, the probability of obtaining the basis vector $|\gamma\rangle$ when measuring the quantum state

$$\sum_j |\psi_j\rangle \qquad \text{is} \qquad \sum_j \langle\psi_j| P_\gamma^\dagger P_\gamma |\psi_j\rangle ,$$

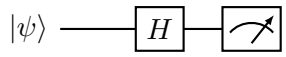
where the matrix $P_\gamma = |\gamma\rangle \langle\gamma|$ is the projection onto state $|\gamma\rangle$ [NC16, Sec. 2.2.3]. With the standard basis, the probability of obtaining $|k\rangle$ when measuring the quantum state

$$\begin{aligned}
\sum_{j=0}^{2^m-1} \underbrace{\alpha_j |j\rangle}_{=|\psi_j\rangle} \qquad \text{is} \qquad \sum_{j=0}^{2^m-1} \langle\psi| P_k^\dagger P_k |\psi\rangle &= \sum_{j=0}^{2^m-1} \alpha_j^\dagger \langle j| P_k \alpha_j |j\rangle \\
&= \sum_{j=0}^{2^m-1} |\alpha_j|^2 \langle j| |k\rangle \langle k| |j\rangle \\
&= |\alpha_k|^2 \cdot 1 \cdot 1 + \sum_{\substack{j=0 \\ j\neq k}}^{2^m-1} |\alpha_j|^2 \cdot 0 \cdot 0 \\
&= |\alpha_k|^2.
\end{aligned}$$

This is because $\langle j| |k\rangle = 1$ if and only if $j = k$. Binary decoded values can be trivially decoded by performing one measurement, as they just consist of one single standard basis vector. Its measurement probability will be 1 and all the others will have 0 probability. Recovering the amplitudes of amplitude encoded values is not as easy, since all the α_j might be nonzero. The brute force method is to repeatedly prepare the quantum state and measure it to obtain information about the relative frequencies of each basis vector $|k\rangle$. From this, we can approximate the probabilities $|\alpha_j|$. The Chebyshev inequality may be employed to get an idea how many measurements are needed to "almost" guarantee a low approximation error. For such a guarantee, $\mathcal{O}(n^2)$ measurements are needed, but more efficient methods might exist.

Independent of the error, classically storing the values in classical RAM takes at least $\mathcal{O}(n)$ classical write operations, like with the read operations when encoding. So even if there are more efficient methods, decoding to classical memory cannot get faster than $\mathcal{O}(n)$.

Knowledge of the probabilities $|\alpha_j|^2$ is only sufficient to reconstruct the vector x if we already know that all entries are real and positive. In this case, we can calculate the square root to get the amplitudes which correspond to the entries of x. If we only know that the entries are real, we need to get more information to reconstruct the individual signs. For a single qubit, sign information may be obtained by measuring in the Hadamard basis. As this is generally physically not possible, we transform the state and subsequently measure in the standard basis to get the same effect:



From the outcome of measuring we can deduce if the two entries of the vector have the same or different sign. The global phase however is impossible to measure, so we cannot differentiate between two negative and two positive entries. This might not be too bad since we can just try to solve the linear system with x and $-x$ to see which works. For multiple qubits, this deduction gets harder as there are way more combinations to rule out. If the entries of x are complex, these calculations get even harder. The problem of completely reconstructing all (complex) amplitudes of a quantum state is known as *quantum tomography* and non-trivial to solve. See for example [GLF+10], [CPF+10].

To summarize, decoding $|x\rangle$ to classical memory is unfavorable in to ways: We need $\mathcal{O}(n)$ to store the information in classical RAM, and the cost for actually obtaining information seems to be at least in the same order of magnitude. It definitely seems best if the solution $|x\rangle$ is processed further on the quantum computer, and only a simpler final result is measured in the end.

4.5 Phase Estimation

Quantum Phase Estimation (QPE) is a subroutine which is at the core of not only HHL, but also Shor's and other quantum algorithms. It combines two mechanisms, phase kickback and quantum Fourier transform, to achieve a quite unique effect: Ignoring some details, the QPE is essentially able to transform from an amplitude encoding to a binary encoding. Amplitude encoded values are notoriously hard to measure and of limited use as input for controlled gates, so this ability adds a lot of flexibility when designing quantum circuits.

Algorithm: Quantum Phase Estimation

- Input:
  - Quantum implementation of unitary $U \in \mathbb{C}^{2^m \times 2^m}$
  - Eigenvector v of U, amplitude-encoded as $|v\rangle$ in Register $\mathcal{I}$ with m qubits. Note that in this situation we have $U|v\rangle = e^{2\pi i\phi}|v\rangle$ with $\phi \in [0, 1)$.
  - Output register $\mathcal{C}$, initialized to $|0\rangle^{\otimes d}$
- Output:
  - Binary encoding of the phase $|\phi\rangle$ on d qubits in register $\mathcal{C}$, precise to approximately 2^{-d}
- Complexity:
  - Qubits: $m + d$
  - Gates: $\mathcal{O}(\text{poly}(d) + 2^d \cdot T_U)$
    The first summand is caused by the QFT, which is polynomial in d. The second summand depends on the cost T_U of implementing a single execution of U. If there is no efficient method available, T_U can reach $\mathcal{O}(2^m)$.

With the algorithm overview we can make the above statement more concrete: QPE does not directly convert between encodings, but gives the complex angle or *phase* of the eigenvalue of the supplied vector v. Since the eigenvalues of unitary matrices always have length one, this is equivalent to knowing the complete eigenvalue, so you could reasonably name the algorithm quantum eigenvalue estimation. If ϕ cannot be exactly represented with d qubits, there are errors introduced. For now, we ignore these errors.

4.5.1 Quantum Fourier Transform

The quantum Fourier transform (QFT) is often introduced as quantum counterpart to the discrete Fourier transform (DFT), which is ubiquitously used in signal processing or compression algorithms to transform data from the "time domain" to the "frequency domain". For example, periodic peaks in the time domain cause a peak at the corresponding frequency in the frequency domain and vice versa. The mathematical operator of the DFT is defined component-by-component:

$$y_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j e^{2\pi i\, jk/n}, \quad k = 0, \ldots, n-1 \tag{4.4}$$

It transforms a vector $x = (x_0, \ldots, x_{n-1})$, most often comprised of measured values in the time domain, to a vector $y = (y_0, \ldots, y_{n-1})$ representing the amplitudes of certain frequencies in the frequency domain. The QFT acts analogously on an amplitude-encoded vector:

$$\sum_{k=0}^{n-1} x_k \left|k\right\rangle \xrightarrow{QFT} \sum_{k=0}^{n-1} y_k \left|k\right\rangle$$

where y_k is defined exactly as in (4.4). Ignoring any overhead from encoding or decoding the data, QFT is able to perform this operation on $m = \log_2(n)$ qubits and $\mathcal{O}(m^2)$ gates. The fastest classical algorithm, the fast Fourier transform (FFT), has a runtime of $\mathcal{O}(n \log n)$. This might make QFT interesting to be used on its own, but the FFT is already fast enough for all common use cases, and encoding/decoding would bring QFT to $\mathcal{O}(n)$ complexity anyway.

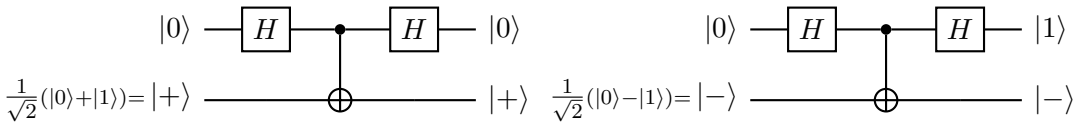
Instead, most quantum algorithms make use of QFT as a subroutine in a way that is not directly connected to the interpretation of time and frequency domain. The QFT can also be described through a mathematically equivalent product representation:

$$\begin{aligned} &\left|j\right\rangle = \left|j_{m-1} \ldots j_0\right\rangle \xrightarrow{QFT} \\ &\frac{1}{\sqrt{n}} \Big(\left|0\right\rangle + e^{2\pi i\, 0.j_{m-1}} \left|1\right\rangle\Big) \Big(\left|0\right\rangle + e^{2\pi i\, 0.j_{m-1} j_{m-2}} \left|1\right\rangle\Big) \cdots \Big(\left|0\right\rangle + e^{2\pi i\, 0.j_{m-1} j_{m-2} \cdots j_0} \left|1\right\rangle\Big) \end{aligned} \tag{4.5}$$

Nielsen and Chuang state [NC16, p. 218]: "This product representation is so useful that you may even wish to consider this to be the definition of the quantum Fourier transform." It relates binary encoded values with phases and is used to build the QPE, combined with the phase kickback effect, which is introduced in the following section.

4.5.2 Phase Kickback

In some situations, controlled gates exhibit an effect that is very counterintuitive when compared to classical conditioned operations work. At the same time, this effect turns out to be incredibly useful in quantum algorithms. It is called phase kickback, but let us not focus too much on the name at first and consider the following two quantum circuits where the controlling qubit is in equal superposition between $|0\rangle$ and $|1\rangle$:



Counterintuitively, the controlled qubit is unaffected, while the controlling qubit on the right is flipped from $|0\rangle$ to $|1\rangle$.

The controlled qubit is unaffected because the input states $|+\rangle$ and $|-\rangle$ were deliberately chosen to be *eigenstates* of the X-Gate. This means that the associated[2] vector is an eigenvector of the matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

For eigenvectors v, the result of Av will still be v, scaled (multiplied) by the eigenvalue λ. For eigenstates $|v\rangle$, the result of $X|v\rangle$ will be $|v\rangle$, just "scaled" in its amplitude. But since the eigenvalues of unitaries are complex numbers with absolute value 1, multiplying λ to the amplitude does not scale, but shift the *phase*, or complex angle.

To summarize: Even if the X-Gate was turned on, it would not affect $|+\rangle$ or $|-\rangle$, only the phase of the amplitude. This phase change is responsible for flipping the controlled qubit. To see how this works, let us look at the CNOT-Gate in the right circuit. The eigenvalue of eigenstate $|-\rangle$ is -1:

$$\begin{aligned} &\text{CNOT}\left(\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |-\rangle\right) \\ =&\tfrac{1}{\sqrt{2}}\left(|0-\rangle + |1\rangle \text{X} |-\rangle\right) \\ =&\tfrac{1}{\sqrt{2}}\left(|0-\rangle - |1\rangle |-\rangle\right) \\ =&\tfrac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \otimes |-\rangle \end{aligned}$$

Due to the superposition in the controlling qubit, the eigenvalue affects not the overall amplitude, but only the "part" where the X-Gate was turned on. The eigenvalue, or phase, is thus effectively transferred the opposite way to the classical intuition. This is the reason for the name phase kickback.

In the circuit on the right, the second Hadamard Gate transforms this changed phase into the final flipped controlling qubit. In the left circuit, precisely the same kickback is occurring. But in this case, the eigenvalue of $|+\rangle$ is 1, so it does not actually change the amplitude. The generalized description of this effect is given by Definition 4.8.

