Faktorisierung auf dem Quantencomputer Erklärung und Implementierung

Einführung in die Funktionsweise von Quantencomputern und Implementation zweier Programmbibliotheken in Q# mit arithmetischen Operationen sowie einer vollständigen Implementierung des Shor-Algorithmus

Maturaarbeit von	Joël Benjamin Huber
Betreut von	Christian Steiger
an der	Kantonsschule Freudenberg Zürich
abgegeben am	15. Dezember 2020

Zusammenfassung: *Die vorliegende Maturitätsarbeit beschäftgt sich mit Quantencomputern und ihrer Anwendung für das Faktorisierungsproblem. Zu diesem Zweck wurden arithmetische Operationen genauer betrachtet, um zu verstehen, wie sie auf Quantencomputern implementiert werden können. Danach wurde der Shor-Algorithmus genauer betrachtet, mit welchem sich Zahlen faktorisieren lassen. Dies alles wurde in der Form zweier Programmbibliotheken in der Quantenprogrammiersprache Q# und der klassischen Programmiersprache C# implementiert, sodass man diese Operationen auch in anderen Projekten verwenden kann.*

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Vorwort

Das Gebiet der Quantencomputer ist zweifellos eines der vielversprechendsten Forschungsgebiete. Quantencomputer würden uns theoretisch erlauben, Berechnungen auszuführen, die auf klassischen Computern nicht effizient ausgeführt werden können. Diese mathematischen Konstrukte werden in der Zukunft wahrscheinlich eine grössere Rolle spielen.

Immer wieder liest man in Zeitungs- und Journalartikeln über Relisierungen von Quantencomputern und immer wieder wurde von Durchbrüchen gesprochen. So wie zum Beispiel, als Google 2019 einen Artikel im Journal "Nature" publizierte, in welchem sie behaupteten, die "Quantenüberlegenheit" erreicht zu haben ([1]), was gleich darauf von der IBM in ihrem Blog angezweifelt wurde ([11]).

Aus diesem Grund ist es meiner Meinung nach ein guter Zeitpunkt, um sich vertieft mit dem Thema auseinanderzusetzen. Ich habe mich deshalb entschieden, mich auf diese Reise zu begeben und im Rahmen meiner Maturitätsarbeit zu versuchen zu verstehen, wie Quantencomputer funktionieren, um danach in einem nächsten Schritt arithmetische Operationen sowie den Shor-Algorithmus zu implementieren. Die grösste Schwierigkeit, mit der ich mich auf meinem Weg konfrontiert sah, war es, die Quantenalgorithmen zu debuggen.

Doch es dauerte immer eine Weile, bis etwas funktionierte. In der Tat denke ich, dass die grösste Schwierigkeit, die ich hatte, war, die Quantenalgorithmen zu debuggen. Mein Code war am Anfang noch sehr fehlerhaft und diese Fehler zu finden war schwierig. Dies liegt auch daran, dass Qubits schwierigere Konstrukte sind, so dass deren momentanen mit deren geplanten Zuständen zu vergleichen und zu finden, wieso etwas nicht stimmt, Stunden in Anspruch nehmen konnte.

In der Tat ist es meiner Meinung nach nicht sehr einfach, in dieses Gebiet einzusteigen. Auch deshalb soll meine Arbeit dazu dienen, zusammen mit der angegebenen Fachliteratur, den Einstieg zu vereinfachen. Dafür werden aber Kentnisse der linearen Algebra vorausgesetzt, da diese sehr wichtig ist, um die mathematische Struktur zu verstehen. Zum Einstieg in die lineare Algebra habe ich selbst das Buch "Lineare Algebra" von Gerd Fischer (siehe [6]) gelesen und empfehle dieses Buch, um sich die Grundlagen der linearen Algebra zu erarbeiten.

Ich habe mich sowohl damit auseinandergesetzt, wie arithmetische Operationen auf Quantencomputern programmiert werden können, als auch, wie man damit Zahlen faktorisieren kann. Was ich gelernt habe, habe ich in Code umgesetzt. Ich habe zwei Programmbibliotheken implementiert, die ich in Abschnitt 1.2 genauer beschreibe und in welchen man alle in dieser Arbeit betrachteten Operationen als Code finden kann. Zudem habe ich bei der Implementation ge-

zielt darauf geachtet, dass diese Bibliotheken auch in späteren Projekten gut verwendet werden können und ich habe vor, diese Bibliotheken in Zukunft kontinuierlich zu erweitern und Implementationen anderer Quantenalgorithmen hinzuzufügen, so dass sie aktuell bleiben.

1.2 Die QInteger- und die QAlgorithm-Libraries und eine Implementation des Faktorisierungsalgorithmus

Das Produkt dieser Maturitätsarbeit sind zwei Programmbibliotheken, welche Algorithmen für Quantencomputer bereitstellen. Diese Programmbibliotheken sind in der Programmiersprache Q# geschrieben, einer Quantenprogrammiersprache entwickelt von Microsoft. Das Quantum Development Kit (QDK) von Microsoft beinhaltet dabei nicht nur die Sprache Q#, sondern auch einen Simulator, auf welchem man den Code ausführen kann. Gleichzeitig ist es das Ziel der Sprache, dass man sie auf echten Quantencomputern einsetzen kann, sobald diese genug weit entwickelt sein werden.

Die beiden Bibliotheken sollen mathematische Operationen auf Quantencomputern vereinfachen. Hier ein kurzer Überblick über ihre Funktionen:

- Die *QInteger-Library* definiert mit dem Typ "QInt" die grundlegende Datenstruktur "Zahl" auf einem Quantencomputer. Zudem stellt sie arithmetische Operationen und nützliche Funktionen für den Typ "QInt" bereit.
- In der *QAlgorithms-Library* sind Algorithmen für Quantencomputer implementiert, darunter auch der quantenbasierte Teil von Shors Algorithmus zur Faktorisierung von Zahlen, welcher zweifelsohne einer der nennenswertesten Errungenschaften auf dem Gebiets der Quantencomputer ist.

Diese beiden Bibliotheken stellen generelle, nützliche Funktionen für Quantencomputer bereit, welche auch zur Vereinfachung in anderen Projekten gebraucht werden können. Den Code zu diesen Programmbibliotheken kann man auf GitHub finden, wo er in Zukunft einige Aktualisierungen erhalten wird. Auf GitHub ist er unter folgendem Link verfügbar:

https://github.com/johutha/QInteger-QAlgorithms

Ich habe Q# gewählt, da es aktuell eine der populärsten Quantenprogrammiersprachen ist, über eine gute Dokumentation verfügt, gut unterstützt und regelmässig aktualisiert wird. Zudem kann der Compiler automatisch zu einer Quantenoperation deren Inverses oder deren kontrollierte Version generieren [1], was den Code kürzer und übersichtlicher macht.

Nebst diesen beiden Bibliotheken finden sich im GitHub-Repository weitere Projekte. In einem dieser Projekte ist der "Factorizer" implementiert, welcher zusammen mit den Quanten-Programmbibliotheken eine komplette Implementierung von Shors Algorithmus bilden. Dazu sind im Projekt auch andere Faktorisierungsalgorithmen programmiert, so dass man diese miteinander vergleichen kann.

Im Weiteren befindet sich im Repository ein Projekt für eine einfache Konsolen-Applikation, welche den "Factorizer" benüützt, so dass man diesen als Proof of Concept ausprobieren kann. Zudem gibt es ein Projekt für einen Zeitmesser, welcher misst, wie lange der "Factorizer " benötigt.

[1] Siehe auch Kapitel 2.2

Zudem befinden sich im Repository drei weitere Projekte, in welchen Unit-Tests implementiert sind. Unit-Tests testen die verschiedenen Komponenten einzeln, wodurch man einfacher Implementationsfehler lokalisieren und Bugs finden kann.

Kapitel 2

Grundlagen

2.1 Lineare Algebra

Um mit Quantencomputern arbeiten zu können, braucht es Kenntnisse der linearen Algebra. Operationen, die man auf Quantencomputern implementiert, sind lineare Operationen auf Qubits. Ich werde an dieser Stelle eine kurze Zusammenfassung der nötigen Grundlagen geben, welche erforderlich sind, um die Quanten-Grundlagen und Shors Algorithmus zu verstehen. Dies wird jedoch nur eine kurze Zusammenfassung und keine Einführung in dieses Gebiet sein. Für eine Einführung verweise ich auf [6].

Vor der Zusammenfassung möchte ich kurz die in der Quantenmechanik gebräuchliche Notation für lineare Algebra einführen. Diese nennt sich Dirac- oder Bra-Ket-Notation.

2.1.1 Schreibweise

Einen Vektor schreibt man in der Quantenmechanik als $|\varphi\rangle$. Diese besondere Art von Klammern wird als *ket* bezeichnet. Für einen Vektor $|\varphi\rangle$ wird der dazugehörige duale Vektor als $\langle\phi|$ bezeichnet. Diese zweite Klammer heisst *bra*, sodass die beiden Klammern zusammen ein Bra-Ket bildet, was vom englischen Wort bracket abstammt. Somit lässt sich das Skalarprodukt von $|\psi\rangle$ und $|\varphi\rangle$ als $\langle\psi|\varphi\rangle$ darstellen.

2.1.2 Vektorräume

Um Quantensysteme mathematisch beschreiben zu können, brauchen wir Vektorräume. Genauer benötigen wir endlichdimensionale Vektorräume über $\mathbb{C}$, zusammen mit einem Skalarprodukt. Diese Vektorräume werden auch unitäre Vektorräume[1] genannt. Diese sind isomorph zu $\mathbb{C}^n$.

Sei V ein solcher Vektorraum.

Wir nennen eine Menge von Vektoren in V ein *Erzeugendensystem*, falls jeder Vektor in V als eine Linearkombination der Vektoren in jener Menge geschrieben werden kann.

Eine Menge von Vektoren $|v_0\rangle, |v_1\rangle, ... |v_{k-1}\rangle$ ist *linear unabhängig*, falls wenn aus der Gleichung $a_0 |v_0\rangle + a_1 |v_1\rangle + ... + a_{k-1} |v_{k-1}\rangle = 0$ folgt, dass $a_0 = a_1 = ... = a_{k-1} = 0$ gilt. Dabei

[1] Im Allgemeinen braucht es sogenannte Hilberträume. Da man es im Zusammenhang mit Quantencomputern aber nur mit endlichdimensionalen Räumen zu tun hat, reichen unitäre Vektorräume

sind die Koeffizienten $a_0, a_1, ..., a_{n-1}$ komplexe Zahlen. Diese Aussage ist äquivalent zur Aussage, dass sich keiner der Vektoren $|v_i\rangle$ als Linearkombination der anderen Vektoren in der Menge darstellen lässt.

Eine *Basis* ist ein linear unabhängiges Erzeugendensystem.

2.1.3 Inneres Produkt

Die für die Beschreibung der Qubits benötigten unitären Vektorräumen sind mit einem Skalarprodukt ausgerüstet: Das heisst, es gibt eine Funktion $(\cdot,\cdot) : V \times V \to C$ mit folgenden Eigenschaften:

1. Linear im zweiten Argument:
$$\sum_j \lambda_j \left(|\psi\rangle\,,\ |\varphi_j\rangle\right) = \left(|\psi\rangle\,,\ \sum_j \lambda_j\,|\varphi_j\rangle\right)$$
2. Hermitesch:
$$\left(|\psi\rangle\,,\ |\varphi\rangle\right) = \overline{\left(|\varphi\rangle\,,\ |\psi\rangle\right)}$$
3. Positiv definit:
$$\varphi \neq 0 \Rightarrow \left(|\varphi\rangle\,,\ |\varphi\rangle\right) > 0$$

Wie oben schon angetönt, schreibt man dieses Produkt in der quantenmechanischen Notation als $\langle\psi|\varphi\rangle$. Ich habe für diese drei Bedingungen jedoch die $(\cdot,\cdot)$-Schreibweise verwendet, da man mit ihr die Bedingungen übersichtlicher darstellen kann.

