itp cp

TU Graz

Bachelorarbeit

Institut für Theoretische Physik
Computational Physics

Quanten Fourier Transformation: Simulation eines Quantenalgorithmus in Matlab

Advisor:
Univ.-Prof. Dr.rer.nat. Enrico Arrigoni

Lena Gasser

Mat.Nr. 01513492

Sommersemester 2018

Inhaltsverzeichnis

1 Theoretische Grundlagen **4**
1.1 Qubits und Verschränkte Zustände 4
1.2 Qubit Gates 5
1.2.1 Single Qubit Gates 5
1.2.2 Multiple Qubit Gates 6
1.3 Quanten Algorithmen 10
1.3.1 Quanten-Adder 10
1.3.2 Quanten Fourier Transformation 11

2 Programm **15**
2.1 Qubit Gates 15
2.1.1 Not Gate 15
2.1.2 Hadamard Gate 17
2.1.3 Controlled Not Gate 18
2.1.4 Toffoli Gate 19
2.1.5 Controlled Phase Gate 20
2.2 Quanten Algorithmen 21
2.2.1 Quanten Adder 21
2.2.2 Quanten Fourier Transfomation 22

3 Zusammenfassung **28**

Abstract

Die Arbeit behandelt die sogennante Quanten Fourier Transformation, welche einen wichtigen Bestandteil des Shor Algorithmus zur Faktorisierung darstellt, sowie die darin vorkommenden wichtigen Quantengates. Der Shor Algorithmus ist ein Quantenalgorithmus der große Zahlen in ihre Primfaktoren in polynomineller Zeit zerlegen kann, was mit klassischen Computern nicht möglich ist. Eine effiziente Realisierung des Faktorisierungsverfahrens würde die Kryptografie vor neue Aufgaben stellen, da die üblichen Verschlüsselungscodes zur Datenübertragung in sinnvoller Zeit geknackt werden könnten. Die bisher bekannten klassischen Verfahren haben eine fast exponentiell wachsende Laufzeit, während der Algorithmus von Shor mit einer polynominelle Laufzeit arbeitet.
Die Arbeit besteht aus zwei Teilen. Im Ersten werden die theoretischen Grundlagen erklärt. Der zweite Teil behandelt das erstellte Programm, dessen Funktionsweise und die erhaltenen Ergebnisse. Zur Erzeugung der Simulation, dies stellt den Großteil der Arbeit dar, wurde Matlab verwendet.

Motivation

In der heutigen Zeit spielt die Datenverschlüsselung eine wichtige Rolle im Leben jedes Menschen. Es müssen beginnend von Passwörtern für Email Accounts bis hin zum Online Banking alle Daten geschützt werden. Dafür werden häufig RSA Kryptosysteme verwendet, welche auf dem Prinzip der Faktorisierung beruhen. Die Realisierung eines Quantencomputers und insbesondere des Shor Algorithmus würde diese Verschlüsselungen binnen kürzester Zeit lösen und die Daten freilegen.
Da die Behandlung des Shor Algorithmus den Umfang dieser Arbeit sprengen würde, wurde der Fokus auf die Quanten Fourier Transformation gelegt. Die Quanten Fourier Transformation ist der zeitintensivste Teil des Shor Algorithmus, nämlich genau jener, welcher klassisch einen exponentiellen Zeitaufwand benötigen würde. Um diese Transformation zu simulieren, wurden zuerst die wichtigsten Quantengates erarbeitet und in Matlab als aufrufbare Funktion erstellt.

1 Theoretische Grundlagen

Die in diesem Kapitel vorkommenden Inhalte beziehen sich vorwiegend auf das Buch 'Quantum Computation and Quantum Information' ([1]).

1.1 Qubits und Verschränkte Zustände

Der klassische Computer basiert auf Bits, diese haben den Wert 0 oder 1. Im Gegensatz dazu geht man bei Quantencomputern von einer Superposition dieser zwei Zustände aus. Dieser sogenannte Qubit sieht in der Dirac-Notation wie folgt aus:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

Die Betragsquatrate der komplexen Zahlen α und β können dabei als Wahrscheinlichkeit, dass der Zustand $|0\rangle$ oder $|1\rangle$ tatsächlich gemessen werden kann, aufgefasst werden. Dabei gilt die Bedingung $|\alpha|^2 + |\beta|^2 = 1$, da sich eine Gesamtwahrscheinlichkeit von 1 ergeben muss.

Für die Beschreibung von zwei oder mehreren Qubits benutzt man eine Tensorproduktnotation, welche speziell für die Erstellung des Programmes von großer Bedeutung ist.

$$|a\rangle\,|b\rangle = |ab\rangle = |a\rangle \otimes |b\rangle \tag{2}$$

Dieser Produktzustand wird in einem Quantencomputer Register genannt. Im Gegensatz zum klassischen Computer kann ein Quantenzustand aus einer linearen Superposition solcher Produktzustände bestehen. Ein einziger Zustand kann dadurch alle möglichen Bitkombinationen enthalten. Das ist die Eigenschaft die den Quantencomputer stark von dem Klassischen abhebt.