[2] by amplitude encoding

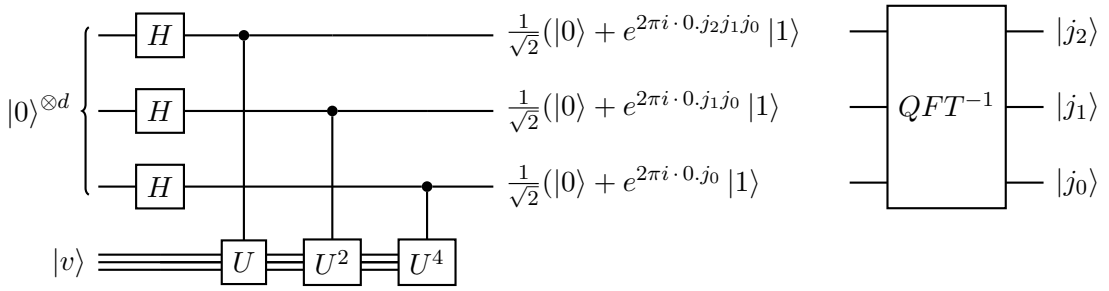
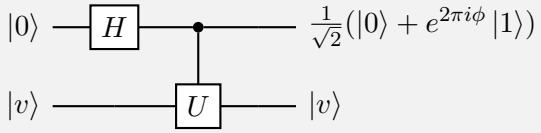


Figure 4.3: A circuit for QPE with 3 qubits

Definition 4.8 Phase Kickback**.** Let U be a unitary matrix and v and eigenvector of U with eigenvalue $e^{2\pi i\phi}$ and $\phi \in [0, 1] \subset \mathbb{R}$. Under these conditions, the following circuit produces a phase kickback into the controlling qubit:



The state of the controlling qubit bears a striking resemblance to the QFT output states in their product representation from (4.5). This observation is the key to the Quantum Phase Estimation.

4.5.3 Standard QPE

To match the notation introduced in (4.5), let ϕ from Definition 4.8 be represented by the binary representation $\phi = 0.j_{m-1}j_{m-2}\dots j_0$. After executing a phase kickback with U and $|v\rangle$, the resulting state is $\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i \cdot 0.j_{m-1}j_{m-2}\dots j_0} |1\rangle)$. If we were able to produce the remaining states of (4.5), we could use the inverse QFT to generate the state $|j_{m-1}j_{m-2}\dots j_0\rangle$, which is the binary representation of ϕ on m bits.

This is actually quite easy to do: For $Uv = \lambda v$ and $k \in \mathbb{N}$ we have $U^k v = \lambda^k v = e^{2\pi i \cdot k\phi}$. If we multiply not by k, but 2^k instead, the binary digits are shifted to the left by k places. Any digits shifted left of the decimal point represent whole numbers. As $e^{2\pi i\phi} = e^{2\pi i(\phi+z)}$ for any $z \in \mathbb{Z}$, these digits can be dropped from the equation. With this information, the factors of QFT's product representation can be built using controlled-U^{2^k} gates that each depend on a different qubit. A full circuit with $d = 3$ control qubits is depicted in Figure 4.3.

This technique, first introduced by [Kit96], is quite remarkable and at the heart of many quantum algorithms, not just HHL. It provides a useful and efficient translation between amplitude and binary encoded values. The drawback is that implementing U^{2^k} with quantum gates is hard to do in the general case: Both Finding a suitable decomposition into elementary gates, and executing a potentially huge number of them without incurring prohibitive errors. For example, implementing a unitary on m qubits requires approx 4^m gates, and the decomposition using classical computers requires time exponential in m [IRM+19, Table I.].

4.5.4 Adjusting for HHL

As described in Section 4.1, the purpose of QPE is to calculate the eigenvalues of A, but A on itself cannot be used as a matrix because it is not unitary. Instead, the matrix exponential $U = e^{2\pi i A}$ is used.

Fact 4.9. Let $H \in \mathbb{C}^{n \times n}$ be a hermitian matrix and $\lambda_0, \ldots, \lambda_{n-1} \in \mathbb{R}$ be the eigenvalues of H. (Note that hermitian matrices always have real eigenvalues.) Then the matrix exponential $e^{2\pi i A}$ has exactly the same eigenvectors, while the corresponding eigenvalues are $e^{2\pi i \lambda_k}$.

Fact 4.9 fits perfectly for the use-case of HHL. When using $U = e^{2\pi i A}$, the decoded phase angle ϕ represents the real eigenvalues of A. These binary encodings lend themselves for the classical operations in the next step at Section 4.6, like computing the reciprocal and as a control input for the rotations.

This causes one of the more subtle constraints that HHL imposes on the input matrix A: The eigenvalues should be not larger than 1. Otherwise, $e^{2\pi i \lambda}$ "loops around" and resulting values are no longer unique. Hence, the matrix might have to be scaled. This can be easily done during Hamiltonian simulation (see Section 4.5.5), but the size of the eigenvalues are not known a priori. Estimating them is possible, but a bad estimation scales them down too far and increases the error.

Another discrepancy to the definition of QPE is that b is required to be an eigenvector of U (or equivalently, A), but this is not the case. Any vector that is not an eigenvector is equivalent to a corresponding superposition of eigenvectors from the eigenbasis $\{u_k \mid 0 \leqslant k \leqslant n-1\}$ of eigenvectors of A:

$$|b\rangle = \sum_{k=0}^{n-1} \beta_k |u_k\rangle$$

Due to the linearity of quantum operations, the output of QPE will be an equally weighted superposition of binary encodings

$$\sum_{k=0}^{n-1} |\lambda_k\rangle .$$

Looking back to (4.3) from Section 4.1, this is exactly what we need. After QPE, we are on the halfway point towards preparing the final quantum state:

$$|b\rangle = \sum_{k=0}^{n-1} \beta_k |u_k\rangle \quad \xrightarrow{\text{QPE}} \quad \sum_{k=0}^{n-1} \beta_k |u_k\rangle |\lambda_k\rangle \quad \xrightarrow{\text{inversion}} \quad \sum_{k=0}^{n-1} \frac{\beta_k}{\lambda_k} |u_k\rangle = |x\rangle$$

The inversion step is described in Section 4.6. Before that, we still need to cover the actual implementation of U. Like already mentioned, for general U, there is no efficient algorithm. HHL solves this problem by restricting itself to sparse matrices, which can be implemented through a technique called Hamiltonian simulation.

4.5.5 Hamiltonian Simulation

Hamiltonian simulation is a well-established technique that was originally developed to simulate quantum mechanical systems. In fact, this was the first-ever proposed use-case for quantum computers introduced by Feynman in 1972. How a quantum mechanical system

evolves over time is specified by a hermitian matrix H, called *Hamiltonian*. If the system is in starting state $|\psi(0)\rangle$, the state $|\psi(t)\rangle$ at any point in time t can be calculated with the Hamiltonian H and the *Schrödinger Equation*:

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

The problem of efficiently calculating e^{-iHt} is called Hamiltonian simulation. The reason why Feynman proposed quantum computing for solving this problem is that the dimension of the Hamiltonian is exponential in the size of the quantum system. Because of this, the problem is near-impossible to solve with classical computers.

HHL uses the quantum algorithms that have since been developed to do Hamiltonian simulation for its own purposes: During the phase estimation, calculating U^{2^k} is exactly the same as "simulating" e^{iA2^k}, as long as A is hermitian. Of course, A does not represent an actual quantum system, but to the algorithms this does not matter.

HHL refers to [BACS07] for doing the actual simulation. Since the publication of HHL, the research activity has increased considerably in the field of Hamiltonian simulations, but apart from some improvements, the general constraints imposed have stayed largely the same:

- A has to be hermitian[3]
- A has to be sparse
- There has to be an efficient function giving back entries of A.

In the remainder of this section, we will outline the basic concepts behind the originally referenced algorithm and the improvements.

All algorithms for Hamiltonian simulation fall into two categories: One is based on product formulas, which are also often named Lie, Trotter and/or Suzuki formulas. The second is based Quantum Walks. Both of them are only efficient for sparse Hamiltonians, but differ in their other characteristics.

Product Formulas

The method referred in the original HHL paper uses the product formula approach [BACS07]. In a first step, a decomposition of the hermitian A is calculated such that

$$A = \sum_{j=1}^{\mathcal{O}(s^2)} A_j$$

and each of the A_j is 1-sparse. 1-sparse means that in each column and each row of the matrix there is at most 1 non-zero element. For an s-sparse A, on the order of s^2 individual A_j are needed. The decomposition is calculated using a graph-theoretic algorithm called distributed edge coloring: The matrix A is represented as a graph by adding one vertex for each row and each column of A. Then, an edge is set to be present between a row vertex x and a column vertex y if and only if A_{xy} is nonzero. As A is hermitian ($A_{xy} = A_{yx}$), the graph can be undirected. Two edges are said to be *incident* if they share a vertex. In our

[3] This condition can be lifted, see Section 5.2.1

model, incident edges mean that the corresponding matrix entries are in the same column, or in the same row. Any selection of edges that are not incident hence represents a 1-sparse matrix.

The well-known graph theoretic problem of edge coloring is to assign each edge a color such that no two incident edges have the same color. Generally, it is desirable to use as few colors as possible for this and there are many algorithms for specific use cases. [BACS07, Lemma 2] uses one that is efficient for sparse graphs. After its execution, every edge is colored. Each set of edges with the same color represents a 1-sparse matrix. A graph theoretic argument gives that $\mathcal{O}(s^2)$ colors are needed to always find an edge coloring.

The 1-sparsity makes the individual A_j efficient to implement [Aho04, Lemma 4.1] on their own. The sum decomposition can be used to approximate the Hamiltonian simulation through product formulas, a principle described by Lie and Trotter:

$$e^{A+B} = \lim_{r\to\infty} (e^{A/r} e^{B/r})^r$$

In the application of Hamiltonian simulation, they use that

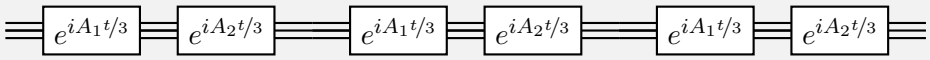
$$e^{i\sum_j A_j t} \approx \Big(\prod_j e^{iA_j t/r}\Big)^r$$

for some large r. The error of this approximation is difficult to estimate [CST+21], but generally roughly $\mathcal{O}(\frac{t^2}{r})$. Note that maybe contrary to intuition, the following equation does not hold in general:

$$e^{A+B} = e^A e^B$$

This would require that the product of A and B commutes. For matrices, this is not the case in general, and only holds if their so-called *commutator* is equal to zero. Hence, the need for the above approximations. Implemented on an actual quantum computer, this simple but wrong formula would equate to just executing the unitaries e^A and e^B consecutively. The Lie-Trotter approximation equates to executing a small step of each A_j and repeating this r times.

Example 4.10. A quantum circuit for doing a Lie-Trotter approximation with $A = A_1 + A_2$ and $r = 3$ looks like this:



In [BACS07, Section III.] an equivalent, more complicated, but slightly more efficient version of the product formula by Suzuki [Suz90] is used.

Alternatives to Product Formulas

This combination of sparse decomposition and product formulas has been the main technique up to the 2010s, when the publication of HHL sparked a lot of research interest which produced different, (asymptotically) more efficient methods. Still, as of 2022, any comprehensive implementation on NISQ hardware uses the basic product formula technique. See [CMN+18] or the paragraph about Qiskit in Section 6.3.