Die *Norm* eines Vektors $|\varphi\rangle$, wird als $\||\varphi\rangle\|$ geschrieben und ist definiert als $\||\varphi\rangle\| = \sqrt{\langle\varphi|\varphi\rangle}$. Ein Vektor $|\varphi\rangle$ ist normiert, falls $\||\varphi\rangle\| = 1$ gilt. Wir benutzen hier das Standardskalarprodukt in $\mathbb{C}^n$ definiert als:
$$\begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} := \sum_{i=1}^{n} \overline{c_i} w_i$$

Es ist nicht schwierig zu sehen, dass das auf diese Weise definierte Skalarprodukt die oben genannten Bedingungen erfüllt.

2.1.4 Lineare Operatoren

Sei v ein endlichdimensionaler, unitärer Vektorraum[2], seien $|v\rangle$ und $|w\rangle \in V$ und $\lambda \in \mathbb{C}$.

Definition: Ein *linerarer Operator*[3] L ist eine Funktion
$$\begin{array}{rccc} L: & V & \longrightarrow & V \\ & v & \mapsto & Lv \end{array}$$
welcher die Bedingung der Linearität $L(\sum_i \lambda_i\,|v_i\rangle) = \sum_i \lambda_i L(|v_i\rangle)$ erfüllt. Für eine Matrix

[2]Die Definitionen dieses Abschnittes würden auch für allgemeinere Vektorräume funktionieren. Das ist aber für diese Arbeit nicht nötig.

[3]In dieser Arbeit wird der Begriff *linearer Operator* synonym zum Begriff *lineare Abbildung* benutzt

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$ mit Einträgen $a_{ij} \in \mathbb{C}$ schreiben wir kurz $A = (a_{ij})$ und definieren ...

die *konjugierte* Matrix von A: $\overline{A} := (\overline{a}_{ij})$

die *transponierte* Matrix von A: $A^T := (a_{ji})$

die *adjungierte* Matrix von A: $A^\dagger := \overline{A}^T$

Definition: Ein *unitärer Operator* U ist ein linearer Operator, welcher die Bedingung $U^\dagger U = I$. Aufgrund dieser Bedingungen erhalten unitäre Operationen das Skalarprodukt der Vektoren:

$$(U|u\rangle, U|v\rangle) = \langle u| U^\dagger U |v\rangle = \langle u|v\rangle$$

Somit verändert sich auch das Skalarprodukt eines Vektors mit sich selbst nicht, weshalb auch die Norm eines Vektors unter unitären Operatoren erhalten bleibt.

Zustände eines Quantensystems können als Vektoren eines unitären Vektorraumes V beschrieben werden. Zustandsänderungen werden durch unitäre Operationen beschrieben.

2.1.5 Eigenwerte und Eigenvektoren

Ein Eigenvektor $|\varphi\rangle$ zu einem linearen Operator U ist ein Vektor, sodass $U|\varphi\rangle = \lambda|\varphi\rangle$ gilt, wobei λ als der dazugehörige Eigenwert bezeichnet wird und eine komplexe Zahl ist. Bei einem unitären Operator müssen alle Eigenwerte einen Betrag von 1 haben, denn sonst würde sich die Norm des Vektors verändern. Der grosse Vorteil von Eigenvektoren ist, dass sie sich bis auf einen skalaren Faktor nicht verändern, wenn der dazugehörige Operator auf sie angewendet wird.

2.1.6 Das Tensorprodukt

Das Tensorprodukt, geschrieben mit dem Zeichen $\otimes$, gibt uns die Möglichkeit alle Möglichkeiten Kombinationen der Einträge ihrer Operanden zu generieren. Sei $A := (a_{ij})$ eine $n_a \times m_a$ Matrix und $B := (b_{ij})$ eine $n_b \times m_b$ Matrix. Die daraus resultierende Matrix $A \otimes B = C := (c_{ij})$ ist eine $n_a n_b \times m_a m_b$ Matrix, die wie folgt generiert wird:

$$C = A \otimes B = \begin{bmatrix} a_{11} & \cdots & a_{1m_a} \\ \vdots & \ddots & \vdots \\ a_{n_a 1} & \cdots & a_{n_a m_a} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & \cdots & b_{1m_b} \\ \vdots & \ddots & \vdots \\ b_{n_b 1} & \cdots & b_{n_b m_b} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} \begin{bmatrix} b_{11} & \cdots & b_{1m_b} \\ \vdots & \ddots & \vdots \\ b_{n_b 1} & \cdots & b_{n_b m_b} \end{bmatrix} & \cdots & a_{1m_a} \begin{bmatrix} b_{11} & \cdots & b_{1m_b} \\ \vdots & \ddots & \vdots \\ b_{n_b 1} & \cdots & b_{n_b m_b} \end{bmatrix} \\ \vdots & \ddots & \vdots \\ a_{n_a 1} \begin{bmatrix} b_{11} & \cdots & b_{1m_b} \\ \vdots & \ddots & \vdots \\ b_{n_b 1} & \cdots & b_{n_b m_b} \end{bmatrix} & \cdots & a_{n_a m_a} \begin{bmatrix} b_{11} & \cdots & b_{1m_b} \\ \vdots & \ddots & \vdots \\ b_{n_b 1} & \cdots & b_{n_b m_b} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1m_b} & a_{12}b_{11} & \cdots & a_{1m_a}b_{1m_b} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2m_b} & a_{12}b_{21} & \cdots & a_{1m_a}b_{2m_b} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{n_b 1} & a_{11}b_{n_b 2} & \cdots & a_{11}b_{n_b m_b} & a_{12}b_{n_b 1} & \cdots & a_{1m_a}b_{n_b m_b} \\ a_{21}b_{11} & a_{21}b_{12} & \cdots & a_{21}b_{1m_b} & a_{22}b_{11} & \cdots & a_{2m_a}b_{1m_b} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n_a 1}b_{n_b 1} & a_{n_a 1}b_{n_b 2} & \cdots & a_{n_a 1}b_{n_b m_b} & a_{n_a 2}b_{n_b 1} & \cdots & a_{n_a m_a}b_{n_b m_b} \end{bmatrix}$$

2.2 Quantensysteme

Die Quantensysteme, die wir im Rahmen dieser Arbeit betrachten, sind rein mathematische Systeme, die nicht an eine fixe physikalische Realisierung gebunden sind. In der Tat gibt es verschiedene Möglichkeiten, solche Systeme physikalisch umzusetzen, wobei jede Variante eigene Vor- und Nachteile hat.

2.2.1 Qubits

Ein klassisches Bit hat genau zwei Zustände - 0 und 1 - und ist immer in genau einem dieser beiden Zustände. Kann man zwei solche Zustände in einem quantenmechanischen System gezielt herbeizuführen, treten auch die Gesetze der Quantenphysik in Kraft. Ein Beispiel einer physikalischen Realisierung, das bei der Vorstellung helfen kann, wäre ein Atom in einem Grundzustand und einem angeregten Zustand, wobei der Grundzustand einem klassischen 0 und der angeregte Zustand einem klassischen 1 entsprechen könnte. Gemäss den Gesetzen der Quantenphysik kann sich das Atom aber auch in einer Überlagerung der beiden Zustände befinden. Dies nennt man eine Superposition. Von diesem Beispiel aus können wir uns nun die genaue Definition eines Qubits betrachten:

Definition: Ein *Qubit* ist ein Quantensystem mit den beiden Basiszuständen $|0\rangle$ und $|1\rangle$. Es kann alle Zustände $\alpha|0\rangle + \beta|1\rangle$ annehmen, sodass $|\alpha|^2 + |\beta|^2 = 1$ gilt.

Ein Zustand des Qubits, welcher durch die Parameter α und β definiert ist, entspricht einem normierten Vektor im Vektorraum $\mathbb{C}^2$.[4]

[4]Dass sich der Zustand eines quantenmechanisches System als Vektor beschreiben lässt, garantiert uns das erste Postulat der Quantenmechanik. Da diese Postulate den Rahmen dieser Arbeit übersteigt, verweise ich für eine genauere Betrachtung dieser Postulate auf [10], Kapitel 2.2

Wenbn wir zu unserem Beispiel von vorhin zurückgehen, dann würde der Zustand $|0\rangle$ dem Grundzustand und der Zustand $|1\rangle$ dem angeregten Zustand entsprechen.

Qubits können wir messen[5]. Betrachten wir ein Qubit im Zustand $\alpha|0\rangle + \beta|1\rangle$, so beträgt die Wahrscheinlichkeit, dass wir $|0\rangle$ messen, $|\alpha|^2$, und die Wahrscheinlichkeit, $|1\rangle$ zu messen, beträgt $|\beta|^2$. Die Bedingung, dass $|a|^2 + |b|^2 = 1$ gelten muss (dass der Vektor normiert sein muss), führt dazu, dass sich die Wahrscheinlichkeiten auf 1 summieren. Nach der Messung kollabiert das Quantensystem in den gemessenen Zustand. Messen wir also $|1\rangle$, befindet sich das Qubit nachher immer im Zustand $|1\rangle$, unabhängig davon, was α und β vorher waren.

Ähnlich entspricht der Zustand eines Multiqubitsystems mit n Qubits einen normierten Vektor im Vektorraum $\mathbb{C}^{2^n}$. Die Basiszustände des Vektorraums entsprechen dabei den einzelnen Kombinationen der $|0\rangle$s und $|1\rangle$s der einzelnen Qubits. Zum Beispiel hat der Vektorraum zu einem Quantensystem mit 2 Qubits die vier Basiszustände $|00\rangle, |01\rangle, |10\rangle$ und $|11\rangle$. Der Bitstring von Länge n innerhalb dem Ket, entspricht dabei der Konfiguration der Qubits. Der Zustand $|101\rangle$ in einem System mit 3 Qubits entspricht dem Zustand, in welchem das erste Qubit im Zustand $|1\rangle$, das zweite im Zustand $|0\rangle$ und das dritte im Zustand $|1\rangle$ ist. Anstelle des Bitstrings wird innerhalb des Kets auch oft eine Zahl verwendet, welche in der Binärdarstellung diesen Bitstring ergibt. Zum Beispiel entspricht $|5\rangle$ dem Zustand $|101\rangle$.
Ein Zustand ist nun ein normierter Vektor im Vektorraum $\mathbb{C}^{2^n}$. Der Koeffizient des Basiszustands $|j\rangle$ eines Vektors in diesem Vektorraum entspricht dabei der Wahrscheinlichkeit, den Zustand j zu messen. Auch hier summmieren sich die Wahrscheinlichkeiten wieder auf 1, da der Vektor normiert ist. Auch hier schreibt man die Zustände auch oft wieder als eine Linearkombiantion der Basiszustände: $\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle + \alpha_2|2\rangle + \alpha_3|3\rangle$.

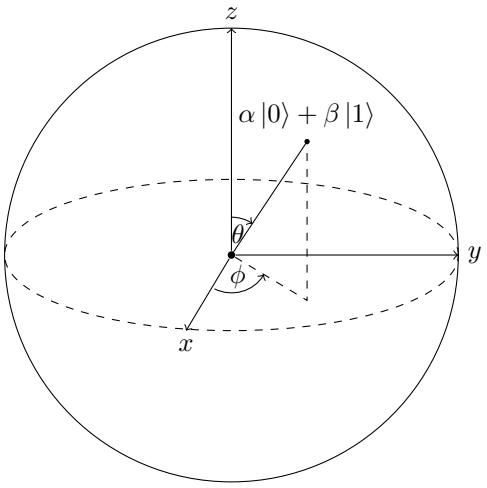
Um den Zustand des Multiqubitsystems beschreiben zu können, wenn wir im Besitz der Zustände der einzelnen Qubits sind, können wir die Vektoren dieser Zustände mit Hilfe des Tensorprodukts zusammenmultiplizieren, und bekommen dann den Zustandsvektor für das Multiqubitsystem. Zudem kann man mit dem Tensorprodukt auch Operationen zusammenmultiplizieren: Wenn wir n Qubits haben und auf jedes Qubit eine Operation anwenden (dies kann auch die Identität sein, falls wir das Qubit unverändert lassen wollen). Wenn wir dann die Matrizen dieser Operationen multiplizieren, bekommen wir die Matrix, die der Operation entspricht, die alle unsere ausgewählten Operationen ausführt. Wichtig ist aber noch anzumerken, dass das Tensorprodukt nicht kommutativ ist, und die Reihenfolge der Faktoren somit wichtig ist.