Wählt man einen Zustand

$$|\psi\rangle = \beta_{00}\,|00\rangle + \beta_{01}\,|01\rangle + \beta_{10}\,|10\rangle + \beta_{11}\,|11\rangle\,, \tag{3}$$

so kann der $2^n = 4$ dimensionale Zustandsvektor wie folgt dargestellt werden

$$|\psi\rangle = \begin{pmatrix} \beta_{00} \\ \beta_{01} \\ \beta_{10} \\ \beta_{11} \end{pmatrix}. \tag{4}$$

Eine Besonderheit des Rechnens mit Qubits sind die sogenannten verschränkten Zustände. Dabei handelt es sich um einen Zustand der nicht als Tensorprodukt von Zuständen der einzelnen Qubits gebildet werden kann. Im Gegensatz dazu ist es möglich, den unverschränkten Zustand als Kombination von Tensorprodukten darzustellen. Ein Beispiel für einen unverschränkten Zustand ist

$$\frac{1}{\sqrt{2}}(|10\rangle + |11\rangle) = |1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle). \tag{5}$$

Der Zustand

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{6}$$

kann nicht weiter aufgeteilt werden und ist deshalb ein verschränkter Zustand. Eine genauere Erklärung und Beschreibung des verschränkten Zustandes und dessen Wichtigkeit in der Quantenmechanik findet man im Kapitel 2.11 aus [3].

1.2 Qubit Gates

Analog zu den logischen Gates eines klassischen Computers gibt es verschiedene Quantengates (Qubit Gates), um ein Schaltwerk aufzubauen. Mathematisch versteht man unter solchen Quantengates eine unitäre Transformation des Zustandes.

1.2.1 Single Qubit Gates

Single Qubit Gates wirken jeweils auf ein Qubit. Da ein Zustand, bestehend aus einem Qubit, als zweidimensionaler Spaltenvektor dargestellt werden kann, sind die jeweiligen Transformationsmatrizen 2x2 Matrizen.

Das einfachste Quantengate ist das Not Gate. Es transformiert wie folgt:

$$\begin{aligned} N|0\rangle &\rightarrow |1\rangle \\ N|1\rangle &\rightarrow |0\rangle \end{aligned} \tag{7}$$

Die Matrix N entspricht dabei einer der vier Pauli Matrizen.

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{8}$$

Eines der wichtigesten Quantengates ist das Hadamard Gate. Dieses Gate führt eine Rotation um 90° in y-Richtung und eine Rotation um 180° in x-Richtung aus.

$$\begin{aligned} H|0\rangle &\rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ H|1\rangle &\rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned} \tag{9}$$

Die Hadamard Matrix H kann einfach dargestellt werden:

$$H = \frac{1}{\sqrt{(2)}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{10}$$

In einem Netzwerk, bestehend aus mehreren Gates, wird das Hadamard Gate durch das Symbol in Abb. 1 angegeben.

Abbildung 1: Symbol des Hadamard Gates; entnommen aus [1]

Es existieren noch viele weitere Single Qubit Gates, jedoch sind die zwei dargestellten Gates die mit der größten Bedeutung und Verwendung (vgl [1]).

1.2.2 Multiple Qubit Gates

Ähnlich wie bei der Erstellung von n-Qubit Zuständen (mit $n > 1$), können Multiple Qubit Gates durch Tensorprodukte unitärer 2x2 Matrizen erzeugt werden. In diesem Kapitel werden drei für die weitere Arbeit wichtige Gates beschrieben.

Ein sehr bekanntes 2-Qubit Gate ist das Controlled Not Gate, auch CNOT, welches in der Abb. 2 dargestellt ist.

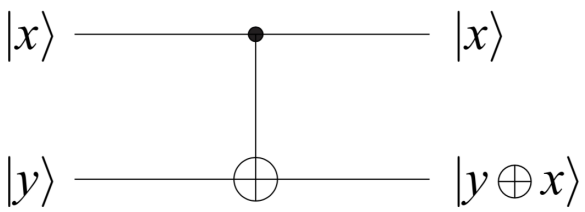

Abbildung 2: Symbol des Controlled Not Gates. Der schwarze Punkt gibt die Information über das control Qubit; entnommen aus [3]