The first direction of improvement is to replace the product formula and use a different means of approximating. [BCC+15] also divide the simulation into r steps, but they use the Taylor series representation of the exponential function to calculate

$$e^{2\pi i A t/r} = \sum_{k=0}^{\infty} \tfrac{1}{k!} \big(2\pi i A \tfrac{t}{r}\big)^k.$$

The Taylor series of course needs to be truncated somewhere in actual implementation. [BCC+15] show that the number of terms has only a logarithmic dependence on the inverse of ε, the required precision. This yields an exponential improvement when compared to product formulas. As the A themselves are not unitary,

$$(2\pi i A \tfrac{t}{r})^k$$

cannot be directly implemented. Instead, the sparse decomposition $A = \sum_j A_j$ allows to apply a result from [Kot14] which yields an equivalent linear combination of unitaries V_j:

$$\sum_{k=0}^{K} \tfrac{1}{k!} \big(2\pi i (\textstyle\sum_j A_j) \tfrac{t}{r}\big)^k = \sum_j \beta_j V_j$$

The unitaries V_j are each built from a certain subset of the A_j:

$$V_j = A_{j_1} \cdot A_{j_2} \cdots A_{j_{\max}}$$

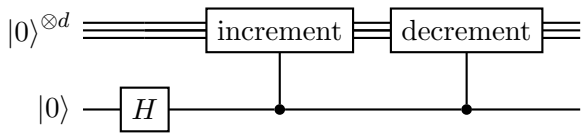
Named after this core part, the technique is called linear combination of unitaries.

Quantum Walks

Another technique, quantum walks, is based on the same principle as the classical random walk. The classic counterpart is well established and has many forms, but the most common example involves a drunk person starting out on a certain point in 2D space. In each time step, the person makes a drunk step into a random direction that is independent of all previous steps. The theory of random walks addresses questions on the most likely position of the person after n steps, or more precisely, the probability density.

Classical random walks have broad applications in finance, where stock values are modeled in a similar way. The problem from above can also be simplified to one dimension: Start at 0 on a number line representing $\mathbb{Z}$. On each time step, move either to the right $(+1)$, or to the left (-1) with equal probability. The intuition suggests that after many time steps, we will be likely be somewhat close to 0 and less likely far away, as the latter would require rolling e.g. "+1" almost all the time which is unlikely.

Such a walk can be simulated in superposition on a quantum computer. Consider a register containing a binary encoded value, starting at 0. In each step, two conditioned gates act on the register: One performs a binary increment operation, the other one a decrement. The quantum circuit for one step would look like this:



One might suspect that this is an efficient way to calculate the whole walk "in superposition", and that the probability distribution when measuring after many steps should give the highest probabilities to binary states close to $|0\rangle$. However, counterintuitively, this is not the case. Quantum interference causes a pattern that is quite different from the expected one [Kem03, Fig. 6]. This makes it hard to think about the effects of such *quantum walks*, but the still provide a lot of value when used carefully.

The quantum walk that is used for Hamiltonian simulation is a walk on graphs. Since a plethora of problems can be modeled as a graph, it makes sense that sometime insights about randomly walking along edges of the graph can be helpful. This can be, for example, the probability distribution, that is, which nodes have the highest probability for the drunken person to end up on them. In the quantum and the classical case, some information about this distribution can be calculated with matrices that reflect something about the graph. Example for this are the adjacency matrix or the graph's Laplacian. These calculations involve eigenvalues of the matrices, so there are already known results about their relationship.

Using this knowledge, it is possible to construct an "artificial" setup for a quantum walk, such that the eigenvalues of the matrix $\tilde{U}$ performing a single step are similar to the eigenvalues of A. Executing such a step triggers oracle queries to O_A, but overall, the process is as efficient or even better than the product formula approach (at least asymptotically). The eigenvalues of $\tilde{U}$ are of the form

$$\pm e^{\pm i \arcsin \lambda},$$

where λ are the actual eigenvectors of A [BC12, Sec. III.], [BCK15, Sec. 3.1]. After the phase estimation, these values need to be corrected to remove the arcsin-part. While asymptotically efficient, computing trigonometric functions adds significant gate overhead. Additionally, the quantum walk approach inherently uses double the amount of qubits compared to product formulas. This makes quantum walks a bad fit at least for NISQ hardware [CMN+18].

The asymptotic efficiency is not to be ignored though: All quantum walk approaches have a reduced dependence on the sparsity, which is only s instead of s^2. Furthermore, the dependence on the error can be improved to be only $\log 1/\varepsilon$, but this improvement was later also applied to product formulas. Indeed, many refinements introduced to these approaches can be used for both, giving rise to a series of algorithms. [Kot17] provides a good overview; the findings are also summarized in Table 4.1.

4.6 Eigenvalue Inversion

After the phase estimation, the quantum circuit is in the overall state

$$\sum_{k=0}^{n-1} \beta_k |\lambda_k\rangle_{\mathcal{C}} |u_k\rangle_{\mathcal{I}},$$

where $\sum \beta_k |u_k\rangle = |b\rangle$ from register $\mathcal{I}$ was not changed by QPE, and $|\lambda_k\rangle$ are the binary encoded eigenvalues. Recall that we want our final state to be

$$\sum_{k=0}^{n-1} \beta_k \frac{1}{\lambda_k} |0\rangle_{\mathcal{C}} |u_k\rangle_{\mathcal{I}} \approx |0\rangle |x\rangle,$$

Divide and Conquer or quantum walks	Method for implementing $\exp(-iHt)$	Query Complexity
Divide and conquer	Product Formulas [BACS07] [CK10]	$\mathcal{O}(s^3 t(st/\varepsilon)^{1/2k})$
Quantum walks	Phase estimation on quantum walks [Chi10] [BC12]	$\mathcal{O}(st/\sqrt{\varepsilon})$
Divide and conquer	Fractional queries [BCC+14], trunc. Taylor series [BCC+15]	$\mathcal{O}(s^2 t \frac{\log(s^2 t/\varepsilon)}{\log\log(s^2 t/\varepsilon)})$
Quantum walks	Linear combination of Quantum Walks [BCK15]	$\mathcal{O}(st \frac{\log(st/\varepsilon)}{\log\log(st/\varepsilon)})$
Lower Bound	Shown in [BCC+14] [BCK15]	$\Omega(st + \frac{\log(1/\varepsilon)}{\log\log(1/\varepsilon)})$
Quantum Walks	Quantum Signal Processing [LC17]	$\mathcal{O}(st + \log(1/\varepsilon))$
Divide and Conquer	Qubitization, Quantum Signal Processing [LC19]	$\mathcal{O}(s^2 t + \log(1/\varepsilon))$

Table 4.1: An overview of the query complexity of algorithms for Hamiltonian simulation

so we have to modify the amplitude of the overall quantum state to add the factor $1/\lambda_k$. HHL achieves this by rotating the ancilla qubit $\mathcal{A}$, essentially performing

$$\mathcal{A} = R_y(\theta)\,|0\rangle = \sqrt{1 - \frac{1}{\lambda^2}}\,|0\rangle + \frac{1}{\lambda}\,|1\rangle\,,$$

where $\theta = 2 \arcsin 1/\lambda$. This ancilla qubit is subsequently measured, which is not a unitary operation and hence gives the algorithm a chance of failing: If the measurement returns $|1\rangle$, the resulting quantum state is only affected by the correct amplitude, giving the desired result. If it is $|0\rangle$, the opposite is true and the algorithm is started over. Since the value of θ is only known at runtime, the rotation cannot be fixed beforehand. But, given a binary encoding of θ, it can be assembled through a series of controlled rotations, each subsequent one rotating by half the angle.

4.6.1 Computing the reciprocal eigenvalues

We have the binary encodings $|1/\lambda_k\rangle$ from QPE, but for the rotations, these need to be transformed to $|2 \arcsin 1/\lambda_k\rangle$. This step can be implemented with classical logic. To compute arcsin, it is easiest to use a truncated version of its Taylor series:

$$\arcsin x \approx x + \frac{1}{6}x^3 + \frac{3}{40}x^5$$

For $x < 1$, the approximation error is quite small, and this approach requires only a few multiplications to get the result. Computing the reciprocal is also not very resource intensive. For example, classical algorithms like Newton's Method or Goldschmidt division could be used. Both are fast; approximately logarithmic in the number of bits used to encode the numbers. Hence, the complexity of the overall calculation is $\mathcal{O}(\log d)$ in the number d of clock register qubits. Since HHL already has a worse dependence on these parameters, this subroutine is of no consequence for the overall complexity of the algorithm, even with (at most linear) quantum overhead.

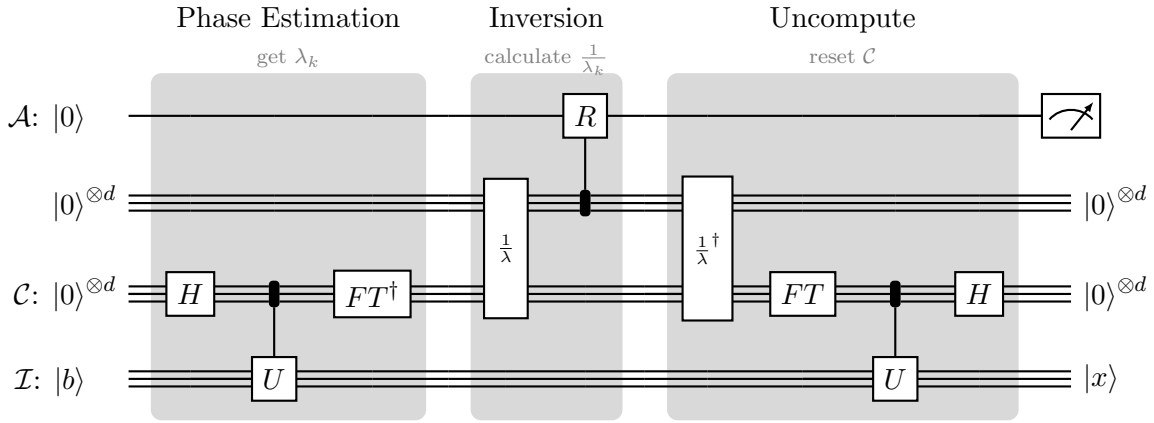


Figure 4.4: The HHL algorithm with expanded eigenvalue inversion

While the classical division algorithms are efficient in theory, some gates are very expensive to implement or introduce considerable errors in current quantum computers. A recent algorithm by Thapliyal et al. [TMVH18] optimizes for these constraints. No matter the concrete choice of algorithm, some more ancilla qubits are needed, expanding the HHL circuit as depicted in Figure 4.4.

As the eigenvalues were restricted to be in the interval $[1/\kappa, 1]$, they will be at least one after the inversion. But as an amplitude of a quantum state they must be at most one. The Taylor series approximation also only works for values smaller than one. To solve this problem, we introduce a scaling factor C which gets multiplied to all inverted eigenvalues. The whole rotation becomes

$$\mathcal{A} = R_y(\theta)\,|0\rangle = \sqrt{1 - \frac{C^2}{\lambda^2}}\,|0\rangle + \frac{C}{\lambda}\,|1\rangle\,.$$

To form a valid quantum state, C needs to be smaller than the smallest eigenvalue λ_{min}. Of course, the eigenvalues are generally not known beforehand. Instead, C is typically set to be approximately $1/\kappa$, as this guarantees that even the smallest eigenvalue is at most one when inverted. While the exact condition number of the matrix is not known either, HHL only works for matrices with small condition anyway, so you can just assume the smallest sensible value for κ. If C is too small however, the probability of measuring $|1\rangle$ diminishes. An adaptive approach might be useful, increasing C slowly if such a case is detected.

4.6.2 Rotating the Ancilla Qubit

The most straight-forward way of implementing the rotation by θ radians is to use an array of fixed "part-rotations" and condition each of them on one qubit. The first rotation is set up to rotate the "full way" to $|1\rangle$, the second one "half-way" to $|1\rangle$, and so on.

As $R_y(\pi)$ rotates the full way to $|1\rangle$ (see Section 3.3), we start with this operation as the first gate and halve the angle each step. Using the fact that

$$R_y(a + b) = R_y(a)R_y(b),$$

we are able to approximate any rotation on $\mathcal{A}$ by choosing which "part-rotations" are turned on.