Falls wir nur einzelne Qubits messen, kollabiert das System in die restlichen, noch möglichen Zustände. Nehmen wir als Beispiel ein 2-Qubit-System im Zustand $\frac{1}{\sqrt{6}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{3}}|11\rangle$ und messen das erste Qubit. Die Wahrscheinlichkeit, dass wir dieses Qubit im Zustand $|1\rangle$ messen, liegt bei $|\alpha_{10}|^2 + |\alpha_{11}|^2 = \frac{1}{3}$. Falls wir diesen Zustand messen, kollabiert unser Quantensystem sofort in den Zustand $\frac{\alpha_{10}|10\rangle + \alpha_{11}|11\rangle}{\sqrt{|\alpha_{10}|^2 + |\alpha_{11}|^2}} = |11\rangle$, wobei der Nenner dafür sorgt, dass der neue Quantenzustand wieder normiert ist. Die Wahrscheinlichkeit eines $|0\rangle$ in der Messung des ersten Qubit hingegen liegt bei $|\alpha_{00}|^2 + |\alpha_{01}|^2 = \frac{2}{3}$. Der Zustand des Systems nach der Messung ist dann $\frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} = \sqrt{\frac{1}{4}}|00\rangle + \sqrt{\frac{3}{4}}|01\rangle$.

[5]Auch die Messungen basieren auf einem Postulat der Quantenmechanik, nämlich auf dem dritten Postulat.

2.2.2 Die Blochkugel

Die Blochkugel dient der graphischen Darstellung des Zustands eines einzelnen Qubits. Zu diesem Zweck betrachten wir noch einmal ein einzelnes Qubit $\alpha|0\rangle + \beta|1\rangle$ mit $|\alpha|^2 + |\beta|^2 = 1$. Diese Bedingung führt dazu, dass wir den Zustand als $e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right)$ umschreiben können. Den Faktor $e^{i\gamma}$ können wir nicht beobachten, da er auf beide Koeffizienten wirkt[6]. Deshalb ist der Zustand durch die beiden Winkel θ und φ definiert. Diese beiden Winkel kann man graphisch als einen Punkt auf einer Einheitskugel darstellen. Diese Darstellung wird die Bloch-Kugel genannt.



2.2.3 Operationen auf Qubits

Operationen, die man auf Qubits implementiert, sind lineare Operationen und lassen sich somit als Matrizen darstellen. Man kann den Vektor, welcher den Zustand beschreibt, mit der Matrix des Operators multiplizieren, um den Zustand nach der Operation zu erhalten. Dazu müssen die Operatoren die Norm des Quantenzustands bewahren, da die Zustände immer normiert sein müssen, und somit unitär sein. Dies hat die direkte Konsequenz (für eine unitäre Matrix U gilt $UU^\dagger = I$), dass ein inverser Operator existieren muss und deshalb alle Berechnungen reversibel sein müssen. Beispielsweise kann der Modulo Operator nicht auf Quantencomputern implementiert werden, da man aus dem Ergebnis $x \equiv 2 \pmod 3$ die Eingabe $x \in \{2, 5, 8 \cdots\}$ nicht eindeutig wiederherstellen kann. Dies hat grosse Konsequenzen für die Berechnungen auf Quantencomputern.

Gleichzeitig stellt sich heraus, dass es verschiedene universelle Kombinationen von Operationen gibt, wobei universell in diesem Zusammenhang bedeutet, dass man jeden unitären Operator nur mit den ausgewählten Operatoren beliebig annähern kann. Die Konstruktion dazu kann man in [10], Seiten 188ff. nachlesen.

2.2.4 Wichtige Quantengatter

An dieser Stelle werden die wichtigsten Quantengatter eingeführt, die wir benötigen werden. Zuerst betrachten wir fünf grundlegende Gatter auf Qubits: die drei Pauli-Matrizen X, Y und Z,

[6] Dies nennt man eine globale Phase: Dieser Faktor hat Betrag 1 und verändert deshalb die Wahrscheinlichkeiten nicht. Da er sich zudem auf alle Zustände auswirkt, kann man ihn bei den Quantenoperatoren wegen der Linearität ausklammern. Somit bleibt er immer über alle Qubits bestehen.

das H-Gatter und das $CNOT$-Gatter.

- Das X-Gatter ist das Qubit-Equivalent zum NOT-Gatter eines elektronischen Schaltkreises. In Matrixform sieht der Operator so aus: $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Dieses Gatter dreht den Zustand des Qubits um π um die x-Achse in der Blochkugel.
- Das Y-Gatter wird durch die Matrix $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ dargestellt. Dieses Gatter entspricht einer Rotation von π um die y-Achse in der Blochkugel.
- Das Z-Gatter, als Matrix $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, dreht den Zustand um π um die z-Achse.
- Das Hadamard-Gatter, meistens einfach durch den Buchstaben H abgekürzt, ist der einfachste Weg, eine Superposition zu erzeugen. Mit der Matrix $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ kann man die beiden Zustände $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ und $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ erzeugen. Diese beiden Zustände kommen so häufig vor, dass man ihnen die Namen $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ und $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ gegeben hat.
- $CNOT$ steht als Abkürzung für "Controlled NOT". Dieses Gatter wirkt auf zwei Qubits und wendet ein NOT auf das zweite Qubit an, wenn das erste Qubit auf 1 ist. Als Matrix sieht die Operation so aus:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

.

Diese Gatter gehören zu den wichtigsten Gattern im Bereich der Quantencomputer. Wir werden auf unserem Weg jedoch weitere Gatter antreffen. Eines, von welchem wir noch mehr Gebrauch machen werden, möchte ich hier kurz definieren. Ich nenne es $\mathrm{Rot}(k)$ und in Matrix-Form sieht es so aus: $\mathrm{Rot}(k) = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2i\pi}{2^k}} \end{bmatrix}$. Dieses Gatter multipliziert den Koeffiztienten von $|1\rangle$ mit $e^{\frac{2i\pi}{2^k}}$ und wir werden es bei der Quanten-Fouriertransformation[7] und dessen Anwendungen antreffen.

Abschliessend möchte ich noch ein Gatter auf zwei Qubits erwähnen, bekannt als das SWAP-Gatter. Dieses Gatter vertauscht die Zustände der beiden Qubits, indem es die Amplituden in derjenigen Zustände vertauscht, in welchen die beiden Qubits nicht gleich sind. Es wird durch die folgende Matrix beschrieben:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Wenn wir das erste Qubit als a und das zweite als b bezeichnen, sehen wir auf der rechten Seite, dass wir das $SWAP(a, b)$ als Abfolge dreier $CNOT$-Gates ($CNOT(a, b)$, dann $CNOT(b, a)$ und dann $CNOT(a, b)$) programmieren können.

[7] Siehe Kapitel 3

2.2.5 Kontrollierte und adjungierte Operatoren

Wie wir bereits erfahren haben, müssen die Operatoren, welche man auf Quantencomputern umsetzen kann, unitär und somit reversibel sein. Dies setzt sofort die Existenz eines adjungierten Operatoren (auch inversen Operator genannt) zu einem Operator U voraus, welcher aus einem Zustand $|\varphi\rangle$ den Zustand $|\psi\rangle$ mit $U|\psi\rangle = |\varphi\rangle$ macht. Ein Schaltkreis zu diesem adjungiertem Operator lässt generieren, indem man die Gatter des Schaltkreises für U rückwerts durchgeht, und immer den adjungierten Operator des jeweiligen Gatters nimmt. Viele Quantenprogrammiersprachen können deshalb auch die die adjungierte Version eines Quantenoperators aus dem Programmcode des ursprünglichen Operators generieren.

Auch lassen sich kontrollierte Versionen eines Operators bilden. Die kontrollierte Version eines Operators nimmt zusätzlich zu den Qubits, auf welche die Operation ausgeführt werden soll, noch eine Menge an sogenannten Kontrollqubits. Diese Version der Operation wird nur dann ausgeführt, wenn die Kontrollqubits im Zustand $|1\rangle$ sind. Als Beispiel nehmen wir eine Operation, welche die Bitrepräsentation zyklisch um ein Bit nach rechts verschiebt (also aus dem Zustand $|01101\rangle$ den Zustand $|10110\rangle$ macht). Haben wir nun den Zustand $|0\rangle|11101\rangle + |1\rangle|10110\rangle + |0\rangle|01000\rangle$ haben (das erste Bit habe ich hier getrennt dargestellt, da es für diese Operation als Kontrollqubit verwenden will). Wenn wir nun die Operation mit dem ersten Bit als Kontrollqubit anwenden, so bekommen wir den Zustand $|0\rangle|11101\rangle+|1\rangle|01011\rangle+|0\rangle|01000\rangle$. Sei U ein Operator, welcher auf Quantencomputern programmierbar ist. Dann schreiben wir die durch das Qubit c kontrollierte Version von U als U^c. In den Fällen, in denen wir mit U^k die k mal wiederholte Anwendung des Operators U meinen, werde ich darauf hinweisen.

Ein Qubit darf aber nie gleichzeitig ein Kontrollqubit und eines jener Qubits, auf dem der Operator angewendet wird, sein. Um dies zu sehen betrachten wir das $CNOT$, welches die kontrolleirte Variante des X-Operators ist. Wenn wir de Operator auf ein QUbit mit demselben QUbit als Kontrollqubit anwenden, so wird dieses Qubit nach der Anwendung immer im Zustand $|0\rangle$ sein. Somit ist dieser Operator nicht mehr reversibel. Wenn diese Bedingung aber eingehalten wird, so ist dieser Operator reversibel, da man aus den Kontrollqubits, welche während der Operation nicht verändert werden, ablesen kann, ob die Operation durchgeführt wurde.

Auchg diese Version wird von den meisten Quantenprogrammiersprachen automatisch generiert, da man den Schaltkreis der kontrollierten Version einfach herstellen kann, indem man vom Schaltkreis des ursprünglichen Operators bei jedem Gatter die kontrollierte Version nimmt.

Zudem lassen sich die beiden oben genannten Varianten auch kombinieren, um eine kontrollierte, adjungierte Version eines Operators zu generieren.

Kapitel 3

Arithmetische Operationen auf Qubits ausführen - Die QInteger Library

3.1 Zahlen in Qubits speichern - Der QInt-Typ

Da ich davon ausgehe, dass in absehbarer Zeit die Anzahl Qubits zwar wachsen, allerdings jedoch nicht so schnell ansteigen wird, dass man schon bald mehrere grössere Qubit-Einheiten speichern kann, habe ich mich entschieden, in meiner Implementation auf eine einheiltiche Grösse zu verzichten und für den Moment eine variable Anzahl von Qubits für eine Zahl zu benutzen. Deshalb besteht der QInt-Typ aus einer klassischen Zahl, der Anzahl Qubits, und einem Array von Qubits, welcher die eigentliche Zahl, welche in den Qubits gespeichert werden soll, speichert.

```
// Definition of the QInt type with variable size. QInts
// are represented in little-endian.
newtype QInt = (Size : Int, Number : Qubit[]);
```

Dies ist die Definition des Typs QInt in meinem Code: Wir definieren den neuen Typ "QInt" als eine Kombination einer Zahl, genannt *Size*, und einem Array von Qubits, genannt *Number*.

Im weiteren Verlauf dieser Arbeit werde ich Zahlen, die auf eine klassische Art und Weise gespeichert sind und somit keine Superpositionen erlauben, als eine "klassische Zahl" bezeichnen und Zahlen, welche in Qubitregistern gespeichert werden, als "Quantenzahlen" oder "QInts" bezeichnen.