Eines der zwei Input-Qubits wirkt dabei als control (eng. für Kontrolle) Qubit, das andere als target (eng. für Ziel) Qubit. Das ist so zu verstehen, dass wenn das control Qubit den Wert $|1\rangle$ hat, das target Qubit verändert wird. Hat es den Wert $|0\rangle$, so bleibt das target Qubit unverändert. Im Falle des ersten Qubit als control und dem zweiten Qubit als target wirkt das CNOT Gate wie folgt:

$$\begin{aligned} C_{NOT}\,|00\rangle &\rightarrow |00\rangle \\ C_{NOT}\,|01\rangle &\rightarrow |01\rangle \\ C_{NOT}\,|10\rangle &\rightarrow |11\rangle \\ C_{NOT}\,|11\rangle &\rightarrow |10\rangle \end{aligned} \tag{11}$$

Wie sich die Transformation auf die einzelnen Qubits auswirkt, ist mit der Tensorproduktdarstellung leicht ersichtlich ([2] [4]).

$$C_{NOT}\,|ab\rangle = P_1\,|a\rangle \otimes N\,|b\rangle + P_0\,|a\rangle \otimes \mathbb{1}\,|b\rangle \qquad mit \quad a, b \in 0, 1 \tag{12}$$

Das im vorherigen Kapitel beschrieben Not Gate N wird für diese Darstellung ebenso benötigt, wie die Einheitsmatrix $\mathbb{1}$ und die beiden Projektoren P_0, Projektor auf $|0\rangle$, und P_1, Projektor auf $|1\rangle$.

$$P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{13}$$

Die Matrixdarstellung des CNOT Gates ist, im Vergleich zur Tensorproduktdarstellung, sehr einfach.

$$C_{NOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Das Toffoli Gate T ist ein 3-Qubit Gate. Es ist dem CNOT Gate sehr ähnlich, jedoch hat es zwei control Qubits und ein target Qubit. Das target Qubit wird dann verändert, wenn beide control Qubits gleich $|1\rangle$ sind. Seine Funktion ist mit dem logischen NAND Gate eines klassischen Computers zu vergleichen. Durch Setzten bestimmter Qubits auf einen bestimmten Wert kann aus dem Toffoli Gate auch das logische NOT und AND Gate erstellt werden. Außerdem kann man durch das Aneinanderreihen von drei Toffoli Gates das logische OR Gate bekommen ([3]).

Für das erste und das zweite Qubit als control und das dritte Qubit als target ergiebt sich die Transformation

$$\begin{aligned} T\,|000\rangle &\rightarrow |000\rangle \\ T\,|001\rangle &\rightarrow |001\rangle \\ T\,|010\rangle &\rightarrow |010\rangle \\ T\,|011\rangle &\rightarrow |011\rangle \\ T\,|100\rangle &\rightarrow |100\rangle \\ T\,|101\rangle &\rightarrow |101\rangle \\ T\,|110\rangle &\rightarrow |111\rangle \\ T\,|111\rangle &\rightarrow |110\rangle\,. \end{aligned} \tag{14}$$

Für die weitere Arbeit ist die Tensorproduktdarstellung von sehr großer Bedeutung, da diese die Grundlage für das geschriebene Programm darstellt. Das Tofolli Gate in dieser Schreibweise ist in folgender Gleichung gegeben.

$$\begin{aligned} T\,|abc\rangle = {} & P_1\,|a\rangle \otimes P_1\,|b\rangle \otimes N\,|c\rangle + P_1\,|a\rangle \otimes P_0\,|b\rangle \otimes \mathbb{1}\,|c\rangle \\ & + P_0\,|a\rangle \otimes P_1\,|b\rangle \otimes \mathbb{1}\,|c\rangle + P_0\,|a\rangle \otimes P_0\,|b\rangle \otimes \mathbb{1}\,|c\rangle \\ & mit \quad a, b, c \in 0, 1 \end{aligned} \tag{15}$$

Die vorkommenden Operatoren N, P_0, P_1 und $\mathbb{1}$ sind aus der Gleichung (12) bekannt. Die Matrixdarstellung ist aus (14) abzulesen.

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Die grafische Darstellung des Toffoli Gates ist in der Abb. 3 gegeben. Dabei ist zu beachten, dass die beiden Variabeln $|a\rangle$ und $|b\rangle$ die control Qubits wiederspiegeln. Das target Qubit $|c\rangle$ wird auf der rechten Seiter des Gates, also nach der unitären Transformation, zu $c \oplus (a \wedge b)$. Das Symbol $\oplus$ ist ein logisches XOR und das Symbol $\wedge$ beschreibt das logische AND Gate.
Anhand der Grafik kann man schon erkennen, dass wenn man $|c\rangle = |0\rangle$ wählt, das Ergebnis nach der Transformation $a \wedge b$ ist und somit das Toffoli Gate nur mehr als logisches AND wirkt.