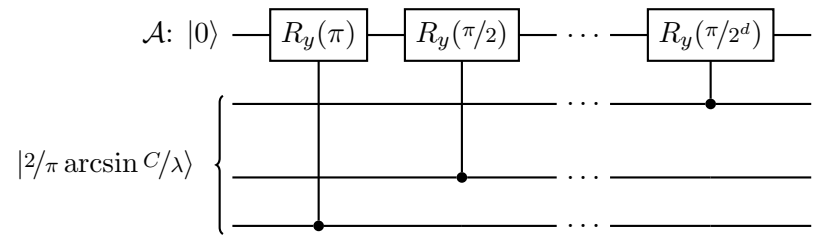


Figure 4.5: Implementation of controlled rotation

The angle θ is given by $2 \arcsin C/\lambda$. This value will be in the range $[0, \pi]$, which is the raw angle of the rotation. To use its binary encoding as a control input, we have to divide by π to get a range of $[0, 1]$. Now the largest possible raw angle π will correspond to the bit-pattern $|1000...\rangle$, such that only the control input to $R_y(\pi)$ is active. A rotation by a raw angle of $\pi/2$ produces $|0100...\rangle$, and so on. The resulting circuit is shown in Figure 4.5.

4.7 The complete algorithm

Now we are equipped to go through a full execution of the HHL algorithm in detail. We work with the registers $\mathcal{A}$, $\mathcal{C}$, and $\mathcal{B}$ like defined in the beginning of this chapter. To start the algorithm, we assume that the following state can be prepared efficiently:

$$|0\rangle_{\mathcal{A}} |0\rangle_{\mathcal{C}}^{\otimes d} |b\rangle_{\mathcal{B}}$$

This problem is not solved yet for arbitrary $|b\rangle$. Possible approaches are discussed in Section 4.3. The first step is to perform a round of quantum phase estimation (Section 4.5). This puts a superposition of the eigenvalues of A into register $\mathcal{C}$. The superposition is weighted by the factors β_j of the decomposition of b into A's eigenbasis.

$$|0\rangle_{\mathcal{A}} |0\rangle_{\mathcal{C}}^{\otimes d} |b\rangle_{\mathcal{B}} = \sum_{j=0}^{n-1} |0\rangle_{\mathcal{A}} |0\rangle_{\mathcal{C}}^{\otimes d} \beta_j |u_j\rangle_{\mathcal{B}} \xrightarrow{\text{Phase Estimation}} \sum_{j=0}^{n-1} |0\rangle_{\mathcal{A}} |\lambda_j\rangle_{\mathcal{C}} \beta_j |u_j\rangle_{\mathcal{B}}$$

Next, the inversion and rotations are performed according to Section 4.6. This adds two amplitudes in superposition.

$$\sum_{j=0}^{n-1} \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right)_{\mathcal{A}} |\lambda_j\rangle_{\mathcal{C}} \beta_j |u_j\rangle_{\mathcal{B}}$$

4.7.1 Postselection and uncomputation

We are only interested in one of these amplitudes, so we use the postselection trick from Section 4.3 and perform a measurement with respect to the computational basis on the ancilla bit $\mathcal{A}$. This collapses the state; if we measure $|1\rangle$, the state turns into

$$\sqrt{\frac{1}{C^2 \sum_j |\beta_j/\lambda_j|^2}} \sum_{j=0}^{n-1} \frac{C}{\lambda_j} |1\rangle_{\mathcal{A}} |\lambda_j\rangle_{\mathcal{C}} \beta_j |u_j\rangle_{\mathcal{B}}.$$

The relevant part for the final measurement is

$$\sum_{j=0}^{n-1} \frac{\beta_j}{\lambda_j} |u_j\rangle_{\mathcal{B}} = |x\rangle ,$$

the solution. The algorithm is now basically finished, but HHL includes the inverse operations of all gates affecting register $\mathcal{C}$. These are all gates only the controlled rotations are excluded. This step is called uncomputation and its effect is to reverse the contents of register $\mathcal{C}$ back to $|0\rangle^{\otimes d}$. As the uncomputation is performed before the measurement, the state before measuring actually looks like this:

$$\sum_{j=0}^{n-1} \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right)_{\mathcal{A}} |0\rangle_{\mathcal{C}}^{\otimes d} \beta_j |u_j\rangle_{\mathcal{B}}$$

As seen above, the register $\mathcal{C}$ is irrelevant for reading out the result. So why go through the effort of "resetting" it back to $|0\rangle$? Doing so helps mitigate a problem that arises from the measurement. Since we need to measure $|1\rangle$ to obtain the result, we need to redo the whole computation every time when measuring $|0\rangle$. The probability of measuring $|1\rangle$ is

$$C^2 \cdot \sum_{j=0}^{n-1} \left| \frac{\beta_j}{\lambda_j} \right|^2 \geqslant C^2 \cdot \sum_{j=0}^{n-1} |\beta_j|^2 = C^2 \cdot 1 = \mathcal{O}(\tfrac{1}{\kappa^2}).$$

We get the first inequality from the fact that all the λ_j are no larger than 1 by assumption of HHL. As the β_j are the coefficients of orthonormal basis vectors and $\|b\|_2 = 1$, we have $\sum_j |\beta_j|^2 = 1$. Finally, we know from Section 4.6 that $C = \mathcal{O}(1/\kappa)$.

This means that to have a good probability for measuring $|1\rangle$, we need to repeat the whole algorithm $\mathcal{O}(\kappa^2)$ times. To reduce this number, the authors of HHL incorporate the whole algorithm into an amplitude amplification routine. Amplitude amplification is an established technique to improve the chances of a marked outcome being measured [BHMT02]. This reduces the number of repetitions to $\mathcal{O}(\kappa)$, but requires that the algorithm is performed on the same qubits. To allow for this, register $\mathcal{C}$ needs to be reset.

4.7.2 Fine touches employed in HHL

It should be noted that HHL does not actually use the "classical" QPE implementation scheme from Section 4.5: Instead of performing multiple controlled-U^{2^k} operations, the following single operator is constructed.

$$U = \sum_{\tau=0}^{T-1} |\tau\rangle \langle\tau| \otimes e^{-iAt_0 \frac{\tau}{T}}$$

The first part $|\tau\rangle \langle\tau|$ acts as a control input, the second part as the matrix. The exponent is realized through the simulation time $t_0 = \mathcal{O}(\kappa/\varepsilon)$. Despite these large differences in notation, the complexity and principle stay essentially the same. The operator is also adapted to a second change: The control qubits in register $\mathcal{C}$ are not initialized to $|0\rangle^{\otimes d}$, or

$$H^{\otimes d} |0\rangle^{\otimes d} = \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} |k\rangle ,$$

if you include the application of the individual Hadamard gates. Instead, the specialized state

$$|\Psi_0\rangle = \sqrt{\frac{2}{T}} \sum_{\tau=0}^{T-1} \sin \frac{\pi(\tau + \frac{1}{2})}{T} |\tau\rangle$$

is used, which can be efficiently prepared, too. In analogue to the classical Fourier transform, the differing starting state "selects" a different sampling instead of the standard equidistant time step or frequency points. Again, the principle stays the same.

This needs to be done because the binary encoded eigenvalues are actually subject to errors introduced by QPE if they cannot be represented exactly with d bits. You might think that this error is only a rounding error on the order of 2^{-d}, but this is not the case: In practice, errors are introduced on every bit with a certain probability. This can be mitigated at the cost of additional runtime (performing the QPE again), but also by optimizing the sampling with $|\Psi_0\rangle$.[4]

Suppose from now on that the QPE was successful and able to produce the binary values up to an error δ. Calculations on the more complex version of HHL (which involves the states and operators from above) show that to achieve a final additive error ε on the result $|x\rangle$, the error from QPE must be at most $\mathcal{O}(\varepsilon/\kappa)$. In consequence, d has to be chosen such that $2^{-d} = \mathcal{O}(\delta) = \mathcal{O}(\varepsilon/\kappa)$. The maximum exponent will be $2^d = \mathcal{O}(\kappa/\varepsilon)$. In the formula above, this is included as the simulation time t_0.

T indirectly controls the size d of register C, as it has to have enough qubits to contain the largest $|\tau\rangle$, hence $\log T \approx d$, or $T \approx 2^d$. In the paper, T is chosen to be approximately $\log(n)s^2\kappa/\varepsilon$. This is similar to the classic QPE which has a factor of κ/ε. For implementation cost, the extra cost from $\log T = \log\log n + 2\log s + \log \kappa/\varepsilon$ is negligible compared to $d = \log \kappa/\varepsilon$.

4.7.3 Runtime analysis

The stated runtime of the original HHL algorithm is

$$\mathcal{O}(\log(n) \cdot s^2 \cdot t_0 \cdot \kappa),$$

where $t_0 = \mathcal{O}(\frac{\kappa}{\varepsilon})$. The individual factors are caused by different subroutines of the algorithm:

- $\log(n)s^2$ is the cost of simulating $U = e^{2\pi i A}$
  By using improved Hamiltonian simulation techniques, the dependence on the sparsity can be reduced from s^2 to s.

- $\frac{\kappa}{\varepsilon}$ is the number of U executions to reach the required exponents U^{2^k}.
  Improving this factor is difficult since the resulting precision is inherent to QPE.

- κ repetitions of the whole algorithm are needed to guarantee a good success probability (of measuring $|1\rangle$).
  Without amplitude amplification, κ^2 repetitions are needed. Otherwise, not much can be done to improve this factor.

While the original paper includes n in the complexity, almost all of the improvements use query complexity and only count the number of times that an oracle returning entries of A

[4] See also [NC16, p. 224, Eg. (5.35)]

has to be invoked. While this is certainly a more theoretic approach that sidesteps an actual implementation, it can be defined very precisely. The following theorem is such a description of the query complexity of HHL with the best improvements already applied:

Theorem 4.11 [CKS17, Thm. 2]**.** *The quantum linear system problem can be solved by a gate-efficient algorithm that makes* $\mathcal{O}(\frac{s\kappa^2}{\varepsilon}\,\mathrm{poly}(\log(\frac{s\kappa}{\varepsilon})))$ *to the oracle* $\mathcal{P}_A$ *and* $\mathcal{O}(s\kappa\,\mathrm{poly}(\log(\frac{s\kappa}{\varepsilon})))$ *to* $\mathcal{P}_B$*.*

The operator poly signifies that the term is a polynomial of the argument. For example, $\mathrm{poly}(\log n)$ could denote $(\log n)^2 + \sqrt{\log n}$, or $(\log n)^{1000}$. The oracle $\mathcal{P}_A$ is defined similar to the one in Section 4.6, while a query to $\mathcal{P}_B$ is used to prepare the quantum state $|b\rangle$. Formally, an algorithm with query complexity Q is gate-efficient if its gate complexity is $\mathcal{O}(Q\,\mathrm{poly}(\log Q, \log N))$.

4.8 Related Algorithms

Apart from the improvements to Hamiltonian simulation, there have been works adapting or changing the algorithm itself. Most notably, [CKS17] improve the dependence on the precision from $1/\varepsilon$ to $\log 1/\varepsilon$ by replacing the quantum phase estimation with methods bases on Fourier series representation or Chebyshev polynomials. Relatively soon after the original HHL publication, [CJS13] developed a method of incorporating a (quantum) preconditioning routine to be able to work with ill-conditioned linear systems. This comes at a cost of a increased dependence on the sparsity. More recently, [WZP18] developed a change to HHL in the other direction: Their method improves performance when the matrix is dense. However, the dependence on the sparsity is still $\sqrt{s}$, "only" a quadratic improvement. Moreover, the creators of HHL introduced a method for simulating dense matrices if they are of low rank [RSML18]. For a non-sparse, non-hermitian, or even possibly non-square matrix A their method is able to simulate $e^{2\pi i A t/n}$ with $\mathcal{O}(\frac{t^2}{\varepsilon}\|A\|^2_{\max})$ to the matrix element access oracle. Note that there is no dependence on the sparsity at all; $\|A\|_{\max}$ is the largest absolute value of entries in to the complexity A.