3.2 Die Quanten-Fouriertransformation und die Fourier-Basis

3.2.1 Die Quantenfouriertransformation

Sei $|x\rangle$ ein Element einer Orthonormalbasis. Damit können wie die QFT wie folgt definieren:

$$QFT\,|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{2i\pi \frac{xj}{2^n}} \,|j\rangle$$

Für diesen Zustand gibt es eine Faktorisierung. Diese werde ich kurz zeigen und darauf auch beweisen.

$$\left(|0\rangle + e^{2i\pi \frac{x}{2^1}} \,|1\rangle\right) \otimes \left(|0\rangle + e^{2i\pi \frac{x}{2^2}} \,|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi \frac{x}{2^n}} \,|1\rangle\right) = \bigotimes_{j=0}^{n-1} \left(|0\rangle + e^{2i\pi \frac{x}{2^{1+j}}} \,|1\rangle\right)$$

Diese Faktorisierung werde ich nun kurz beweisen. Sei die Darstellung von $j \in \{0, 1, \ldots, 2^n - 1\}$ im Binärsystem:

$$j = 2^{n-1} \cdot j_{n-1} + \ldots + 2^1 \cdot j_1 + 2^0 \cdot j_0 = \sum_{s=0}^{n-1} 2^s j_s$$

Dann können wir folgende Umformung durchführen:

$$\begin{aligned}
&\bigotimes_{s=0}^{n-1} \left(|0\rangle + e^{2i\pi \frac{x}{2^{(1+s)}}} \,|1\rangle\right) = \bigotimes_{s=0}^{n-1} \left(e^{2i\pi \frac{0\cdot x}{2^{(1+s)}}} \,|0\rangle + e^{2i\pi \frac{1\cdot x}{2^{(1+s)}}} \,|1\rangle\right) \\
&= \sum_{(j_{n-1},\ldots,j_0)\in\{0,1\}^n} \bigotimes_{s=0}^{n-1} e^{2i\pi \frac{x\cdot j_{n-1-s}}{2^{(1+s)}}} = \sum_{(j_{n-1},\ldots,j_0)\in\{0,1\}^n} \left(\prod_{s=0}^{n-1} e^{2i\pi \frac{x\cdot j_{n-1-s}}{2^{(1+s)}}}\right) |j_{n-1} j_{n-2} \cdots j_0\rangle \\
&= \sum_{j=0}^{2^n-1} \left(\prod_{s=0}^{n-1} e^{2i\pi \frac{x\cdot j_{n-1-s}}{2^{(1+s)}}}\right) |j\rangle = \sum_{j=0}^{2^n-1} \left(e^{2i\pi \frac{x\sum_{s=0}^{n-2} j_{n-1-s}\cdot 2^{n-1-s}}{2^n}}\right) |j\rangle = \sum_{j=0}^{2^n-1} e^{2i\pi \frac{xj}{2^n}} \,|j\rangle
\end{aligned}$$

Schauen wir uns nun die Produktionsdarstellung der Fouriertransformation von $|x\rangle$ genauer an. Dazu benutzen wir die folgende Notation, um x im Binärsystem darzustellen:
$x = 2^{n-1} \cdot x_{n-1} + \cdots + 2^1 \cdot x_1 + 2^0 \cdot x_0 =: (x_{n-1} \ldots x_1 x_0)$ für $x_s \in 0, 1$
und für $k \in \mathbb{N}, k < n$ schreiben wir:
$\frac{x}{2^k} =: (x_{n-1} \ldots x_k . x_{k-1} \ldots x_1 x_0)$ und $\frac{x}{2^n} =: (0.x_{n-1} \ldots x_1 x_0)$
Diese Schreibweise ist das Äquivalent zu den Dezimalzahlen mit Nachkommastellen im binären Zahlensystem.

Mit dieser Notation erhält man:

$$\left(|0\rangle + e^{2i\pi \frac{x}{2^1}}\right) \otimes \left(|0\rangle + e^{2i\pi \frac{x}{2^2}}\right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi \frac{x}{2^{n-1}}}\right) \otimes \left(|0\rangle + e^{2i\pi \frac{x}{2^n}}\right)$$

$$= \left(|0\rangle + e^{2i\pi \frac{(x_{n-1}\ldots x_0)}{2^1}}\right) \otimes \left(|0\rangle + e^{2i\pi \frac{(x_{n-1}\ldots x_0)}{2^2}}\right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi \frac{(x_{n-1}\ldots x_0)}{2^n}}\right)$$

$$= \left(|0\rangle + e^{2i\pi\cdot(x_{n-1}\,\ldots\,x_1+0\,.\,x_0)}\right) \otimes \left(|0\rangle + e^{2i\pi\cdot(x_{n-1}\,\ldots\,x_2+0\,.\,x_1x_0)}\right) \otimes \ldots$$

$$\ldots \otimes \left(|0\rangle + e^{2i\pi\cdot(x_{n-1}+0\,.\,x_{n-2}\,\ldots\,x_0)}\right) \otimes \left(|0\rangle + e^{2i\pi\cdot(0\,.\,x_{n-1}\,\ldots\,x_0)}\right)$$

$$= \left(|0\rangle + e^{2i\pi\cdot(0\,.\,x_0)}\right) \otimes \left(|0\rangle + e^{2i\pi\cdot(0\,.\,x_1x_0)}\right) \otimes \ldots$$

$$\ldots \otimes \left(|0\rangle + e^{2i\pi\cdot(0\,.\,x_{n-2}\,\ldots\,x_0)}\right) \otimes \left(|0\rangle + e^{2i\pi\cdot(0\,.\,x_{n-1}\,\ldots\,x_0)}\right)$$

Wobei wir im letzen Abschnitt benutzt haben, dass für $z \in \mathbb{Z}$ und $\alpha \in \mathbb{R}$ gilt:

$$e^{2i\pi\cdot(z+\alpha)} = \underbrace{e^{2i\pi z}}_{1} \cdot e^{2i\pi\alpha} = e^{2i\pi\alpha}$$

Unter der Darstellung der Zahl $x \in \mathbb{N}_0$ in der *binären Basis* verstehen wir die Darstellung von x im Binärsystem, also $x = 2^{n-1} \cdot x_{n-1} + \cdots + 2^1 \cdot x_1 + 2^0 \cdot x_0$, beziehungsweise die Darstellung durch den Zustand $|x_{n-1} \ldots x_0\rangle$.

Unter der Darstellung x in der *Fourierbasis* verstehen wir die Produktionsdarstellung der Fouriertransformierten von $|x\rangle$, also

$$\frac{1}{\sqrt{2^n}} \left(|0\rangle + e^{2i\pi(0.x_0)}\right) \otimes \left(|0\rangle + e^{2i\pi(0.x_1x_0)}\right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi(0.x_{n-1}\ldots x_0)}\right)$$

Die im obigen Produkt vorkommenden Zustände nennen wir in diesem Zusammenhang die *Fourierbasis*[1].

Dass wir die Darstellung einer Zahl in der Fourierbasis als Tensorprodukt der einzelnen Qubits schreiben können, bedeutet, dass diese Qubits unabhängig voneinander, also nicht verschränkt sind. Dies bedeutet für uns, dass wenn wir mit einer Zahl in der Fourierbasis rechnen wollen, so können wir die Qubits einzeln bearbeiten. Zudem sehen wir, wenn wir den Zustand eines einzelnen Qubits betrachten, $\frac{1}{\sqrt{2}} \left(|0\rangle + e^{2i\pi \frac{x}{2^{(1+s)}}}\right)$ betrachten, dass dieser Zustand einem Zeiger in der Blochkugel entspricht, welcher in der XY-Ebene liegt und dort um $2\pi \frac{x}{2^{s+1}}$ um den Mittelpunkt gedreht ist.

[1] Man beachte, dass die Fourierbasis keine Vektorraumbasis im Sinno von Kapitel 2.1.2 ist.

Oben wurden mit Hilfe der Blochkugel die Zustände der Qubits der Zahl 5 in der Fourierbasis dargestellt.
Hier können wir feststellen, dass wenn wir die Zahl in der Fourierbasis verändern wollen, müssen wir nur die Qubits im richtigen Winkel drehen, was der Multiplikation des Koeffizienten von $|1\rangle$ mit $e^{i\pi\theta}$ entspricht. Zudem können wir feststellen, dass wenn wir den Koeffizienten von $|1\rangle$ eines Qubits der Fourierbasis mit $e^{2i\pi\theta}$ multiplizieren, dann können wir den Winkel θ mit Hilfe der inversen QFT auslesen. Dies werden wir im Kapitel 4.3 benötigen.

3.2.2 Unitarität

Wir haben allerdings noch nicht gezeigt, dass die oben beschriebene Operation unitär ist. Dafür schauen wir uns die Matrix an, die die Operation der QFT darstellt. Dafür sei $\omega_N^j = e^{2i\pi\frac{j}{N}}$. Dann können wir die QFT wie folgt darstellen $QFT := (\omega_{2^n}^{ij})$. Nun können wir die Bedingung $QFT^\dagger\, QFT = I$ überprüfen. Dazu berechnen wir den Eintrag ij von $QFT^\dagger\, QFT$:

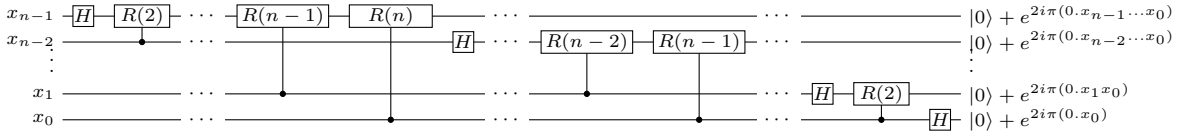
$$\sum_{k=0}^{2^n-1} \frac{\overline{\omega_{2^n}^{ik}}}{\sqrt{2^n}} \cdot \frac{\omega_{2^n}^{kj}}{\sqrt{2^n}} = \frac{1}{2^n} \sum_{k=0}^{2^n-1} \omega_{2^n}^{-ik} \cdot \omega_{2^n}^{kj} = \frac{1}{2^n} \sum_{k=0}^{2^n-1} \omega_{2^n}^{k(j-i)}$$

Nun stellen wir fest, dass falls $i = j$ gilt, dass dann $\omega_{2^n}^{k\cdot 0} = 1$ gilt und somit $\frac{1}{2^n}\sum_{k=0}^{2^n-1} \omega_{2^n}^{k0} = 1$ gilt. Gleichzeitig, falls $i \neq j$ gilt, dann gilt $\frac{1}{2^n}\sum_{k=0}^{2^n-1} \omega_{2^n}^{k(j-i)} = \frac{1}{2^n} \cdot \frac{w_{2^n}^{(j-i)2^n}-1}{w_{2^n}^{(j-i)}-1}$ (da $\omega_{2^n}^{0(j-i)}$, $\omega_{2^n}^{0(j-i)}$, ..., $\omega_{2^n}^{(2^n-1)(j-i)}$ eine geometrische Reihe ist). Gleichzeitig gilt $w_{2^n}^{(j-i)2^n} = 1$, und somit $\frac{1}{2^n} \cdot \frac{w_{2^n}^{(j-i)2^n}-1}{w_{2^n}^{(j-i)}-1} = 0$, da $\omega_{2^n}^{(j-i)} \neq 1$ gilt.

Nun sehen wir, dass in der resultierenden Matrix det Eintrag ij dann 1 ist, wenn $i = j$ gilt, und sonst 0. Dies ist aber genau die Definition von I. Daraus folgt, dass die QFT eine unitäre Operation ist, und somit auf Quantencomputern implementierbar ist.