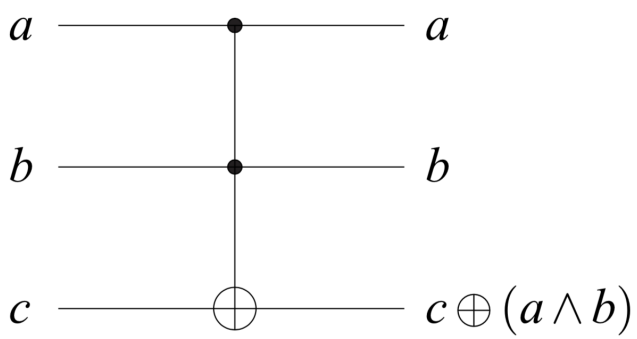

Abbildung 3: Symbol des Toffoli Gates. Die beiden control Qubits werden durch die schwarzen Punkte gekennzeichnet; entnommen aus [3]

Ein essentieller Teil des Algorithmus zur Berechnung der Quanten Fourier Transformation ist das Controlled Phase Gate (siehe Abb.4). Dabei ist die Idee dieselbe wie bei dem Controlled Not Gate: Ist das control Qubit $|0\rangle$ ändert sich nichts, ist es jedoch $|1\rangle$ so wird das Phase Gate R_k auf das target Qubit angewand (siehe [4]).

Es gehört erwähnt, dass es bei der Anwendung des Controlled Phase Gates egal ist, welches Qubit als target und welches als control gewählt wird, da beide Möglichkeiten dasselbe Ergebnis liefern. (vgl. [4]).

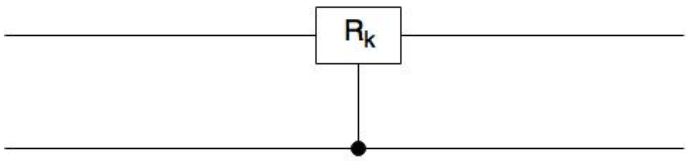

Abbildung 4: Symbol des Controlled Phase Gate. Der schwarze Punkt gibt das control Qubit an. (Grafik wurde selbst erstellt)

Das Phase Gate ist ein Single Qubit Gate und führt die Transformation

$$\begin{aligned} R_k\,|0\rangle &\rightarrow |0\rangle \\ R_k\,|1\rangle &\rightarrow e^{2\pi i/2^k}\,|1\rangle \end{aligned} \tag{16}$$

durch.
Die passende unitäre Matrix wird wie folgt dargestellt:

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}. \tag{17}$$

Die Wirkung des gesamten Controlled Phase Gates C_k auf einen 2-Qubit Zustand, mit dem ersten Qubit als target und dem zweiten als control, ist folgende:

$$\begin{aligned} C_k\,|00\rangle &\rightarrow |00\rangle \\ C_k\,|01\rangle &\rightarrow |01\rangle \\ C_k\,|10\rangle &\rightarrow |10\rangle \\ C_k\,|11\rangle &\rightarrow e^{2\pi i/2^k}\,|11\rangle \end{aligned} \tag{18}$$

Die Auswirkung auf die einzelnen Qubits ist in der Tensorproduktdarstellung

$$C_k\,|ab\rangle = P_1\,|b\rangle \otimes R_k\,|a\rangle + P_0\,|b\rangle \otimes \mathbb{1}\,|a\rangle \qquad mit \quad a,b \in 0,1 \tag{19}$$

gut erkennbar.
Die Matrixdarstellung ist daraus wieder gut abzulesen.

$$C_k = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2\pi i/2^k} \end{pmatrix}$$

1.3 Quanten Algorithmen

1.3.1 Quanten-Adder

Ein einfacherer Quanten Algorithmus ist der Quanten-Adder. Er besteht aus zwei Input- und zwei Output-Qubits. Zu Beginn werden die zwei Output-Qubits auf $|0\rangle$ gesetzt. Nach dem Durchgang des Netzwerkes bleiben die Input-Qubits erhalten. Das erste Output-Qubit, $|(X_1 + X_2)_{MOD_2}\rangle$, stellt die Bitsumme der zwei Input-Qubits $|X_1\rangle$ und $|X_2\rangle$ dar und das zweite Output-Qubit, $|X_1X_2\rangle$, wiederspiegelt die logische AND-Verknüpfung des Inputs.