However, a dependency on the condition gets introduced nonetheless by the phase estimation itself: Recall that for the desired final precision, the resulting $|\lambda_k\rangle$ of QPE need to be precise to $\mathcal{O}(\frac{\kappa}{\varepsilon})$. Another hurdle is that only $1/n \cdot A$ is simulated, so the eigenvalues would be scaled to λ_k/n. We could scale A to $A' = nA$, but this would introduce a factor n^2 to the complexity through $\|A'\|^2_{\max}$. We also could scale the eigenvalues after the phase estimation, but this would amplify the error to $\mathcal{O}(n\,\kappa/\varepsilon)$. Counteracting this again introduces a factor of n to the runtime. If the requirements on the precision are very low, the method might be feasible nonetheless. One example could be if $|x\rangle$ is known to be very sparse and only the locations of a few large non-zero entries are needed.

Lastly, a different approach tailored for near term quantum hardware is called variational quantum linear solver and essentially combines HHL-like steps with classical computation [BPLC+19]. This method trades asymptotic efficiency for a smaller qubit and gate footprint.

5 Constraints and Decision Matrix

The exponential speedup of HHL can only be achieved if all the conditions and assumptions of the algorithm are met. Some conditions can be lifted, but other more subtle restrictions cause significant problems in practice. In this chapter, all constraints are summarized in one place and their impact is discussed. This includes topics that were already touched on in the previous chapters, but new aspects are also introduced. We start with an exemplary HHL execution values to get a more insight on how the algorithm works, and how the values and calculations behave in a more concrete setting.

5.1 Guided Tour of one HHL execution

We take the matrix A to be diagonal to simplify less relevant calculations and because the eigenvalues can be read directly from the diagonal. As any hermitian matrix can be diagonalized, we do not really restrict ourselves to some "special" subset of matrices, but rather focus only on the "nice" part. Like with any hermitian matrix, the diagonal elements and eigenvalues are real. To further simply calculations, we assume that they are positive, too. Setting $n = 8$, we get a system of linear equations $Ax = b$, formed by

$$A = \begin{pmatrix} \lambda_0 & & & \\ & \lambda_1 & & \\ & & \ddots & \\ & & & \lambda_7 \end{pmatrix}, \qquad x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_7 \end{pmatrix}, \qquad b = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_7 \end{pmatrix},$$

where $\lambda_0, \dots \lambda_7 \in \mathbb{R}^+$ are the positive eigenvalues of $A \in \mathbb{R}^{8\times 8}$, and $b, x \in \mathbb{C}^8$. We define

$$\lambda_{\max} = \max_{k\in\{0\dots7\}} \lambda_k \quad \text{and} \quad \lambda_{\min} = \min_{k\in\{0\dots7\}} \lambda_k.$$

With this we have

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}.$$

We assume $\lambda_{\max} = 1$ to get $\lambda_{\min} = \frac{1}{\kappa}$ and hence satisfy this further condition of HHL. To get more concrete, we set $\kappa = 10$ and $\varepsilon = \frac{1}{100}$. As the solution vector has 8 entries, a uniform solution $|x\rangle$ would have amplitudes of $\frac{1}{\sqrt{8}} \approx 0.35$. An error of 0.1 would be comparatively large, so 0.01 seems like a good idea. This sets the precision for QPE to $t_0 \approx \frac{\kappa}{\varepsilon} = 1000$. To achieve this, we will use $d = 10$ qubits, as $2^d = 1024$.

A diagonal A also makes reasoning about $|b\rangle$ easier: The eigenvectors $|u_k\rangle$ of diagonal matrices are unit vectors, or in the quantum context, the computational basis states $|k\rangle$. Hence, when preparing $|b\rangle$, the amplitudes b_k are the same as the β_k from the eigendecomposition:

$$|b\rangle = \sum_{k=0}^{7} b_k |k\rangle = \sum_{k=0}^{7} \beta_k |k\rangle$$

Now we are ready to start the phase estimation process and prepare the matrices

$$U^{2^k} = e^{2\pi i A 2^k} = \begin{pmatrix} e^{2\pi i\, \lambda_0\, 2^k} & & & \\ & e^{2\pi i\, \lambda_1\, 2^k} & & \\ & & \ddots & \\ & & & e^{2\pi i\, \lambda_7\, 2^k} \end{pmatrix}$$

for $k \in \{1, \ldots, d = 10\}$. By performing the controlled-U operations after applying the Hadamard gates, the phases containing the eigenvalues are kicked back into the $d = 10$ qubits of register $\mathcal{C}$. The most significant (uppermost) qubit in register $\mathcal{C}$ is now in the state

$$\sum_{k=0}^{7} b_k \tfrac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.\lambda_{k_9}\lambda_{k_8}\ldots\lambda_{k_0}}).$$

The least significant (lowermost) qubit is in state

$$\sum_{k=0}^{7} b_k \tfrac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.\lambda_{k_0}}).$$

The remaining qubits contain the corresponding phase values in between. Inverse QFT transforms register $\mathcal{C}$ to

$$\sum_{k=0}^{7} b_k \,|\lambda_k\rangle\,,$$

assuming the QPE was successful. Each $|\lambda_k\rangle$ is now a 10-bit binary representation of λ_k. These values are then scaled and inverted using classical logic. After that, they control the conditioned rotation which acts on register $\mathcal{A}$. The initial state $|0\rangle$ is transformed to

$$\sum_{k=0}^{7} b_k \left(\sqrt{1 - \tfrac{C^2}{\lambda_k^2}}\,|0\rangle + \tfrac{C}{\lambda_j}\,|1\rangle \right).$$

With a scaling set to $C = \frac{1}{\kappa} = \lambda_{\min}$ and $\lambda_{\max} = 1$, let us investigate the summands corresponding to the largest and smallest eigenvalue. They are

$$\sqrt{1 - \tfrac{1}{\kappa^2}}\,|0\rangle + \frac{1}{\kappa}\,|1\rangle \text{ for } \lambda_{\max} \qquad \text{and} \qquad \sqrt{1 - \tfrac{\kappa^2}{\kappa^2}}\,|0\rangle + \frac{\kappa}{\kappa}\,|1\rangle \text{ for } \lambda_{\min}.$$

Recall that the scaling factor C is needed to make sure that the inverted eigenvalues are not larger than one, as otherwise they would not be a valid amplitude, making it impossible to build a corresponding rotation on $\mathcal{A}$. In our ideal case, C is chosen perfectly such that the inverse of the smallest eigenvalue is exactly one. Unfortunately, this scales down the amplitude of the inverted large eigenvalues as well, decreasing the overall chance to measure $|1\rangle$. If C has to be guessed and is too small, the chance is decreased further.

After uncomputation and postselection, the whole state is

$$\frac{1}{C\sqrt{\sum_k |b_k/\lambda_k|^2}} \sum_{k=0}^{7} \frac{C}{\lambda_k} b_k \,|1\rangle\,|0\rangle^{\otimes d}\,|k\rangle\,,$$

if the measurement returned $|1\rangle$. Notice that C (which equals $\lambda_t min$ in our case) eliminates itself. With the resulting normalization factor

$$\mathcal{N} = \sqrt{\sum_k \left|\frac{b_k}{\lambda_k}\right|^2},$$

we have that the probability of measuring each state $|k\rangle$ is

$$\mathbf{P}(|k\rangle) = \left|\frac{b_k}{\mathcal{N}\lambda_k}\right|^2.$$

By performing repeated measurements, we can reconstruct the amplitudes and form our solution vector

$$x_{HHL} = \frac{1}{\mathcal{N}} \begin{pmatrix} b_0/\lambda_0 \\ b_1/\lambda_1 \\ \vdots \\ b_7/\lambda_7 \end{pmatrix}.$$

This is the exact solution x, only scaled by $1/\mathcal{N}$.

5.2 Constraints

5.2.1 Summary of already discussed constraints

Hermitian A

The requirement of A being hermitian can be relaxed in practice. By using an intermediary matrix, non-hermitian and even non-square matrices can be used with HHL. Suppose that $A \in \mathbb{C}^{c\times n}$ $(c, n \in \mathbb{N})$ is non-hermitian and potentially non-square, and let

$$H := \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix} \mathbb{C}^{(n+c)\times(n+c)}.$$

Then H is hermitian and square by construction. Invoking HHL with H and $|b'\rangle = |0\rangle^{\otimes m} |b\rangle$, will return a state $|1\rangle^{\otimes m} |x\rangle$[1], that is, up to normalization, equivalent to

$$|x\rangle = \mathcal{A}^+ |b\rangle,$$

where A^+ is the so-called Moore-Penrose pseudoinverse of A [DHM+18, Sec. 3.6]. If A is square and full rank (but not hermitian), there is still a unique solution to the linear system and $A^+ = A^{-1}$.

If the linear system is underdetermined (either when A is not full-rank or when $c < n$), there are many possible solution vectors w. The solution given by the pseudoinverse matrix

$$x = A^+ b$$

[1] The actual state is $\sum \frac{\beta_j}{\sigma_j} |1\rangle^{\otimes m} |v_j\rangle$ with singular values σ_j and singular vectors v_j. Singular values are a generalization of eigenvalues that are needed for non-square non-hermitian matrices. Note also that $m \approx \log(n + c)$ in this setting.

is the one with the smallest norm of all possible solutions [Gol12, Prop. 19.5]:

$$\|x\|_2 = \inf_{w:\ Aw=b} \|w\|_2$$

If the linear system is overdetermined (when $c > n$ and A has full rank), there is no solution. However, one is often interested in the least-squares solution in this case, which is the vector x for which

$$\|Ax - b\|_2$$

is as small as possible. $x = A^+ b$ returns precisely this least-squares solution [DHM+18, Sec. 3.6].

To summarize, not only can the hermitian condition be relaxed, but HHL provides very useful results even if the matrix is non-square: In both the under- and overdetermined case, one would classically be interested in exactly the vectors that are returned by applying the Moore-Penrose pseudoinverse to b. The layout of HHL produces this state elegantly and automatically without adaptions to the logic.

Preparation of $|b\rangle$ and input of A

As discussed in Section 4.3, preparing $|b\rangle$ for general b with complexity close to $\log n$ is an unsolved problem. Additionally, the procedure requires that all n entries of b be present in a classical RAM, which always needs at least n classical operations to prepare. However, if b is very uniform or has other known characteristics, specialized methods might perform significantly better.

Quantum operations that depend on A usually employ calls to a black box function to obtain single entries of A. This is only feasible if A is sparse, that is, $s = \mathcal{O}(\log n)$. The black box could be realized with a quantum RAM, or compute the values of A on the fly.

Readout of $|x\rangle$

As discussed in Section 4.4, decoding all amplitudes of $|x\rangle$ absolutely requires at least $\mathcal{O}(n)$ measurements. As measuring destroys the state, HHL must be repeated for each measurement. This would bring the dependence on n up from $\log n$ to $n \log n$, so HHL seems to be suited only for cases where $|x\rangle$ is further processed on the quantum computer itself.