3.2.3 Implementierung

Gehen wir zurück zur Produktdarstellung der Fouriertransformierten von $|x\rangle$. Diese lässt sich folgendermassen mit Quantengattern realisieren:



In dieser Grafik habe ich aus Platzgründen den Namen $R(k)$ für das $Rot(k)$-Gatter verwendet. Um die Funktionsweise dieses Schaltkreises zu sehen, bemerken wir, dass $H\,|x_j\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi(0.x_j)}\,|1\rangle\right)$ gilt, da $e^{2i\pi\cdot 0} = 1$ und $e^{2i\pi\cdot\frac{1}{2}} = -1$ gilt. Nun wenden wir $Rot(2)^{x_{j-1}}$ an. Wir bekommen:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi(0.x_j)}\,|1\rangle\right) \xrightarrow{Rot(2)^{x_{j-1}}} \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi\frac{x_j}{2}+2i\pi\frac{x_{j-1}}{4}}\,|1\rangle\right) = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi(0.x_jx_{j-1})}\,|1\rangle\right)$$

Dies können wir fortsetzen:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi(0.x_j x_{j-1})}|1\rangle\right) \xrightarrow{Rot(3)^{x_{j-2}}} \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi\left(\frac{x_j}{2}+\frac{x_{j-1}}{4}+\frac{x_{j-2}}{8}\right)}|1\rangle\right) = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi(0.x_j x_{j-1} x_{j-2})}|1\rangle\right)$$

$$\ldots$$

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi(0.x_j x_{j-1}\ldots x_{j-k+1})}|1\rangle\right) \xrightarrow{Rot(1+k)^{x_{j-k}}} \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi(0.x_j x_{j-1} x_{j-2}\ldots x_{j-k+1} x_{j-k})}|1\rangle\right)$$

So können wir mit Hilfe der $Rot(k)$-Gatter die Nachkommastellen der Binärzahl eine Stelle nach der anderen aufbauen. Zum Schluss müssen wir noch die Reihenfolge der Qubits umkehren, was mit $SWAP$-Gattern erreicht werden kann.

Wenn wir uns den Schaltkreis anschauen, stellen wir fest, dass wir $O(n^2)$ Gatter benötigen, um diese Operation zu implementieren, wobei n die Anzahl Qubits des Registers ist. Dabei werden auch keine zusätzlichen temporäre Qubits benötigt.

3.3 Addition

Die wohl grundlegendste arithmetische Operation ist die Addition. Die Subtraktion kann als Addition ausgedrückt werden und auch die Multiplikation (und somit die Division) ist abhängig von der Addition. Deshalb ist sie die erste arithmetische Operation, die wir betrachten.

Wir wollen die Operation implementieren, welche zwei QInts mit Quantenregistern der Grösse n in den Zuständen $|x\rangle$ und $|y\rangle$ nimmt und den Zustand des zweiten QInt zu $|x+y\rangle$ umwandelt. Die Implementation anderer Additionsmethoden (Addition einer klassischen Zahl zu einem QInt, Addition zweier QInts in ein drittes QInt) funktioniert analog. Zusätzlich kann man feststellen, dass die Subtraktion nichts anderes als die inverse Operation zur Addition ist, somit hat man zur Addition die Subtraktion mit-implementiert.

Für die Addition gibt es zwei verschiedene Techniken, die oft benutzt werden. Die eine erreicht eine Gatterzahl von $\mathcal{O}(n)$, benötigt aber $\mathcal{O}(n)$ zusätzliche Qubits, während die andere ohne zusätzliche Qubits auskommt, dafür aber $\mathcal{O}(n^2)$ Gatteroperationen benötigt. Ich habe mich entschieden, die zweite Version in meiner QInteger-Library zu implementieren. Gründe dafür sind, dass in heutigen Systemen die Anzahl verfügbarer Qubits stark begrenzt ist und in Simulationen einzelne Qubits sehr viel zusätzliche Leistung benötigen, während eine Laufzeit von $\mathcal{O}(n^2)$ in diesem Fall weniger ausmacht. Sobald mehr Qubits zur Verfügung stehen, wird es lohnenswerter, auf die andere Version zu wechseln, denn da die Addition eine Operation auf einem sehr tiefen Level ist, kann die Zeit, welche sie benötigt, beträchtliche Auswirkungen auf die gesamte Laufzeit eines komplexeren Algorithmus haben.

Schauen wir uns nun den in der QInteger-Library verwendeten Additionsalgorithmus an. Der Algorithmus basiert auf der Fourierbasis (und somit auf der Faktorisierung der Fouriertransformation). Bei der Addition in der binären Basis sind die einzelnen Bits voneinander abhängig. Deshalb werden sogenannte Carry-Bits verwendet, welche für jedes Bit abspeichern, ob wir beim nächsten Bit noch eine zusätzliche 1 addieren müssen. Dies ist bei der Fourierbasis nicht der Fall: Die Bits sind voneinander unabhängig. Das heisst, wir können die einzelnen Bits voneinander unabhängig modifizieren, ohne dabei auf die anderen Bits achten zu müssen. Dies ist der grosse Vorteil der Fourier-Basis, welcher uns erlaubt, auf zusätzliche Qubits zu verzichten. Betrachten wir nun noch einmal die Faktorisierung: Das j-te Qubit der Zahl y in der Fourierbasis ist im

Zustand $\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{y}{2^{1+j}}}|1\rangle)$. Wir wollen es aber in den Zustand $\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{x+y}{2^{1+j}}}|1\rangle)$ bringen, denn wenn wir alle Qubits in den entsprechenden Zustand bringen könnten, könnten wir mit der inversen QFT den Zustand $|x+y\rangle$ wiederherstellen. Nehmen wir wieder das aus der Fouriertransformation bereits bekannte Gatter $Rot(k) = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0\\ 0 & e^{\frac{2i\pi}{2^k}}\end{bmatrix}$. Mit dem Gatter können wir den Wert $2i\pi\frac{1}{2^k}$ zum Exponenten des Koeffizienten von $|1\rangle$ addieren. Das heisst, wenn wir das Gatter auf ein Qubit im Zustand $\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{y}{2^{1+j}}}|1\rangle)$ anwenden, wird es in den Zustand $\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{y+2^{n-j-k}}{2^{n-j}}}|1\rangle)$ versetzt. Wir können also mit Hilfe dieses Gatters Zweierpotenzen zum Qubit in der Fourierbasis addieren. Wenn wir das Qubit im Zustand $|x\rangle$ in der binären Basis belassen, können wir die Addition wie folgt mit $\mathcal{O}(n^2)$ Gatteroperationen implementieren:

1. Wende QFT auf den zweiten Summanden im Zustand $|y\rangle$ an. Das Register befindet sich nun im Zustand $\frac{1}{\sqrt{2^n}}\bigotimes_{j=0}^{n-1}\left(|0\rangle + e^{2i\pi\frac{y}{2^{1+j}}}\right)$.

2. Für jedes $j \in \{0,...,n-1\}$, wende für jedes Bit x_k mit $k \leq j$ ein kontrolliertes $Rot(j+1-k)$ auf das Qubit im y-Register im Zustand $\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2i\pi\frac{y}{1+j}}\right)$ an. Dieses befindet sich danach im Zustand:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{y}{2^{1+j}}}\cdot e^{2i\pi\sum_{s=0}^{j}\frac{x_{j-s}\cdot 2^{j-s}}{2^{1+s}\cdot 2^{j-s}}}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{y+\sum_{s=0}^{j}x_{j-s}\cdot 2^{j-s}}{2^{1+j}}}|1\rangle)$$
$$= \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{y+x}{2^{1+j}}}|1\rangle)$$

   Wobei uns alle Bits mit Indizes grösser als j nicht interessieren, wie man analog zu den Überlegungen aus dem Kapitel 3.2.1 feststellen kann.

3. Die Qubits im zweiten Register befinden sich nun in folgendem Zustand: $\frac{1}{\sqrt{2^n}}\bigotimes_{j=0}^{n-1}\left(|0\rangle + e^{2i\pi\frac{x+y}{2^{1+j}}}\right)$. Mit der inversen QFT kann man aus diesem Zustand den Zustand $|x+y\rangle$ wiederherstellen.

3.4 Modulare Addition

Den uns bekannten Modulo-Operator kann man auf Qubits nicht implementieren, da er nicht invertierbar ist (a und $a+m$ haben dasselbe Resultat Modulo m). Die modulare Addition ist jedoch invertierbar, wenn beide Summanden kleiner als m sind. Wir haben wieder zwei QInts in den Zuständen $|x\rangle$ und $|y\rangle$ und eine klassische Zahl m und möchten den Zustand des zweiten QInts auf $|x+y \pmod m\rangle$ setzen. Hier lässt sich das Modulo m durch einen QInt ersetzen (oder den ersten Summanden durch eine klassische Zahl). Für den Shor-Algorithmus benötigen wir die Operation nur mit einem klassischen Modulo m, die Implementationen für QInt-Modulos folgen analog und sind auch in der QInt-Library enthalten.

Zuerst addieren wir $|x\rangle$ zum Register $|y\rangle$, um das Register in den Zustand $|x+y\rangle$ zu versetzen. Nun überprüfen wir, ob diese Summe grösser oder gleich dem Modulo m ist.

Wie überprüfen wir, ob ein QInt x grösser oder gleich einer anderen Zahl y ist? Hier unterscheiden sich die Implementationen für den Fall, wenn die andere Zahl auch ein QInt oder eine klassische ist, die zugrunde liegende Idee ist jedoch bei beiden Implementationen die gleiche. In der QInteger-Bibliothek ist die Funktion $GreaterOrEqual$ als $\neg LessThan$ implementiert. Das

heisst, wir probieren herausezufinden, ob $x < y$ gilt, und negieren dann das Resultat. Wir können dies wie folgt umformen: $x < y \Leftrightarrow x - y < 0$. Subtrahieren wir also nun y von x müssen wir nun herausfinden, ob das Resultat negativ ist. Ist $x - y$ nun tatsächlich negativ, passiert ein sogenannter *Underflow*. Dies bedeutet, dass $x - y$ zu $2^n + (x - y)$ wird, weil das nächstgrössere Bit fehlt. Um zu sehen, dass dieses von klassischen Bits bekannten Phänomen auf auch Qubits auftritt, beobachten wir, was passiert, wenn wir a und b auf Qubits addieren, mit $a + b \geq 2^n$. Der Zustand in der Addition vor der inversen QFT ist:

$$\bigotimes_{s=0}^{n-1}\left(|0\rangle + e^{2i\pi\frac{a+b}{2^{(1+s)}}}|1\rangle\right) = \bigotimes_{s=0}^{n-1}\left(|0\rangle + e^{2i\pi\left(\frac{a+b}{2^{(1+s)}}-1\right)}|1\rangle\right) = \bigotimes_{s=0}^{n-1}\left(|0\rangle + e^{2i\pi\frac{a+b-2^n}{2^{(1+s)}}}|1\rangle\right)$$

Die Resultate $a+b$ und $a+b-2^n$ sind somit nicht unterscheidbar. Da aber nach der inversen QFT kein Bit vorhanden ist, um das Bit mit dem Wert 2^n zu speichern, wird das Resultat dieser Addition zu $a + b - 2^n$. Gleichzeitig ist die Subtraktion dazu die inverse Operation. Dies bedeutet, wenn wir b von $a+b-2^n$ subtrahieren, gilt $a-2^n < 0$, das Resultat wird aber zu a. Dies bedeutet, dass wenn das Resultat einer Subtraktion $x-y$ kleiner als 0 ist, wird es zu $2^n+(x-y)$. Dies bedeutet, wir können messen, ob $x - y < 0$ gilt, indem wir uns einfach das vorderste Qubit anschauen. Ist dieses 1, dann gab es möglicherweise einen Underflow. Um sicherzugehen, dass das vorderste Bit nur dann 1 ist, und nicht wenn $x - y \geq 2^{n-1}$ gilt, verlängere ich in meiner Implementation noch die Länge der beiden Register um 1.

Nun können wir schauen, ob $x + y \geq m$ gilt, und diese Information in einem zusätzlichen Qubit speichern. Falls $x+y \geq m$ gilt, subtrahieren wir m von der Zahl und haben somit die Zahl $x + y - m$ im zweiten QInt. Nun ist die Information, ob $x + y \geq m$ gilt, noch in einem Qubit gespeichert, welches wir zurücksetzen müssen. Dazu behaupte ich, dass $x + y \geq m$ genau dann gilt, wenn das Resultat grösser oder gleich dem Summanden x ist.