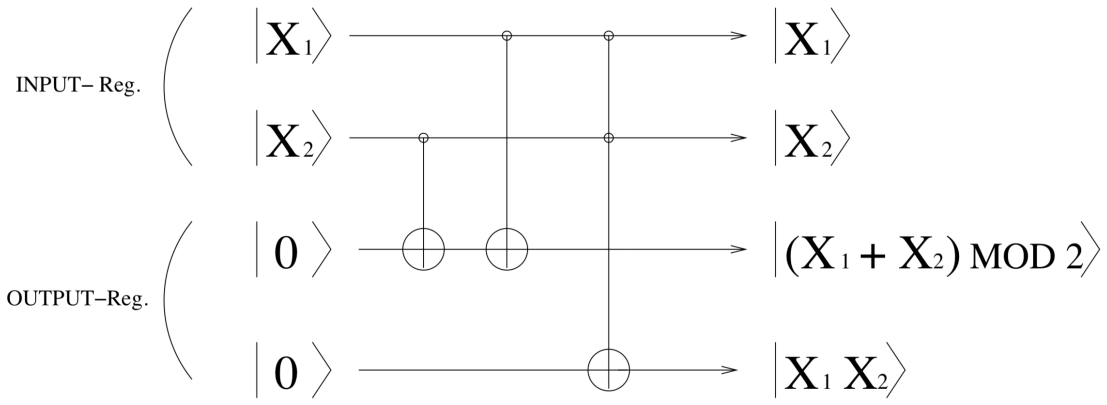

Abbildung 5: Netzwerk zur Erstellung eines Quanten-Adders; entnommen aus [2]

Wie in Abb. 5 ersichtlich werden dafür zwei Controlled Not Gates und ein Toffoli Gate benötigt. Allgemein kann die Transformation des Netzwerkes durch

$$|X_1\rangle \otimes |X_2\rangle \otimes |0\rangle \otimes |0\rangle \quad \rightarrow \quad |X_1\rangle \otimes |X_2\rangle \otimes |(X_1 + X_2)_{MOD2}\rangle \otimes |X_1 X_2\rangle \tag{20}$$

dargestellt werden. Die Variablen X_1 und X_2 können dabei die Werte 0 und 1 annehmen.

1.3.2 Quanten Fourier Transformation

Die Quanten Fourier Transformation ist ein zentraler Bestandteil des von Shor entwickelten Algorithmus zur Faktorisierung ganzer Zahlen. Die Faktorisierung wird für Verschlüsselungen, wie in der RSA-Kryptographie, genützt, da es mit den herkömmlichen Computern kein Verfahren gibt, das eine große Zahl in realisierbarer Zeit in deren Primfaktoren zerlegt. Die Existenz eines Quanten Computers macht dies jedoch möglich und somit wäre ein völlig neues Sicherheitssystem erforderlich. (vgl. [3])

Die Quanten Fourier Transformation hat ihren Ursprung in der Diskreten Fourier Transformation. Sie wird als linearer Operator F, der auf einen Zustand einer orthonormalen Basis, welche wiederum aus N Zuständen aufgebaut wird, wirkt, verstanden. Eine $N = 2^n$ dimensionale Basis enthält n Qubits. Im Falle eines Zustandes $|j\rangle$ kann dies

$$F\,|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N}\,|k\rangle \tag{21}$$

geschrieben werden.In dieser Gleichung werden für die Zustände $|j\rangle$ und $|k\rangle$ die jeweiligen Dezimaldarstellungen eingesetzt. Für zum Beispiel $|j\rangle = |011\rangle$ schreibt man $|3\rangle$, in diesem Fall wäre $N = 2^3 = 8$ und $|k\rangle$ würde die Werte $|0\rangle = |000\rangle$, $|1\rangle$, $|2\rangle$, $|3\rangle$, $|4\rangle$, $|5\rangle$, $|6\rangle$,$|7\rangle$ annehmen.

Geht man von einer Superposition mehrerer Zustände mit unterschiedlichen Vorfaktoren aus, ist auch das Ergebnis der Transformation ebenso eine Superposition und kann wie folgt dargestellt werden:

$$F \sum_{j=0}^{N-1} x_j |j\rangle = \frac{1}{\sqrt{N}} \sum_{k,j=0}^{N-1} x_j e^{2\pi i jk/N} |k\rangle \tag{22}$$

Für die folgenden Darstellungen werden einige formale Ausdrücke verwendet, um eine bessere Anschaulichkeit zu gewähren. Der Zustand $|j\rangle$ wird mit der Binärdarstellung $|j_1 j_2 ... j_n\rangle$ geschrieben, wobei j_i die einzelnen Qubits darstellen (zum Beispiel $|1\rangle = |01\rangle$ bei $n = 2$ Qubits). Außerdem wird die Notation $0.j_l j_{l+1} ... j_m$ zur Darstellung des Binärbruchs $j_l/2 + j_{l+1}/4 + ... + j_m/2^{m-l+1}$ gewählt (vgl. [1]).

Um einen Quantenalgorithmus zu finden der genau diese Transformation durchführt, beginnt man damit ein einzelnes Qubit ($n = 1$ und $N = 2$) zu betrachten.

$$F |j_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_1} |1\rangle) \tag{23}$$

Da der Vorfaktor für $j_1 = 1$ zu $e^{\pi i} = -1$ und für $j_1 = 0$ zu $e^0 = 1$ wird, kann man erkennen, dass dies dem Ergebnis eines Hadamard Gates entspricht.