Condition of A

The condition number κ of A poses constraints in two parts of the algorithm: First, the worst-case probability of measuring 1 in the postselection diminishes for large κ. Depending on the eigenvalues, the probability might be better[2] , but generally this limitation is inherent to the algorithm and requires $\mathcal{O}(\kappa)$ repetitions of HHL to get a valid result (see Section 4.7).

Second, a bad condition increases the precision required of the phase estimation. This linearly affects the number of U executions, causing another factor $\mathcal{O}(\kappa)$ in the overall complexity. This could be improved by replacing the phase estimation with another subroutine that has a better dependence on precision.

For a very well written overview of these constraints, also see [Aar15].

[2] For example, if only a single eigenvalue is 1 and all the others are almost equal to $\lambda_{\min}$, the condition is bad. But the overall probability of measuring 1 is still quite high, since most of the inverted values cause a rotation close to $1\,|1\rangle$.

5.2.2 Further constraints

$|b\rangle$ needs to be prepared multiple times

Since measuring the ancilla register $\mathcal{A}$ irreversibly changes the quantum state including the entangled register $\mathcal{B}$, the state $|b\rangle$ is changed as well. Thus, $|b\rangle$ needs to be prepared anew for every repetition of HHL (as long as the measurement returns 0). This occurs $\mathcal{O}(\kappa)$ times, yielding κ queries to the oracle O_b from Section 4.3. The additional queries can affect the overall performance, no matter if $|b\rangle$ is prepared from classical data or the output of a previous quantum procedure.

Scaling of A to fit assumptions about the eigenvalues

In the original HHL paper, the authors make the assumption that all eigenvalues of A lie between 1 and $\frac{1}{\kappa}$. This is due to the phase kickback during phase estimation: By exponentiating $U = e^{-2\pi i A}$, the eigenvalues λ are encoded into the phases as $e^{2\pi i \lambda}$. If the eigenvalues are larger than one, this representation is no longer unique, breaking the algorithm.[3]

The authors of HHL note that to fix this, the matrix can be scaled such that the eigenvalues fulfil the condition. To do this however, the eigenvalues of A need to be known, but we generally do not know anything about them. Most of the time, not even $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$ is known. There are some bounds on the eigenvalues, but they are not very sharp and any mismatch scales the matrix down too far, increasing the error unnecessarily.

The overall error indirectly depends on n

The output error ε of HHL guarantees that

$$\| |x\rangle - |\tilde{x}\rangle \|_2 < \varepsilon$$

holds [HHL09, Thm 1]. Here $|x\rangle = |A^{-1}b\rangle$ is the theoretically ideal solution, while real $|\tilde{x}\rangle$ is the real solution returned by the algorithm. So ε essentially is the additive error on the result. However, the vector represented by $|x\rangle$ is normalized. When scaling it to the size of the actual vector, the error increases to

$$\| \|x\|_2 |x\rangle - \|x\|_2 |\tilde{x}\rangle \|_2 < \|x\|_2 \varepsilon.$$

If n is very large, the norm of x will be generally large as well. The only exceptions are when x is very sparse or when almost all the entries of x are very small. To combat this loss of precision, we would need to introduce a factor of e.g. $1/n$ into the precision. So instead of just requiring a precision of e.g. $1/100$, we would require $1/100 \cdot n$.

Most of the versions of HHL have a linear dependence on $\frac{1}{\varepsilon}$, so doing so would introduce a factor of n to the runtime, negating the exponential improvement. The improved version [CKS17] is not affected.

Another way of thinking about this is that each entry of a relatively uniform and normalized vector is approximately $1/n$. An additive error of $1/100$ is negligible for a two-dimensional vector, where each entry is $1/2$. For a vector of length 100 however, such an error is on the same order of the entries, which likely is undesirable.

The last two of these further constraints are covered more in-depth in [Sha18].

[3] As a note, negative eigenvalues can be represented by exponentiation of $e^{\pi i A}$ instead of $e^{2\pi i A}$, such that positive values $\leqslant 1$ only cause a phase angle up to 180. The negative values then turn the angle "the other way around".

5.3 Decision Matrix

To get a better overview of the situation, all the improvements and alternative forms of HLL are collected in a table, where each of them is evaluated with regard to the constraints discussed. A red "✘" signifies that the algorithm either cannot be used in this case or has a worse runtime than the classical algorithm. A yellow "(✔)" is set when the algorithm can be used in principle, but there are some drawbacks. For example, the "Dense Adaption" can be used for dense matrices, but since the runtime is $\mathcal{O}(\sqrt{n})$, the improvement is not as significant as an exponential speedup. A green "✔" stands for constraints that pose no problem for an implementation. For example, non-square matrices can be used with all algorithms via the technique introduced in Section 5.2.1. Finally, the algorithms are also evaluated on their suitability for practical implementations and the number of citations of the individual papers.

Note that as the matrix cannot cover every possible use case, some subtleties get lost. For example, the preparation from classical b is marked as red, because there is no efficient method for the general case. However, as discussed in Section 4.3, if b is very uniform, an efficient preparation can be possible. For such concrete cases, the matrix can be easily adapted, which has been done in Table 6.1.

				Improvements to Hamiltonian Simulation				
	Classic HHL	Precon-ditioning[a]	Dense Adaption[b]	Trotteri-zation[c]	Quantum Walks[d]	Theoretical Optimum[e]	Fourier Series[f]	Dense, Low-rank[g]
classical b input	✘	✘	✘	✘	✘	✘	✘	✘
$\kappa(A) \gg \log n$	✘	✔	✘	✘	✘	✘	✘	(✔)
$s(A) \gg \log n$	✘	✘	(✔)	✘	✘	✘	✘	✔
non-square A	✔	✔	✔	✔	✔	✔	✔	✔
non-hermitian A	✔	✔	✔	✔	✔	✔	✔	✔
dependence on s	s^2	s^7	$\approx \sqrt{s}$	s^2	s	s	s	no dep.
dependence on κ	κ^2	depends	κ^2	κ^2	κ^2	κ^2	$\approx \kappa^2$	$\approx \kappa^2$
dependence on ε	$1/\varepsilon$	$1/\varepsilon^2$	$1/\varepsilon$	$\log 1/\varepsilon$	$\log 1/\varepsilon$	$\log 1/\varepsilon$	$\log 1/\varepsilon$	$\approx 1/\varepsilon^2$
NISQ performance	?	?	?	✔	✘	?	(✔)	?
pract. implementation	(✔)	?	?	✘	✘	✘	(✔)	?
confidence/citations	✔	(✔)	(✔)	✔	✔	✔	✔	(✔)

[a] [CJS13]
[b] [WZP18]
[c] [LC19]
[d] [LC17]
[e] [BCK15]
[f] [CKS17]
[g] [RSML18]

6 Applications

Since current quantum computers only have a few qubits, exploring applications of HHL on real-world hardware is impossible. Simulations are possible in principle, but their complexity is exponential, so again only small examples can be calculated.

Hence, we make use of the constraint matrix developed the last chapter to evaluate the feasibility of HHL (and with it the QFT) for earth observation. The use case of SAR tomography is introduced as a representative example. Its properties are then discussed in detail to correctly apply them to the constraint matrix.

The chapter is closed with an overview of existing implementations on actual quantum computers. These only work with very small vectors (e.g. $n = 2$), but are a useful resource for the understanding of HHL.

6.1 Space-born SAR Tomography in Urban Areas

The German Aerospace Center (DLR) is working with radar backscatter data from earth observation satellites. These satellites are equipped with a synthetic aperture radar (SAR). Synthetic aperture means that multiple radar measurements at different positions (along the satellite's orbit) are mathematically combined to achieve the equivalent resolution of a much bigger antenna. Tomography is the task of reconstructing three-dimensional features in the scanned area, often from looking at the same point from different angles.

The radar reflectivity varies depending on the surface feature: Volumetric scatterers like vegetation produce a very diffused echo. Walls of buildings and roads are relatively rough and produce a more pronounced, but still diffuse echo. The sharpest radar echos are generated by metallic objects and shapes that are similar to retroreflectors, like balconies on buildings [ZB10, Fig. 2]. These sharp reflections are best suited to reconstruct three-dimensional objects like buildings. The processed SAR data is similar to an image that a hypothetical radar camera would take when pointed at earth's surface [KG20]. The goal of SAR tomography is to reconstruct 3D information from parts, or even single pixels of the two-dimensional "image". To do this, the data from multiple images taken from slightly different positions are combined. This is done mathematically via a continuous Fourier transform with discrete frequencies and can be brought into the form of a linear system of equations

$$g = R\gamma.$$

Each element of g is the same "pixel" viewed from one of n different positions. R, the matrix transforming the data, is defined entry wise by $R_{jk} = \exp(-2\pi i \xi_j s_k)$. In this formula, s is the elevation and ξ_j is a quantity related to the position of the satellite. The solution vector γ contains the backscatter intensity for each discrete height step. This data can be visualized (see for example [SZYB18, Fig. 5]) or used in further calculations. To obtain γ, this system of linear equations needs to be solved. To evaluate the prospects of using HHL for this problem, we start by laying out a few key characteristics:

1) The matrix is non-square: We have $R \in \mathbb{C}^{n \times l}$, where n is the dimension of g and l is the dimension of γ. n is much larger than l.

2) The matrix R is hence not hermitian

3) γ should be as sparse as possible

4) The matrix R is not sparse

5) The condition of R is bad

6) g and R need to be prepared from classical data

1), 2), 3): As discussed in Section 5.2.1, non-square matrices are not a problem in principle. R has a typical matrix size of ca. 100×10^6 million [QWSZ22]. This makes the system extremely underdetermined, but of the vast solution space, one is generally only interested in γ that have few non-zero entries. This additional constraint is hard to model in HHL, but the Moore-Penrose Pseudoinverse at least returns the solution for which $\|\gamma\|_2$ is the smallest. Depending on the linear system, this could be close to the desired solution.

4), 5): The next two conditions are harder to deal with: In practice, very similar positions of the satellite cause some rows of R to be very similar, yielding a bad condition. Typical values of κ range from 10^6 to 10^{16}. Also, the matrix is not sparse at all with $s \approx n$. For standard HHL, this destroys the exponential speedup, even if e.g. the dependence on s is reduced to s instead of s^2 through improved Hamiltonian simulation.

If one does not care about the parts that are contributed to the solution by the ill-conditioned parts of R, there could be a way to achieve a good runtime despite a bad condition number κ. The complex version of HHL presented in [HHL09, Appendix A, Sec. 1] is able to "discard" bad eigenvalues, if the algorithm is configured with a κ' that is smaller (better) than the actual κ. In the eigenvalue inversion step, special filter functions remove any eigenvalue $\lambda < \frac{1}{\kappa}$. The hermitian embedding of R complicates the effect of this procedure, but it might be similar to removing one of the near-equal rows of R. These rows might not be needed anyway.

If this trick is not feasible, the most promising result is a method for exponentiating dense and low-rank matrices [RSML18], which is also discussed in Section 4.8. One of its limitations would be no problem in this application: The complexity depends on $\|A\|_{\max}$, the maximum absolute value found in entries of A. But all entries of R are of the form $e^{i \cdot \text{something}}$ which has an absolute value of 1. Still, there are problems: As this result regards only the phase estimation, the dependency on κ from the remaining part of HHL would be still present. Also, actually simulated is only $e^{iA \cdot 1/n}$, which could pose problems.

6): There is currently no sufficiently fast method for preparing g from classical input (see Sections 4.3 and 5.2.1). If there is no method found for this in the future, this problem could be avoided by integrating the radar hardware on the satellite into a quantum device and storing the values directly as quantum amplitudes. The matrix R follows a quite regular pattern, maybe parts of it could be calculated on the fly on the quantum computer, reducing the need for a large quantum RAM.