Die Richtung $res < x \rightarrow x + y \geq m$ ist nicht schwierig. Für die andere Richtung sehen wir, dass $x + y - m \geq x$ genau dann gelten kann, falls $y \geq m$ gilt, was aber nach der Annahme $x, y < m$ nicht stimmen kann. Somit können wir mit diesem Vergleich die Information in unserem Aushilfsqubit wieder löschen.

3.5 Modulare Multiplikation

Mit Hilfe der modularen Addition können wir nun die modulare Multiplikation implementieren. Zuerst stellen wir fest, dass wir die modulare Multiplikation $|x\rangle \rightarrow |(ax) \pmod m\rangle$ nur dann implementieren können, wenn $ggT(a, m) = 1$ gilt, ansonsten wäre sie nicht invertierbar (zum Beispiel ist $2 \cdot 3 \equiv 2 \equiv 2 \cdot 1 \pmod 4$). Gilt aber $ggt(a, m) = 1$, so lässt sich ein inverses a^{-1} zu a modulo m finden, so dass $a \cdot a^{-1} \equiv 1 \pmod m$ gilt. Somit lässt sich die Operation durch die Multiplikation mit a^{-1} invertieren.

Wir schauen uns die modulare Multiplikation auf QInts in zwei Schritten an.
Zuerst implementieren wir die Quantenoperation auf zwei Registern, welche für gegebenes a und m folgende Operation bewirkt:

$$U'_{a,m}|x\rangle|y\rangle = |x\rangle|(y + ax) \pmod m\rangle$$

Wir sehen, dass wenn wir $x = 2^0x_0 + 2^1x_1...$ in seine Zweierpotenzen aufteilen, dann können wir $ax \pmod m = x_0(2^0a \pmod m) + x_1(2^1a \pmod m) + ...$ schreiben. Dieses $+ x_0(...)$ ist nichts anders als Addition, kontrolliert durch das x_0 Qubit. Dies heisst, wir können diese Operation

relativ einfach durchführen:
Für jedes j, führe eine modulare Addition, kontrolliert durch das Qubit x_j, auf das Ausgaberegister mit dem Summanden $2^j a \pmod m$ durch, den wir klassisch berechnen können.
Diese Unteroperation ruft den modularen Addierer $\mathcal{O}(n)$ mal auf und jede dieser Additionen braucht $\mathcal{O}(n^2)$ Gatteroperationen. Damit kommen wir auf $\mathcal{O}(n^3)$ Gatteroperationen.

Mit Hilfe dieser Unteroperation können wir nun die Operation, welche

$$U_{a,m} |x\rangle = |(ax) \pmod m\rangle$$

bewirkt, implementieren:

1. Führe ein temporäres Register im Zustand $|0\rangle$ ein und bringe es mit Hilfe der oberen Unteroperation in den Zustand $|ax \pmod m\rangle$.

2. Berechne klassisch das Inverse von a Modulo m. Dieses Inverse existiert, da a und m teilerfremd sind.

3. Die inverse Operation zur oben definierten Unteroperation $U'_{a,m}$ ist

   $$[U'_{a,m}]^{-1} |u\rangle |v\rangle = |u\rangle |v - av \pmod m\rangle$$

   . Wende diese Operation mit a^{-1} mit dem temporären Register als erstem Register und dem Register im Zustand $|x\rangle$ als zweitem Register an. Dies ergibt den Zustand:

   $$[U'_{a^{-1},m}]^{-1} |ax \pmod m\rangle |x\rangle = |ax \pmod m\rangle \left|x - a^{-1}(ax) \pmod m\right\rangle = |ax \pmod m\rangle |0\rangle$$

4. Wechsle den Zustand der beiden Register mit Hilfe der Swap-Operation, sodass $|ax \pmod m\rangle$ in unserem Register und das temporäre Register wieder im Zustand $|0\rangle$ ist. Das temporäre Register im Zustand kann wieder wieder freigeben werden und das erste Register befindet sich nun im Zustand $|ax \pmod m\rangle$.

Diese Multiplikation benötigt n extra Qubits für das temporäre Register. Sie ruft die Unteroperation $2 \in \mathcal{O}(1)$ mal auf und benötigt somit $\mathcal{O}(n^3)$ Gatteroperationen. Wir erinnern uns, dass man die Addition mit $\mathcal{O}(n)$ Gatteroperationen und dafür n zusätzlichen Qubits implementieren könnte, was dazu führt, dass wir nur noch $\mathcal{O}(n^2)$ Gatteroperationen, dafür aber $2n$ extra Qubits benötigen würden.

Kapitel 4

Der Weg zu Shor

4.1 Überblick

In diesem Kapitel werden wir uns die notwendigen Konzepte und Ideen hinter dem quantenbasierten Teil de Shor-Algorithmus anschauen. Dabei beginnen wir beim simplen Konzept des "Phase Kickback" , schauen uns dann die darauf basierende "Phase Estimation" an, bevor wir mit deren Hilfe die Ordnung einer Zahl finden. Zum Schluss werden wir uns den kompletten quantenbasierten Teil des Algorithmus zusammenfassend anschauen.

4.2 Phase-Kickback

Beginnen wir den Abschnitt mit einer Frage: Wenn wir eine kontrollierte Operation ausführen, sollte sich das Control-Qubit eigentlich nicht ändern, oder? In diesem Abschnitt werden wir sehen, dass dies überraschenderweise doch so ist. Dafür schauen wir uns das CNOT-Gatter an. Was geschieht, wenn wir CNOT auf zwei Qubits im Zustand $|+-\rangle$ anwenden, mit dem ersten Qubit als Control-Qubit? Zuerst haben wir $|+-\rangle = |00\rangle - |01\rangle + |10\rangle - |11\rangle$. Nachdem wir das CNOT anwenden, bekommen wir den Zustand $|00\rangle - |01\rangle - |10\rangle + |11\rangle = |--\rangle$. Überraschenderweise stellen wir fest, dass sich das Control-Qubit verändert hat, während das Ziel-Qubit gleich blieb. Was ist passiert? Betrachten wir das CNOT-Gatter genauer: Das CNOT-Gatter ist nichts anderes als eine kontrollierte Version des X-Gatters. Was geschieht, wenn wir das X-Gatter auf den $|-\rangle$-Zustand anwenden? $X|-\rangle = -|0\rangle + |1\rangle = -|-\rangle = (-1) * |-\rangle$. Hier können wir sehen, dass $|-\rangle$ ein Eigenvektor des X-Gatters mit Eigenwert -1 ist. Das heisst, der Zustand des Qubits ändert sich nicht, nur die betroffenen Koeffizienten werden mit dem Eigenwert multipliziert. Da aber in diesem Fall alle Koeffizienten mit diesem Wert (mit Betrag 1) multipliziert werden, können wir keinen Unterschied feststellen.

Wenn wir hingegen die Operation kontrolliert durchführen, wird diese Phase nur in den Zuständen sichtbar, in welchen die Operation durchgeführt wurde, sprich in den Zuständen, in denen das Control-Qubit im Zustand $|1\rangle$ ist. Dies konnten wir zuvor beim CNOT-Gatter beobachten. Betrachten wir nun eine allgemeinere Operation U mit einem Eigenvektor $|\psi\rangle$ und dem Eigenwert λ. Nehmen wir jetzt ein Kontrollqubit im Zustand $\alpha|0\rangle + \beta|1\rangle$, ein Qubit-Register im Zustand $|\psi\rangle$ und führen ein kontrolliertes U auf das Register $|\psi\rangle$ kontrolliert durch jenes Kontrollqubit durch:

$$(\alpha|0\rangle + \beta|1\rangle)|\psi\rangle \xrightarrow{\text{C-U}} \alpha|0\psi\rangle + \beta|1\rangle * U|\psi\rangle = (\alpha|0\rangle + \lambda\beta|1\rangle)|\psi\rangle$$

Das Ziel-Qubit verändert sich nicht, es ist ja ein Eigenvektor, stattdessen sehen wir, dass der Eigenwert zurück in die Phase des Kontroll-Qubit "gekickt" wird, daher der Name Phase Kickback. Im nächsten Abschnitt werden wir diesen Effekt anwenden, um den Eigenwert eines Operators abzuschätzen.

4.3 Phase Estimation

Verschiedene Quanten-Algorithmen basieren darauf, den Eigenwert eines Operators zu einem Eigenvektor abzuschätzen. Dazu benutzen wir Phase-Kickbacks, um den Eigenwert in ein Quantenregister in der Fourier-Basis zu schreiben, welches wir dann mit der inversen Quanten-Fouriertransformation in die binäre Basis zurückrechnen. Dazu können wir die Anzahl Qubits variieren, um die Präzision der Approximation festlegen. Genauer gibt der Algorithmus zum Eigenwert $\lambda = e^{2i\pi\theta}$ die Zahl $2^n\theta$ zurück, wobei n die Anzahl Qubits des Zählerregisters ist, die für bessere Präzision erhöht werden kann.
Um zu verstehen, wie dieser Algorithmus funktioniert, erinnern wir uns zuerst daran, wie eine Zahl in der Fourierbasis aussieht. Dafür benutzen wir die Bloch-Kugel. Wir erinnern uns, dass für die Zahl x in der Fourierbasis mit n Qubits das k-te Qubit um $\frac{2^k x}{2^n}$ um die Z-Achse gedreht wird. Das heisst, es befindet sich im Zustand $\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{2^k x}{2^n}}|1\rangle)$. Wir machen jetzt die Beobachtung, dass wir mit Hilfe von Phase-Kickback das gesuchte θ in der Fourierbasis in die Kontrollqubits schreiben können, da der Phase-Kickback nichts anderes macht, als das Kontrollqubit auf die selbe Art und Weise zu rotieren. Schauen wir uns mal an, was passert, wenn wir 2^k mal das kontrollierte U anwenden:

$$\begin{aligned}
\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|\psi\rangle &\xrightarrow{(C-U)^{2^k}} \frac{1}{\sqrt{2}}(|0\psi\rangle + |1\rangle * U^{2^k}|\psi\rangle) \\
&= \frac{1}{\sqrt{2}}(|0\rangle + (e^{2i\pi\theta})^{2^k}|1\rangle)|\psi\rangle \\
&= \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 2^k\theta}|1\rangle)|\psi\rangle \\
&= \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi\frac{2^k(2^n\theta)}{2^n}}|1\rangle)|\psi\rangle
\end{aligned}$$

Dies entspricht genau dem k-ten Qubit der Repräsentation von $2^n\theta$ in der Fourierbasis. Das heisst, wenn wir für jedes Qubit im Zählerregister mehrmals ein kontrolliertes U anwenden, können wir einen Zustand kreieren, welcher der Zahl $2^n\theta$ in der Fourierbasis entspricht. Wenden wir anschliessend die inverse Fouriertransformation an, können wir die Zahl $2^n\theta$ im Zählerregister ablesen. Falls $2^n\theta$ keine ganze Zahl ist, erhalten wir im Zählerregister eine Superposition, wobei eine Zahl wahrscheinlicher ist, je näher sie am echten Wert liegt.

Um die Phase abzuschätzen, müssen wir also den Operator mehrmals hintereinander anwenden, zuerst nur einmal, dann zweimal, im i-ten Mal 2^i mal. Dies führt dazu, dass wir die Operation 2^n mal anwenden müssen. Allerdings ist es oft möglich, dass wir die Operation U^{2^m} für einen beliebigen Parameter m implementieren können. Wenn dies möglich ist, brauchen wir nur n Anwendungen jener Operation.