Nimmt man nun n Qubits und wendet das Hadamard Gate auf das erste Qubit an erhält man

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_1} |1\rangle) |j_2 ... j_n\rangle . \tag{24}$$

Im nächsten Schritt wird das Controlled Phase Gate $n - 1$ Mal mit $k = 2...n$ verwendet ($R_2 ... R_n$). Das target Qubit ist dabei immer das erste Qubit j_1. Dies liefert den Zustand

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} |1\rangle) |j_2 ... j_n\rangle . \tag{25}$$

Nun wird der gleiche Vorgang für alle weitern Qubits $j_2 ... j_n$ durchgeführt, jedoch werden immer weniger Controlled Phase Gates benötigt. Bei j_2 wird R_2 bis R_{n-1} verwendet, bei j_3 die Controlled Phase Gates R_2 bis R_{n-2} und bei dem letzten Qubit j_n wird keine Controlled Phase Gate verwendet, sondern es wird nur mehr das Hadamard Gate angewandt. Das führt zu folgendem Ergebnis:

$$\frac{1}{\sqrt{2^n}}(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_2 ... j_n} |1\rangle)...(|0\rangle + e^{2\pi i 0.j_n} |1\rangle) \tag{26}$$

Um zur richtigen Produktdarstellung der in Gleichung (21) angegebenen Transformation zu kommen, müssen die erhaltenen Qubits noch vertauscht werden. Das erfolgt mit der Swap Transformation.

Bei der Swap Transformation werden die einzelnen Qubits in die umgekehrte Reihenfolge gebracht. Der Zustand $|j_1 j_2 ... j_n\rangle$ wird zu $|j_n j_{n-1} ... j_1\rangle$. Betrachtet man einen 4 Qubit Zustand

$$\begin{aligned}|\psi\rangle =&\alpha_1 |0000\rangle + \alpha_2 |0001\rangle + \alpha_3 |0010\rangle + \alpha_4 |0011\rangle + \alpha_5 |0100\rangle + \alpha_6 |0101\rangle + \alpha_7 |0110\rangle + \alpha_8 |0111\rangle + \\ &\alpha_9 |1000\rangle + \alpha_{10} |1001\rangle + \alpha_{11} |1010\rangle + \alpha_{12} |1011\rangle + \alpha_{13} |1100\rangle + \alpha_{14} |1101\rangle + \alpha_{15} |1110\rangle + \alpha_{16} |1111\rangle\end{aligned}$$

so wird dieser nach dem Swap zu

$$\begin{aligned}|\psi\rangle = &\alpha_1 |0000\rangle + \alpha_9 |0001\rangle + \alpha_5 |0010\rangle + \alpha_{13} |0011\rangle + \alpha_3 |0100\rangle + \alpha_{11} |0101\rangle + \alpha_7 |0110\rangle + \alpha_{15} |0111\rangle + \\ &\alpha_2 |1000\rangle + \alpha_{10} |1001\rangle + \alpha_6 |1010\rangle + \alpha_{14} |1011\rangle + \alpha_4 |1100\rangle + \alpha_{12} |1101\rangle + \alpha_8 |1110\rangle + \alpha_{16} |1111\rangle .\end{aligned}$$

Bildet man aus den Koeffizienten einen Spaltenvektor, ist das Resultat etwas offensichtlicher:

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \\ \alpha_8 \\ \alpha_9 \\ \alpha_{10} \\ \alpha_{11} \\ \alpha_{12} \\ \alpha_{13} \\ \alpha_{14} \\ \alpha_{15} \\ \alpha_{16} \end{pmatrix} \rightarrow \begin{pmatrix} \alpha_1 \\ \alpha_9 \\ \alpha_5 \\ \alpha_{13} \\ \alpha_3 \\ \alpha_{11} \\ \alpha_7 \\ \alpha_{15} \\ \alpha_2 \\ \alpha_{10} \\ \alpha_6 \\ \alpha_{14} \\ \alpha_4 \\ \alpha_{12} \\ \alpha_8 \\ \alpha_{16} \end{pmatrix} \tag{27}$$

Schließlich erhält man das gewünschte Ergebnis

$$\frac{1}{\sqrt{2^n}}(|0\rangle + e^{2\pi i 0.j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1} j_n} |1\rangle)...(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} |1\rangle). \tag{28}$$

Auf den ersten Blick kann die Gleichheit der Gleichungen (21) und (28) noch nicht festgestellt werden. Mit etwas Algebra kommt man von der Summendarstellung, Gleichung (21), recht schnell zur erhaltenen Produktdarstellung (vgl. [1]).

Das gefundene Netzwerk wird in der Abb. 6 dargestellt. In diesem Algorithmus ist das Swap Gate nicht eingezeichnet. Das in dieser Abbildung angegebene Ergebnis entspricht also dem der Gleichung (26).