Finally, the matrix basically performs an intricate version of the Fourier transform, there might be a solution possible without using HHL by adapting the QFT algorithm instead. This would remove the constraints of HHL, still pose the problem of input preparation. The problems regarding readout (outlined in Section 4.4) do not apply, however: Each solution

vector entry describes the backscatter intensity at a given height. If one is only interested in the position of these few non-zero entries with high backscatter, very few measurements could suffice.

The findings on the applicability of different versions of HHL are summarized in Table 6.1. As a general caveat: The table only regards asymptotics, but in reality the matrices in SAR tomography do not get arbitrarily large due to limited physical resolution. So constants could play a role, as well.

6.2 Machine Learning

Solving linear systems is also of great importance in various machine learning settings, and HHL enables the speedup of a large portion of quantum machine learning algorithms. Examples include classification with support vector machines or linear regression. As this is not within the scope of this work, we refer to [DYY+20] or [BWP+17, Box 1], which give a good overview of the topic, and some results on quantum machine learning that build on HHL [SSP16] [WBL12].

6.3 HHL Implementations

When assessing HHL for practical applications, a working implementation is a useful tool to check hypotheses and discover more subtle limitations. However, current quantum hardware is severely limited in the number of available qubits. This poses a challenge, since general HHL implementations require a significant amount even for small n. One factor for this is the precision of QPE, which needs additional qubits for cases when the eigenvalues λ_j cannot exactly be described by d bits. Another factor is the QRAM or similar structure that would be needed to realistically implement the oracle for A in the Hamiltonian simulation. Finally, the inversion and application of trigonometric functions like arcsin is also hard to do on less than e.g. 5 qubits without incurring significant error.

For these reasons, current implementations targeting quantum hardware are carefully crafted for a single matrix A. Individual steps of Hamiltonian simulation and eigenvalue inversion are combined into single operations which have been calculated classically beforehand. A selection of these implementations is presented here:

- [CDFK12] give a general HHL circuit overview and describe a concrete simulation for

$$A = \tfrac{1}{2}\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}.$$

- [CEP+18, Sec. 4] present a HHL run for the same matrix A and give concrete gates that work in the `ibmqx4` architecture.

- Another implementation of HHL for the matrix A is given by [ZSC+17], with a focus on the physics of the superconducting qubit hardware architecture used by IBM.

- [DYY+20, Sec. 3.3] present gates and pseudocode for implementing A, while also giving an overview of applications in machine learning.

				Improvements to Hamiltonian Simulation				
	Classic HHL	Precon-ditioning	Dense Adaption	Trotteri-zation	Quantum Walks	Theoretical Optimum	Fourier series	Dense, Low-rank
Application: SAR	✘	✘	✘	✘	✘	✘	✘	(✔)
b is classic, non-uniform	✘	✘	✘	✘	✘	✘	✘	✘
A has bad condition	✘[1]	(✔)	✘[1]	✘[1]	✘[1]	✘[1]	✘[1]	(✔)
A is dense	✘	✘	(✔)	✘	✘	✘	✘	✔
A a is non-hermitian	✔	✔	✔	✔	✔	✔	✔	✔
x should be sparse	(✔)	(✔)	(✔)	(✔)	(✔)	(✔)	(✔)	(✔)

Table 6.1: Applicability of different HHL version on the SAR tomography problem.

[a] As discussed, a bad condition number might be no problem if similar rows can be discarded by running HHL with a smaller condition number.

- [MJW21] give a very detailed overview of all the unitary matrices and quantum states that occur during an execution of HHL for the matrix

$$B = \frac{1}{3}\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}.$$

  A gate-based description for an implementation on IBM's quantum system is also provided.

- The same matrix B is used in the introduction of IBM Qiskit at https://qiskit.org/textbook/ch-applications/hhl_tutorial.html.

A more general implementation of HHL is provided by IBM Qiskit as a library function.[1] The accompanying paper is [VHW22] and describes the method used. The efficient implementation is made possible by a comparatively strict additional constraint on A, which is assumed to be tridiagonal Toeplitz. This means that A is of the form

$$T = \begin{pmatrix} a & c & & 0 \\ b & \ddots & \ddots & \\ & \ddots & \ddots & c \\ 0 & & b & a \end{pmatrix}$$

with $a, b, c \in \mathbb{C}$. This type of matrix appears occasionally in numerical mathematics, for example when interpolating with cubic splines. Its simple structure allows for an efficient and concrete implementation of the oracles needed in Hamiltonian simulation, enabling the complete implementation of HHL.

A more abstract publication about a concrete gate-based implementation is [CMN+18], which compares the different improvements of HHL with respect to projected gate complexity and other potential problems that could arise when implementing HHL on noisy intermediate-scale quantum computers (NISQ). For example, the theoretically best version of Hamiltonian simulation [LC19] needs a large amount of gates and ancilla qubits to implement trigonometric functions and perform the quantum walk. Although this plays no role in the asymptotic performance, the overhead for NISQ implementation is considerable. See also the accompanying talk to the paper by Childs [Chi17].

[1] See https://qiskit.org/documentation/stubs/qiskit.algorithms.linear_solvers.HHL.html for the documentation and https://qiskit.org/documentation/_modules/qiskit/algorithms/linear_solvers/hhl.html#HHL for the source code.

7 Summary and Outlook

In this work, the opportunities and drawbacks of HHL with respect to possible applications were discussed. As many of the constraints have their roots in certain subroutines, not only the idea of the algorithm, but also each of its subroutines were introduced in detail. The knowledge of these algorithms and their interactions, like the quantum Fourier transform with the phase estimation, provides the tool set for drawing well-founded conclusion when investigating the parameters of HHL.

HHL and its applications aside, the detailed approach also acts as a reference for the subroutines themselves. Compared to the more abstract definitions and toy examples often employed in introductory books, seeing how the algorithms are used "in action" can be of value. Furthermore, a comprehensive overview on the current research on HHL was provided, giving an idea for what might and might not be possible in the future.

The properties, constraints, and improvements of HHL form a large set of information. To help with their assessment for general applications, they were assembled in a decision matrix. The corresponding complexities were also incorporated. Data processing in SAR tomography at DLR was introduced as a concrete example and its core properties discussed. These were then put against the developed matrix to evaluate the general applicability, and some more subtle aspects were covered in more detail.

The result was that even with a capable quantum computer, there are some significant hurdles in the way. Apart from that, SAR tomography could potentially work with an adapted version of HHL. Roughly the same conclusion holds for general applications for solving linear systems. The hurdles that were discovered represent opportunities for further research at the same time.

The first main issue is relevant to any quantum algorithm, especially if it is used as a subroutine of a classical computation: All the arguments must be encoded as a quantum state, and after completion, the return values must be decoded. Especially for encoding, there are no established and practical methods that would allow to prepare an arbitrary quantum state, yet. There are quite concrete proposals for how such a quantum RAM could be theoretically implemented, though.

Nonetheless, any such method would require that the data is already prepared in classical RAM. This preparation takes $\mathcal{O}(n)$ steps for a vector with n elements by definition. Any quantum algorithm that achieves a runtime faster than $\mathcal{O}(n)$ with an n-element vector as an argument will have the runtime capped to $\mathcal{O}(n)$ as a consequence. This is true for HHL and essentially rules out such an application.

The most obvious solution is to use HHL as a subroutine for quantum algorithms; quantum machine learning presents such a case. But even as a subroutine to classical algorithms, there are some situations where the speedup would not be destroyed: For example, if the classical data is already present, like in a database, no concrete cost is associated with its generation. Also, the same classical data might be used as an input to many quantum algorithms, increasing the overall efficiency.

For decoding, saving the result to classical RAM also takes $\mathcal{O}(n)$ steps, but one might only be interested in the positions of nonzero elements, the value of the first element, or the result of some scalar product with the solution vector. In all these cases, much fewer steps are required.

For the SAR tomography in particular, the linear system is very underdetermined, and of the many possible solutions, the most sparse vector x is desired. This requirement cannot be directly mapped into the linear system, but HHL returns the Moore-Penrose pseudoinverse A^+ applied to b in the underdetermined case. This is equivalent to the solution with the smallest norm $\|x\|_2$, which is related, but not quite the same. Depending on how the vectors x look like, it might still be close enough; this could be explored in further research.

Also, the matrix is of bad condition, which generally destroys the speedup of HHL. However, as discussed in Section 6.1, HHL can also run with a better (but false) condition number as parameter, which then returns only a part of the solution vector where any values contributed from ill-conditioned parts of A are missing. When these contributions are irrelevant anyway, HHL can still be used. When they are actually unwanted, HHL might even pose an improvement. Further research could find out if any of these cases apply in SAR tomography.

List of Figures

Bibliography

[Aar15] Aaronson, Scott: Read the fine print. In: *Nature Physics* 11 (2015), Nr. 4, S. 291–293

[Aho04] Ahokas, Graeme R.: *Improved algorithms for approximate quantum Fourier transforms and sparse Hamiltonian simulations*, University of Calgary, Diplomarbeit, 2004

[Amb17] Ambainis, Andris: Understanding Quantum Algorithms via Query Complexity. In: *CoRR* (2017)

[BACS07] Berry, Dominic W. ; Ahokas, Graeme ; Cleve, Richard ; Sanders, Barry C.: Efficient quantum algorithms for simulating sparse Hamiltonians. In: *Communications in Mathematical Physics* 270 (2007), Nr. 2, S. 359–371

[BC12] Berry, Dominic W. ; Childs, Andrew M.: Black-box hamiltonian simulation and unitary implementation. In: *Quantum Inf. Comput.* 12 (2012), Nr. 1-2, S. 29–62

[BCC+14] Berry, Dominic W. ; Childs, Andrew M. ; Cleve, Richard ; Kothari, Robin ; Somma, Rolando D.: Exponential improvement in precision for simulating sparse Hamiltonians. In: Shmoys, David B. (Hrsg.): *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, ACM, 2014, S. 283–292

[BCC+15] Berry, Dominic W. ; Childs, Andrew M. ; Cleve, Richard ; Kothari, Robin ; Somma, Rolando D.: Simulating Hamiltonian dynamics with a truncated Taylor series. In: *Physical review letters* 114 (2015), Nr. 9, S. 090502

[BCK15] Berry, Dominic W. ; Childs, Andrew M. ; Kothari, Robin: Hamiltonian Simulation with Nearly Optimal Dependence on all Parameters. In: Guruswami, Venkatesan (Hrsg.): *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, IEEE Computer Society, 2015, S. 792–809

[BHMT02] Brassard, Gilles ; Hoyer, Peter ; Mosca, Michele ; Tapp, Alain: Quantum amplitude amplification and estimation. In: *Contemporary Mathematics* 305 (2002), S. 53–74

[BHT98] Brassard, Gilles ; Høyer, Peter ; Tapp, Alain: Quantum counting. In: *International Colloquium on Automata, Languages, and Programming* Springer, 1998, S. 820–831

[BPLC+19] Bravo-Prieto, Carlos ; LaRose, Ryan ; Cerezo, Marco ; Subasi, Yigit ; Cincio, Lukasz ; Coles, Patrick J.: Variational quantum linear solver. In: *arXiv preprint arXiv:1909.05820* (2019)

[BWP+17] Biamonte, Jacob ; Wittek, Peter ; Pancotti, Nicola ; Rebentrost, Patrick ; Wiebe, Nathan ; Lloyd, Seth: Quantum machine learning. In: *Nature* 549 (2017), Nr. 7671, S. 195–202