Algorithmus

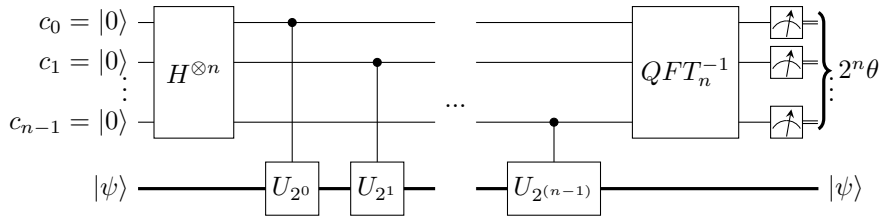
1. Initialisiere zwei Quantenregister, das Zählerregister und das Eigenvektor-Register, und setze das Eigenvektor-Register auf den gewünschten Eigenvektor $|\psi\rangle$.

2. Wende $H^{\otimes n}$ auf das Zähler-Register an, um es auf $|+\rangle^{\otimes n}$ zu setzen.

3. Für jedes i-te Bit im Zählerregister, wende ein kontrolliertes U^{2^i} mit c_i als Kontroll-Qubit an.

4. Wende die inverse Quantenfouriertransformation auf das Zählerregister an, um die Approximation in die binäre Basis umzurechnen.

5. Miss das Zählerregister, um die Abschätzung abzulesen.

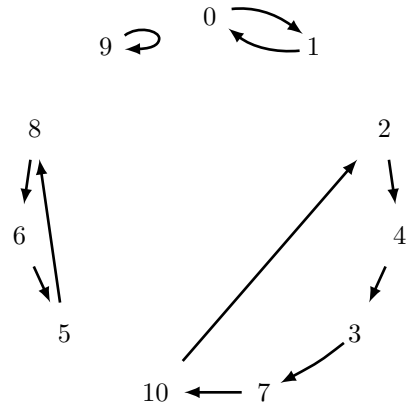


4.4 Period Finding

Gegeben sei eine Funktion $f : S \to S$ mit $S \subset \mathbb{Z}$, welche sich auf einem Quantencomputer implementieren lässt, und ein Wert $x \in S$. Wir versuchen nun, die kleinste Zahl $r \in \mathbb{N}$ zu berechnen, sodass $f^r(x) = x$ gilt.

Unsere Funktion soll auf einem Quantencomputer implementierbar sein. Daraus folgt bereits, dass f bijektiv ist: Falls es ein a und ein b mit $f(a) = f(b) = c$ gibt, lässt sich $f^{-1}(c)$ nicht berechnen, was im Widerspruch zur Reversibilität steht. Daraus folgt, dass f injektiv ist. Aus diesem Grund müssen $|S|$ verschiedene Funktionswerte von f existieren, woraus die Surjektivität folgt. Somit ist f bijektiv und man kann f als Permutation der Elemente von S interpretieren. Diese Permutation kann als Vereinigung disjunkter Zyklen zerlegen. Dies bedeutet, dass man S in verschiedene Teilmengen $S_0, S_1, ...$ aufteilen kann, sodass jede dieser Teilmengen einen einzelnen Zyklus der Permutation bildet. Sei nun $x \in S_i$. Da S_i einen Zyklus bildet, gilt $f^{|S_i|}(x) = x$. Gleichzeitig kann kein $r \in \mathbb{N}$ mit $r < |S_i|$ existieren, sodass $f^r(x) = x$ gilt, denn sonst hätte unser Zyklus nur $r < |S_i|$ Elemente. Wir wollen nun also für ein $x \in S_i$ die Grösse $|S_i|$ finden.

Als Beispiel nehmen wir $g : A \to A$ mit $A = \mathbb{Z}/11\mathbb{Z}$, $g(x) = -x^3 + 1$. Man kann zeigen, dass $x^3 \pmod p$ bijektiv ist, falls $p \equiv 2 \pmod 3$. Somit ist auch f bijektiv. Wenn wir den Graphen anschauen, sehen wir die einzelnen Zyklen: $A_0 = \{0, 1\}$, $A_1 = \{2, 3, 4, 7, 10\}$, $A_2 = \{5, 6, 8\}$ und $A_3 = \{9\}$. Wir sehen nun, dass $f^1(9) = 9$, $f^3(8) = 8$, $f^5(2) = 2$ etc.



Die Frage ist nun, wie können wir effizient die Grösse der Teilmenge finden, in der x sich befindet. Dafür müssen wir den Operator f genauer betrachten. Was passiert, wenn wir dem Operator eine Superposition der Zahlen in S_i übergeben? Seien $r = |S_i|$, $x_0, x_1, ..., x_{r-1}$ die Zahlen in S_i, sodass $f(x_j) = x_{(j+1) \pmod r}$ und U_f die Quantenoperation, die f implementiert. Schauen wir mal, was passiert, wenn wir U_f auf den Zustand $\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |x_j\rangle$ anwenden. Wir bekommen:

$$U_f(\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |x_j\rangle) = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |f(x_j)\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} \left|x_{(j+1) \pmod r}\right\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |x_j\rangle)$$

Daraus schliessen wir, dass $\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |x_j\rangle$ ein Eigenvektor von U_f mit Eigenwert 1 ist. Dieser Eigenwert ist nicht wirklich interessant. Wir können ihn aber interessanter machen, indem wir den einzelnen Summanden eine Phase mitgeben. Dazu konstruieren wir die Superposition $\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} |x_j\rangle)$ für ein $k < r$. Was passiert, wenn wie U_f darauf anwenden?

$$U_f(\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} |x_j\rangle)) = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} \left|x_{(j+1) \pmod r}\right\rangle) =$$
$$\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} (e^{-2i\pi \frac{k(j-1)}{r}} |x_j\rangle) = e^{2i\pi \frac{k}{r}} (\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} |x_j\rangle))$$

Auch hier haben wir wieder einen Eigenvektor, aber mit einem interessanteren Eigenwert, nämlich $e^{2i\pi \frac{k}{r}}$, denn r ist im Eigenwert enthalten. Wir machen auch die Beobachtung, dass unser Eigenvektor von vorher $(\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |x_j\rangle)$ auch von der Form ist, die wir gerade analysiert haben, einfach mit $k = 0$. Falls wir jetzt irgendwie einen Zustand von der Form $\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} |x_j\rangle)$ erzeugen können, könnten wir mit Hilfe der Phase Estimation den Quotienten $\frac{k}{r}$ abschätzen. Die Frage ist, wie können wir solch einen Zustand generieren? Zuerst sagen wir, $|\psi_k\rangle$ sei $\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} |x_j\rangle)$. Dann stellen wir fest, dass $|\psi_0\rangle, |\psi_1\rangle, ..., |\psi_{r-1}\rangle$ linear unabhängig und damit eine Basis des Untervektorraums über die Zahlen $x_0, x_1, ..., x_{r-1}$ sind. Was passiert nun, wenn wir alle diese Vektoren mit gleichem Gewicht aufsummieren?

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \frac{1}{\sqrt{r}} (\sum_{j=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} |x_j\rangle)) = \frac{1}{r} \sum_{j=0}^{r-1} \sum_{k=0}^{r-1} (e^{-2i\pi \frac{kj}{r}} |x_j\rangle) = |x_0\rangle$$

Eine andere Art, dieses überraschende Resultat zu sehen, ist, dass man die Summe $\sum_{k=0}^{r-1}(e^{-2i\pi\frac{kj}{r}}\left|x_j\right\rangle)$ für $j=0$ betrachtet. Da $j=0$ gilt, gilt $e^{-2i\pi\frac{kj}{r}}=e^0=1$ und somit $\sum_{k=0}^{r-1}(e^{-2i\pi\frac{kj}{r}}\left|x_j\right\rangle)=r\left|x_j\right\rangle$. Da $\frac{1}{r}(r\left|x_j\right\rangle)$ bereits einen Betrag von 1 hat, kann kein anderer Zustand mit positivem Betrag existieren, da die Beträge sich sonst zu etwas Grösserem als 1 aufsummieren würden.

Somit ist x_0 einfach eine Superposition jener Eigenvektoren. Da wir der Periodenabschätzungsfunktion einen Startwert mitgeben, sei jener Startwert ohne Beschränkung der Allgemeinheit x_0, haben wir eine Superposition dieser Eigenvektoren. Schätzen wir somit den Eigenwert dieser Superposition ab, kollabiert sie in eine der Eigenvektoren und wir bekommen einen Quotienten $\frac{k}{r}$ zurück, wobei jedes k die gleiche Wahrscheinlichkeit hat. Genauer bekommen wir die Zahl $2^n\frac{k}{r}$ zurück, wobei n die Präzision ist, die wir dem Phase Estimation-Algorithmus mitgeben. Wir können mit Hilfe von Kettenbrüchen den Quotienten $\frac{k}{r}$ vom Quotienten $\frac{2^n\frac{k}{r}}{2^n}$ abschätzen. Sobald wir den Bruch $\frac{k}{r}$ haben, wissen wir r, welches die Zahl ist, die unsere Funktion zurückgeben soll. Nun kann es sein, dass $ggT(k,r)=g\neq 1$ ist, somit der Bruch mit g gekürzt wird und wir dann als Resultat $\frac{r}{g}$ bekommen. Wenn wir die Prozedur aber $2log(N)$ mal wiederholen, bekommen wir mit sehr hoher Wahrscheinlichkeit mindestens einmal die korrekte Periode. Den Beweis dazu kann man in [10], Seiten 229ff., nachlesen.

4.5 Die Ordnung von Zahlen bestimmen

Der Algorithmus von Shor ist deshalb so schnell, da mit Hilfe des quantenbasierten Teils des Algorithmus die Ordnung einer Zahl effizient bestimmt werden kann. Die Ordnung einer Zahl a Modulo einer Zahl n, geschrieben als $ord_n(a)$, ist die kleinste positive Zahl r, sodass $a^r\equiv 1 \pmod n$ gilt. Dabei muss $gcd(a,n)=1$ gelten, denn sonst kann dieses r nicht existieren. Wir rechnen nun in $\mathbb{Z}/n\mathbb{Z}$. Da $ord_n(a)$ nichts anderes ist als die Periode der Funktion $g(x)=a^x$, können wir die Funktion $f(x)=ax$ implementieren, sodass $f_s(x)=f^x(s)=sa^x$ gilt. Mit $s=1$ bekommen wir dann $f_1(x)=f^x(1)=a^x$. Sei U die Quantenoperation, die f_1 implementiert, dafür können wir einfach die Multiplikation aus der QInteger-Library verwenden. Gleichzeitig können wir auch U^{2^i} effizient implementieren: U^{2^i} ist nichts anderes als die Operation zu f^{2^i}. Da $f^{2^i}(x)=a^{2^i}x$, können wir ganz einfach a^{2^i} klassisch berechnen und dann wieder die gewöhnliche Multiplikation aus der QInteger-Library verwenden. Wir können nun den Algorithmus aus dem vorherigen Kapitel verwenden, um die Periode der Funktion $f_1(x)=a^x$ abzuschätzen. Wir brauchen dafür nur noch eine Funktion, die $f(x)=a^x$ klassisch berechnet, um das Resultat überprüfen zu können, dafür kann man direkt FastPowMod aus der QInteger-Library verwenden. Dies führt dazu, dass der Code dieser Funktion nur sehr kurz ist.