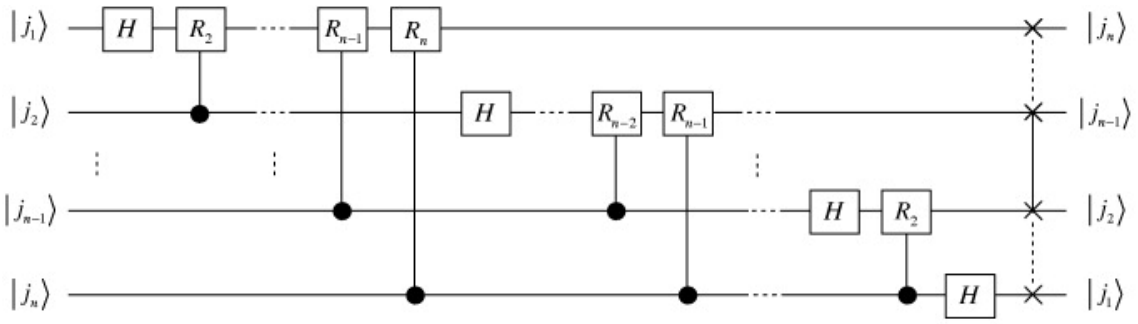

Abbildung 6: Netzwerk zur Durchführung der Fouriertransformation eines n-Qubit Zustandes; entnommen aus [1]

2 Programm

Das gesamte Programm wurde in Matlab erstellt, dementsprechend wird von der Bedeutung der Grundbegriffen wie Funktion, Skript, Input- und Outputwerte ausgegangen. Das Grundkonzept des Programmes ist es, dass nicht die Qubits beziehungsweise Zustände, sondern nur deren Koeffizienten als Spaltenvektoren übergeben werden. Die Reihenfolge der Koeffizienten ist dabei dieselbe der Dezimaldarstellung bei klassischen Bits. Der Zustand

$$|\psi\rangle = \alpha_1 |000\rangle + \alpha_2 |001\rangle + \alpha_3 |010\rangle + \alpha_4 |011\rangle + \alpha_5 |100\rangle + \alpha_6 |101\rangle + \alpha_7 |110\rangle + \alpha_8 |111\rangle$$

wird dem Program als 8-dimensionaler Spaltenvektor $\vec{q}$ übergeben:

$$\vec{q} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \\ \alpha_8 \end{pmatrix}$$

2.1 Qubit Gates

Die Quantengates werden über einzelne Funktionen aufgerufen. In den Funktionen werden sie über Tensorprodukte realisiert. Dies ermöglicht es, Input Qubits unterschiedlichster Dimensionen zu verwenden. Dabei wurde in der Funktion darauf geachtet, dass der Output Vektor die gleiche Größe wie der Input Vektor hat.

Um eine allgemeine und leichtere Nutzbarkeit zu gewährleisten, wird jedes Gate beziehungsweise dessen Funktion nach dem Aufruf „help 'Name der Funktion'“ auf Englisch genauer erklärt. Außerdem bekommt man über diesen Befehl auch einen Hinweis auf alle anderen erstellten Gates.

Bei fehlenden Input Werten wird entweder ein Error ausgegeben, oder man bekommt eine Warnung und ein Defaultwert wird gesetzt. Außerdem kommt es zu einem Error wenn der Input-Qubit kein Spaltenvekor ist, oder wenn die Werte für das control oder target Qubit nicht möglich sind.

2.1.1 Not Gate

Das Not Gate wird aus Tensorprodukten von 2x2 Matrizen gebildet. Diese Matrizen sind die Einheitsmatrix $\mathbb{1}$ und die Not Matrix N, welche in (8) gegeben ist. Die Not Transformation wirkt immer nur auf ein Qubit. Da man einem n-Qubit Zustand aus $n-1$

Tensorprodukten der einzelnen Qubits darstellen kann, wird die gesamte Transformationsmatrix auch aus $n-1$ Tensorprodukten von n 2x2 Matrizen gebildet. An der Stelle des zu transformierenden Qubits wird die N Matrix gesetzt, an allen anderen Positionen befindet sich die Einheitsmatrix $\mathbb{1}$, da sie keine Veränderungen erzeugt.
Wählt man den 2-Qubit Zustand $|\psi\rangle = |ab\rangle = |a\rangle \otimes |b\rangle$ und möchte das erste Qubit der Not Transformation unterziehen, so erzeugt das Programm die passende Matrix M.

$$M = N \otimes \mathbb{1}$$

In Matlab wird die Funktion mit folgendem Befehl aufgerufen:

$$[result\_qubit] = Not(qubit, target)$$

mit den Inputparametern

qubit	Der Koeffizientenvektor des gegebenen Qubits.
target	Der Index des zu transformierenden Qubits (beginnend bei 1). Falls dieser Input fehlt wird der Defaultwert auf 1 gesetzt.