[CaeBC] Caesar, Gaius J.: *Comentarii De Bello Gallico*. 51BC

[CDFK12] Cao, Yudong ; Daskin, Anmer ; Frankel, Steven ; Kais, Sabre: Quantum circuit design for solving linear systems of equations. In: *Molecular Physics* 110 (2012), Nr. 15-16, S. 1675–1680

[CEP+18] Coles, Patrick J. ; Eidenbenz, Stephan J. ; Pakin, Scott ; Adedoyin, Adetokunbo ; Ambrosiano, John u. a.: Quantum Algorithm Implementations for Beginners. In: *CoRR* (2018)

[Chi10] Childs, Andrew M.: On the relationship between continuous-and discrete-time quantum walk. In: *Communications in Mathematical Physics* 294 (2010), Nr. 2, S. 581–603

[Chi17] Childs, Andrew M.: *Toward the first quantum simulation with quantum speedup.* `https://www.youtube.com/watch?v=VSTzptzsNp0`. Version: 2017. – IBM ThinkQ Conference

[Chi21] Childs, Andrew M.: Lecture notes on quantum algorithms. In: *Lecture notes at University of Maryland* (2021). `http://www.cs.umd.edu/~amchilds/qa/qa.pdf`

[CJS13] Clader, B D. ; Jacobs, Bryan C. ; Sprouse, Chad R.: Preconditioned quantum linear system algorithm. In: *Physical review letters* 110 (2013), Nr. 25, S. 250504

[CK10] Childs, Andrew M. ; Kothari, Robin: Simulating Sparse Hamiltonians with Star Decompositions. In: Dam, Wim van (Hrsg.) ; Kendon, Vivien M. (Hrsg.) ; Severini, Simone (Hrsg.): *Theory of Quantum Computation, Communication, and Cryptography - 5th Conference, TQC 2010, Leeds, UK, April 13-15, 2010, Revised Selected Papers* Bd. 6519, Springer, 2010 (Lecture Notes in Computer Science), S. 94–103

[CKS17] Childs, Andrew M. ; Kothari, Robin ; Somma, Rolando D.: Quantum Algorithm for Systems of Linear Equations with Exponentially Improved Dependence on Precision. In: *SIAM J. Comput.* 46 (2017), Nr. 6, S. 1920–1950

[CMN+18] Childs, Andrew M. ; Maslov, Dmitri ; Nam, Yun S. ; Ross, Neil J. ; Su, Yuan: Toward the first quantum simulation with quantum speedup. In: *Proc. Natl. Acad. Sci. USA* 115 (2018), Nr. 38, S. 9456–9461

[CPF+10] Cramer, Marcus ; Plenio, Martin B. ; Flammia, Steven T. ; Somma, Rolando ; Gross, David ; Bartlett, Stephen D. ; Landon-Cardinal, Olivier ; Poulin, David ; Liu, Yi-Kai: Efficient quantum state tomography. In: *Nature communications* 1 (2010), Nr. 1, S. 1–7

[CST+21] Childs, Andrew M. ; Su, Yuan ; Tran, Minh C. ; Wiebe, Nathan ; Zhu, Shuchen: Theory of trotter error with commutator scaling. In: *Physical Review X* 11 (2021), Nr. 1, S. 011020

[DHM+18] Dervovic, Danial ; Herbster, Mark ; Mountney, Peter ; Severini, Simone ; Usher, Naïri ; Wossnig, Leonard: Quantum linear systems algorithms: a primer. In: *CoRR* (2018)

[DVDAPDS20] De Veras, Tiago M. ; De Araujo, Ismael C. ; Park, Daniel K. ; Da Silva, Adenilton J.: Circuit-based quantum random access memory for classical data with continuous amplitudes. In: *IEEE Transactions on Computers* 70 (2020), Nr. 12, S. 2125–2135

[DYY+20] Duan, Bojia ; Yuan, Jiabin ; Yu, Chao-Hua ; Huang, Jianbang ; Hsieh, Chang-Yu: A survey on HHL algorithm: From theory to application in quantum machine learning. In: *Physics Letters A* 384 (2020), Nr. 24, S. 126595

[GLF+10] Gross, David ; Liu, Yi-Kai ; Flammia, Steven T. ; Becker, Stephen ; Eisert, Jens: Quantum state tomography via compressed sensing. In: *Physical review letters* 105 (2010), Nr. 15, S. 150401

[GLM08] Giovannetti, Vittorio ; Lloyd, Seth ; Maccone, Lorenzo: Architectures for a quantum random access memory. In: *Physical Review A* 78 (2008), Nr. 5, S. 052310

[Gol12] Golan, Jonathan S.: Moore–penrose pseudoinverses. In: *The linear algebra a beginning graduate student ought to know*. Springer, 2012, S. 441–452

[Gro96] Grover, Lov K.: A fast quantum mechanical algorithm for database search. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, S. 212–219

[Han21] Hann, Connor T.: *Practicality of Quantum Random Access Memory*, Yale University, Diss., 2021

[HHL09] Harrow, Aram W. ; Hassidim, Avinatan ; Lloyd, Seth: Quantum algorithm for linear systems of equations. In: *Physical review letters* 103 (2009), Nr. 15, S. 150502

[Hom05] Homeister, Matthias: *Quantum Computing verstehen - Grundlagen, Anwendungen, Perspektiven*. Vieweg, 2005 (Computational intelligence). – ISBN 978–3–528–05921–7

[IRM+19] Iten, Raban ; Reardon-Smith, Oliver ; Mondada, Luca ; Redmond, Ethan ; Kohli, Ravjot S. ; Colbeck, Roger: Introduction to UniversalQ-Compiler. In: *CoRR* (2019)

[Kem03] Kempe, Julia: Quantum random walks: an introductory overview. In: *Contemporary Physics* 44 (2003), Nr. 4, S. 307–327

[KG20] Khoshnevis, Seyed A. ; Ghorshi, Seyed: A tutorial on tomographic synthetic aperture radar methods. In: *SN Applied Sciences* 2 (2020), Nr. 9, S. 1–14

[Kit96] Kitaev, Alexei Y.: Quantum measurements and the Abelian Stabilizer Problem. In: *Electron. Colloquium Comput. Complex.* (1996), Nr. 3

[Kot14] Kothari, Robin: *Efficient algorithms in quantum query complexity*, University of Waterloo, Diss., 2014

[Kot17] Kothari, Robin: *Quantum algorithms for Hamiltonian simulation: Recent results and open problems.* `https://www.youtube.com/watch?v=PerdRJ-offU`. Version: 2017. – IBM ThinkQ Conference

[KP16] Kerenidis, Iordanis ; Prakash, Anupam: Quantum recommendation systems. In: *arXiv preprint arXiv:1603.08675* (2016)

[LC17] Low, Guang H. ; Chuang, Isaac L.: Optimal Hamiltonian simulation by quantum signal processing. In: *Physical review letters* 118 (2017), Nr. 1, S. 010501

[LC19] Low, Guang H. ; Chuang, Isaac L.: Hamiltonian simulation by qubitization. In: *Quantum* 3 (2019), S. 163

[MJW21] Morrell Jr, Hector J. ; Wong, Hiu Y.: Step-by-Step HHL Algorithm Walkthrough to Enhance the Understanding of Critical Quantum Computing Concepts. In: *arXiv preprint arXiv:2108.09004* (2021)

[NC16] Nielsen, Michael A. ; Chuang, Isaac L.: *Quantum Computation and Quantum Information (10th Anniversary edition).* Cambridge University Press, 2016. – ISBN 978–1–10–700217–3

[PPR19] Park, Daniel K. ; Petruccione, Francesco ; Rhee, June-Koo K.: Circuit-based quantum random access memory for classical data. In: *Scientific reports* 9 (2019), Nr. 1, S. 1–8

[QWSZ22] Qian, Kun ; Wang, Yuanyuan ; Shi, Yilei ; Zhu, Xiao X.: γ-Net: Super-resolving SAR Tomographic Inversion via Deep Learning. In: *IEEE Trans. Geosci. Remote. Sens.* 60 (2022), S. 1–16

[RSML18] Rebentrost, Patrick ; Steffens, Adrian ; Marvian, Iman ; Lloyd, Seth: Quantum singular-value decomposition of nonsparse low-rank matrices. In: *Physical review A* 97 (2018), Nr. 1, S. 012327

[Sha18] Shao, Changpeng: Reconsider HHL algorithm and its related quantum machine learning algorithms. In: *arXiv preprint arXiv:1803.01486* (2018)

[She94] Shewchuk, Jonathan R.: *An introduction to the conjugate gradient method without the agonizing pain.* 1994

[Sho94] Shor, Peter W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, IEEE Computer Society, 1994, S. 124–134

[SSP16] Schuld, Maria ; Sinayskiy, Ilya ; Petruccione, Francesco: Prediction by linear regression on a quantum computer. In: *Physical Review A* 94 (2016), Nr. 2, S. 022342

[Suz90] Suzuki, Masuo: Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations. In: *Physics Letters A* 146 (1990), Nr. 6, S. 319–323

[SZYB18] Shi, Yilei ; Zhu, Xiao X. ; Yin, Wotao ; Bamler, Richard: A Fast and Accurate Basis Pursuit Denoising Algorithm With Application to Super-Resolving Tomographic SAR. In: *IEEE Trans. Geosci. Remote. Sens.* 56 (2018), Nr. 10, S. 6148–6158

[TMVH18] Thapliyal, Himanshu ; Muñoz-Coreas, Edgard ; Varun, T. S. S. ; Humble, Travis S.: Quantum Circuit Designs of Integer Division Optimizing T-count and T-depth. In: *CoRR* (2018)

[UG26] Uhlenbeck, George E. ; Goudsmit, Samuel: Spinning electrons and the structure of spectra. In: *Nature* 117 (1926), Nr. 2938, S. 264–265

[VHW22] Vazquez, Almudena C. ; Hiptmair, Ralf ; Woerner, Stefan: Enhancing the quantum linear systems algorithm using Richardson extrapolation. In: *ACM Transactions on Quantum Computing* 3 (2022), Nr. 1, S. 1–37

[WBL12] Wiebe, Nathan ; Braun, Daniel ; Lloyd, Seth: Quantum algorithm for data fitting. In: *Physical review letters* 109 (2012), Nr. 5, S. 050505

[WZP18] Wossnig, Leonard ; Zhao, Zhikuan ; Prakash, Anupam: Quantum linear system algorithm for dense matrices. In: *Physical review letters* 120 (2018), Nr. 5, S. 050502

[ZB10] Zhu, Xiao X. ; Bamler, Richard: Tomographic SAR inversion by $L\_\{1\}$-norm regularization—The compressive sensing approach. In: *IEEE transactions on Geoscience and Remote Sensing* 48 (2010), Nr. 10, S. 3839–3846

[ZFR+21] Zhao, Zhikuan ; Fitzsimons, Jack K. ; Rebentrost, Patrick ; Dunjko, Vedran ; Fitzsimons, Joseph F.: Smooth input preparation for quantum and quantum-inspired machine learning. In: *Quantum Machine Intelligence* 3 (2021), Nr. 1, S. 1–6

[ZSC+17] Zheng, Yarui ; Song, Chao ; Chen, Ming-Cheng ; Xia, Benxiang ; Liu, Wuxin ; Guo, Qiujiang ; Zhang, Libo ; Xu, Da ; Deng, Hui ; Huang, Keqiang u. a.: Solving systems of linear equations with a superconducting quantum processor. In: *Physical review letters* 118 (2017), Nr. 21, S. 210504