4.6 Das Ziel - Der Shor-Algorithmus

Wie erlaubt uns dies nun, Zahlen zu faktorisieren? Sei n die zu faktorisierende Zahl. Zuerst überprüfen wir, ob die Zahl durch 2 teilbar oder eine Primpotenz ist, und finden diese Faktoren entsprechend. Nun nehmen wir ein zufälliges $1<a<n$. Falls $g=ggT(a,n)\neq 1$, haben wir bereits einen Teiler gefunden, nämlich g. Sonst sind a und n teilerfremd. Danach suchen wir die Ordnung von a modulo n. Falls diese Ordnung ungerade ist, beginnen wir nochmals von vorne, sonst ist sie gerade. Sei diese Ordnung r. Mit r können wir nun mit gewisser Wahrscheinlichkeit einen Teiler finden. Falls $a^{\frac{r}{2}}\not\equiv -1 \pmod n$ gilt, haben wir eine Wurzel von $1 \pmod n$ gefunden, sonst müssen wir es noch einmal versuchen. Da r die Ordnung von a ist, muss $a^{\frac{r}{2}}\not\equiv 1 \pmod n$ gelten. Nun gilt: $n|(a^{\frac{r}{2}}+1)(a^{\frac{r}{2}}-1)$, aber $n\not| a^{\frac{r}{2}}+1$ und $n\not| a^{\frac{r}{2}}-1$. Wir haben zwei Zahlen b und c, sodass $n|bc$, aber $n\not| b$ und $n\not| c$. Sei $n=p_0^{\alpha_0}p_1^{\alpha_1}....$ Schreibe $b=s_bp_0^{\beta_0}p_1^{\beta_1}...$ und

$b = s_c p_0^{\gamma_0} p_1^{\gamma_1} ....$ Wir wissen nun, dass $\beta_i + \gamma_i \geq \alpha_i$ gelten muss, da sonst $n|bc$ nicht erfüllt wäre. Gleichzeitig müssen ein j_b und ein j_c existieren, sodass $\beta_{j_b} > 0$ und $\gamma_{j_c} > 0$ stimmt. Nehme an, dass ohne Beschränkung der Allgemeinheit $\gamma_i = 0$ für alle i gelte. Dann müsste $\beta_i \geq \alpha_i$ für alle i gelten und $n \mid b$ teilen, was ein Widerspruch zur Annahme $n \nmid b$ wäre. Somit beinhalten beide Faktoren b und c Teiler von n, welche mit dem einfachen ggT-Algorithmus extrahiert werden können. Somit kennen wir nun den ganzen Algorithmus, um einen Teiler von n zu finden.

1. Falls n durch zwei teilbar ist, gib 2 zurück und terminiere.
2. Falls $n = p^a$ eine Primpotenz ist, gib die Primzahl p zurück und terminiere.
3. Bestimme eine zufällige Zahl $1 < a < n - 1$.
4. Finde $g = ggT(a, n)$. Falls $g \neq 1$ ist, gib g zurück und terminiere.
5. Bestimme die Ordnung von a modulo n mit Hilfe des Quantenteils des Algorithmus:
   (a) Schätze die Phase des Operators U_f ab, der $f(x) = ax$ implementiert mit einer Präzision von $m = 2\log_2(n)$ ab. Benutze dazu den in Kapitel 4.3 vorgestellten Algorithmus. Sei das Resultat $2^m\lambda$.
   (b) Schätze den Quotienten $\frac{k}{r}$ von $\frac{2^m\lambda}{2^m}$ ab. Falls r nicht die gesuchte Periode ist, gehe zurück zu (a), sonst gib die Periode r zurück.
6. Falls r ungerade ist, gehe zurück zu 2. Sonst berechne $a^{r/2} \pmod n$. Falls dies kongruent zu $-1 \pmod n$ ist, gehe zurück zu 2.
7. Berechne $b = (a^{\frac{r}{2}} + 1)$. Gib $ggT(b, n)$ zurück und terminiere.

Kapitel 5

Ausblick

5.1 Man kann Zahlen effizient faktorisieren - Was nun?

Wir haben nun gesehen, dass man Zahlen mit Hilfe von Quantencomputern effizient faktorisieren kann. Einerseits ist dies eine grosse Errungenschaft: Shors Algorithmus demonstriert uns, wie man mit Quantencomputern exponentielle Verschnellerungen erreichen kann. Auf der anderen Seite birgt diese Errungenschaft auch Gefahren. Zum Beispiel können Verschlüsselungsverfahren geknackt werden, wenn man Zahlen faktorisieren kann. Ich möchte hier kurz vorzeigen, wie man das RSA-Verschlüsselungsverfahren knacken kann, wenn man Zahlen effizient faktorisieren kann. Zu einer kurzen Beschreibung verweise ich auf [3] oder für die originale Arbeit auf [12].

Sei $n = pq$ der Modulus der Verschlüsselung und e der öffentliche Schlüssel. Da wir nun $n = pq$ faktorisieren können, können wir $\phi(n) = \phi(p) \cdot \phi(q) = (p-1)(q-1)$ berechnen. Damit können wir den privaten Schlüssel $d \equiv e^{-1} \pmod{\phi(n)}$ berechnen, denn wir wissen, dass $m^{e \cdot d} \equiv m^{e \cdot e^{-1} \pmod{\phi(n)}} \equiv m^1 \equiv m \pmod{n}$ gilt, wobei die letzte Äquivalenz wegen des Satzes von Euler-Fermat gilt.

Da nun Verschlüsselungen, die auf der Schwierigkeit der Faktorisierung beruhen, geknackt werden können, steht das Forschungsgebiet der Kryptographie vor neuen Herausforderungen. Das neue Gebiet, welches sich mit der Kryptographie im Zeitalter der Quantencomputer befasst, nennt sich Post-Quantum-Kryptographie.

5.2 Quantencomputer - Wie bald?

Wie bereits angesprochen, gibt es verschiedene Möglichkeiten, Quantencomputer physikalisch zu realisieren. Technik-Firmen und Forschungsinstitute versuchen schon länger, Quantencomputer zu bauen. Bereits im Jahr 1998 gelang es zwei Forschern aus Oxford, einen Quantenalgorithmus für Deutschs Problem [8], und drei amerikanischen Forschern, Grovers Algorithmus experimentell zu realisieren [4]. Bereits im Jahr 2001 folgte darauf die erste experimentelle Realisation des Shor-Algorithmus [13], mit welcher die Zahl 15 faktorisiert werden konnte.

Über die Jahre entwickelten sich Quantencomputer immer weiter. Zum Zeitpunkt dieser Arbeit haben Tech-Firmen wie Google, IBM, Intel etc. Quantenprozessoren mit bis zu 72 Qubits (Googles Bristlecone [9]). Im Jahr 2019 legte Google einen Artikel [1] vor, laut welchem sie die Quantenüberlegenheit erreicht hätten. Dies bedeutet, dass sie auf einem Quantencomputer etwas effizient berechnet hätten, was auf einem klassischen Computer nicht effizient berechenbar wäre. Dies löste prompt einen Disput aus und die IBM zweifelte in einem Blog die Quantenüberlegenheit

an [11]. Während im Bericht von Google behauptet wurde, ein klassischer Computer würde 10000 Jahre benötigen, wurde in IBMs Blog behauptet, dass ein klassischer Computer dies in 2.5 Tage tun könne. Der Artikel von Google blieb unpubliziert. Trotzdem ist es meiner Meinung nach beeindruckend, dass Googles Quantencomputer für diese Aufgabe nur etwa 200 Sekunden benötigt, während ein Supercomputer 2.5 Tage benötigt.

Die IBM selbst jedoch hat auch grosse Pläne. Im September 2020 hat sie in ihrer Quantum Roadmap [7] angekündigt, dass sie bis im Jahr 2023 einen Quantencomputer mit 1121 Qubits bauen und somit die Qubit-Zahlen sprengen möchte.

Wenn die IBM diesen Plan durchziehen kann und Google sowie die anderen Tech-Firmen auch in naher Zukunft so grosse Quantencomputer bauen, denke ich, dass die Zeit, in der quantenbasierte Supercomputer schwierige Berechnungen übernehmen werden, nicht mehr weit entfernt ist.

Kapitel 6

Nachwort

Ich höre immer wieder die Frage, ob Quantencomputer die Computer der Zukunft sind. Ich denke, dass Quantencomputer in Zukunft sehr wichtig werden. Da sie auf anderen physikalischen Vorgängen beruhen als normale Computer, kann man mit ihnen weitaus schwierigere Dinge berechnen als mit klassischen Comptuern. Zudem ist das Gebiet der Quantencomputer noch sehr neu, deshalb denke ich, dass wir noch nicht das ganze Potenzial von Quantencomputern kennen.

Quantencomputer sind ein aktuelles Thema. Ich lese immer wieder Zeitungs- und Journalartikel über Fortschritte in diesem Gebiet.

Dieses Gebiet weckte so mein Interesse, jedoch verstand ich anfangs nichts davon und ich brauchte eine gewisse Zeit, bis ich die Grundlagen auch nur ansatzweise verstand. Aber das Gebiet faszinierte mich sehr und ich war motiviert, weiterzufahren. Zu sehen, wenn etwas funktionierte, erfreute mich jedes Mal von Neuem.

Ich habe durch diese Arbeit sehr viel gelernt und es war die Mühe wert, in dieses komplexe Thema einzutauchen.

Kapitel 7

Anhang

7.1 Mathematische Symbole

7.2 Literatur

Literatur

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven und J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, Jg. 574, Nr. 7779, S. 505–510, Okt. 2019, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5. Adresse: https://doi.org/10.1038/s41586-019-1666-5.

[2] A. Asfaw, L. Bello, Y. Ben-Haim, S. Bravyi, N. Bronn, L. Capelluto, A. C. Vazquez, J. Ceroni, R. Chen, A. Frisch, J. Gambetta, S. Garion, L. Gil, S. D. L. P. Gonzalez, F. Harkins, T. Imamichi, D. McKay, A. Mezzacapo, Z. Minev, R. Movassagh, G. Nannicni, P. Nation, A. Phan, M. Pistoia, A. Rattew, J. Schaefer, J. Shabani, J. Smolin, K. Temme, M. Tod, S. Wood und J. Wootton. (2020). "Learn Quantum Computation Using Qiskit," Adresse: http://community.qiskit.org/textbook.

[3] S. Asjad, "The RSA Algorithm," Dez. 2019.

[4] I. L. Chuang, N. Gershenfeld und M. Kubinec, "Experimental Implementation of Fast Quantum Searching," *Phys. Rev. Lett.*, Jg. 80, S. 3408–3411, 15 Apr. 1998. DOI: 10.1103/PhysRevLett.80.3408. Adresse: https://link.aps.org/doi/10.1103/PhysRevLett.80.3408.

[5] T. G. Draper, "Addition on a Quantum Computer," Sep. 1998. Adresse: https://cds.cern.ch/record/450434/files/0008033.pdf.

[6] G. Fischer, *Lineare Algebra*, 18 A. Springer Spektrum, 2013, ISBN: 978-3-658-03944-8.

[7] J. Gambetta, *IBM's Roadmap For Scaling Quantum Technology*, Sep. 2020. Adresse: https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/ (besucht am 28. 10. 2020).

[8] J. A. Jones und M. Mosca, "Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer," *The Journal of Chemical Physics*, Jg. 109, Nr. 5, S. 1648–1653, Aug. 1998, ISSN: 1089-7690. DOI: 10.1063/1.476739. Adresse: http://dx.doi.org/10.1063/1.476739.

[9] J. Kelly, *A Preview of Bristlecone, Google's New Quantum Processor*, März 2018. Adresse: https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html (besucht am 28. 10. 2020).

[10] M. A. Nielsen und I. L. Chuang, *Quantum Computation and Quantum Information*, 10. Aufl. Cambridge University Press, 2010, ISBN: 978-1-107-00217-3.

[11] E. Pednault, J. Gunnels, D. Maslov und J. Gambetta. (Okt. 2019). "On "Quantum Supremacy"," Adresse: https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/ (besucht am 25. 10. 2020).

[12] R. L. Rivest, A. Shamir und L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Commun. ACM*, Jg. 21, Nr. 2, S. 120–126, Feb. 1978, ISSN: 0001-0782. DOI: 10.1145/359340.359342. Adresse: https://doi.org/10.1145/359340.359342.

[13] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood und I. L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, Jg. 414, Nr. 6866, S. 883–887, Dez. 2001, ISSN: 1476-4687. DOI: 10.1038/414883a. Adresse: https://doi.org/10.1038/414883a.

[14] V. Vedral, A. Barenco und A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, Jg. 54, Nr. 1, S. 147–153, Juli 1996, ISSN: 1094-1622. DOI: 10.1103/physreva.54.147. Adresse: http://dx.doi.org/10.1103/PhysRevA.54.147.

7.3 Redlichkeitserklärung