Der Outputparameter *result_qubit* beinhaltet die Information der Koeffizienten nach der unitären Not Transformation.

Wählt man als Anfangszustand $|0110\rangle$ wird der Koeffizientenvektor *qubit* auf

$$qubit = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^\top \tag{29}$$

und gibt in Matlab dann den Code

$$Not(qubit, 1)$$

ein, so wird der Vektor

$$qubit = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}^\top$$

ausgegeben. In Matlab werden die Vektoren direkt als Spaltenvektoren dargestellt, hier wurde nur aus Platzgründen die Schreibweise mit transponiert gewählt. Aus dem Ergebnisvektor kann man ablesen, dass der Endzustand $|1110\rangle$ ist. Da dem Programm das erste Qubit als target übergeben wurde, ist das Ergebnis korrekt.

2.1.2 Hadamard Gate

Die Erstellung des Hadamard Gates erfolgt auf die gleiche Weise wie das Not Gate. An Stelle der Matrix N wird jedoch die Matrix zur Hadamard Transformation H, siehe (10), verwendet.

Um das Hadmarad Gate auf ein Qubit wirken zu lassen wird die Funktion

$$[result\_qubit] = Hadamard(qubit, target)$$

aufgerufen. Die Inputparameter sind

qubit	Der Koeffizientenvektor des gegebenen Qubits.
target	Der Index des zu transformierenden Qubits (beginnend bei 1). Bei fehlendem Input wird der Defaultwert auf 1 gesetzt.

Im Output Spaltenvektor *result_qubit* sind die Koeffizienten des Qubits nach Durchlauf der Funktion gespeichert.

Als Beispiel wird wieder der Anfangszustand $|0110\rangle$ gewählt, der Koeffizientenvektor *qubit* ist dabei in (29) gegeben. Matlab wird der Befehl

$$Hadamard(qubit, 1)$$

übergeben. Der resultierende Vektor ist dann

$$qubit = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0,707 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,707 & 0 \end{pmatrix}^\top$$

Der Endzustand ist also $1/\sqrt{2}\,(|0110\rangle + |1110\rangle)$. Dies entspricht genau dem erwateten Ergebnis der Hadamard Transformation des ersten Qubits.

2.1.3 Controlled Not Gate

Da das Controlled Not Gate selbst aus Tensorprodukten aufgebaut ist, ist das Programm komplexer zu erstellen, als jene der Single Qubit Gates. Es werden dazu vier 2x2 Matrizen benötigt. Diese sind die Not Matrix N, die Projektionsmatrix auf $|0\rangle$ P_0, die Projektionsmatrix auf $|1\rangle$ P_1 und die Einheitsmatrix $\mathbb{1}$ (die P_i Matrizen sind in (13) gegeben). Anhand der Tensorproduktdarstellung des CNOT Gates in (12) kann man erkennen, dass auf das control Qubit die Projektionsmatrizen und auf das target Qubit die Matrizen N und $\mathbb{1}$ wirken. Alle anderen Qubits werden ebenso mit der Einheitsmatrix multipliziert.
Betrachtet man den 3-Qubit Zustand $|\psi\rangle = |abc\rangle = |a\rangle \otimes |b\rangle \otimes |c\rangle$ und setzt als control das Qubit $|a\rangle$ und als target $|c\rangle$ so wird die Matrix

$$M = P_1 \otimes \mathbb{1} \otimes N + P_0 \otimes \mathbb{1} \otimes \mathbb{1}$$

erstellt.

Das Controlled Not Gate ist ein 2-Qubit Gate und hat dementsprechend auch einen weiteren Inputparameter. Die Funktion sowie deren Inputs sind durch

$$[result\_qubit] = Cnot(qubit, control, target)$$

und

qubit	Der Koeffizientenvektor des gegebenen Qubits.
control	Der Index kennzeichnet das control Qubit. Wird kein Wert übergeben, so wird der Defaultwert auf 1 gesetzt.
target	Der Index des zu transformierenden Qubits. Falls kein Input gegeben ist, wird der Defaultwert auf 2 gesetzt.

gegeben. Nach der Transformation wird der Koeffizientenvektor $result\_qubit$ ausgegeben.

Als Beispiel wird die Anwendung des Gates auf den Anfangszustand $|0110\rangle$ mit dem Koeffizientenvektor (29) gewählt. Anschließend ruft man die Funktion

$$Cnot(qubit, 2, 4)$$

auf, so erhält man den Vektor

$$qubit = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^{\top}$$

Man erhält als Endzustand $|0111\rangle$. In der Codezeile wurde als control das zweite und als target das vierte Qubit gewählt. Daraus folgt, dass das erste, das zweite und das dritte Qubit unverändert bleiben müssen. Das letzte Qubit wird auf Grund des auf $|1\rangle$ gesetzten zweiten Qubits von $|0\rangle$ auf $|1\rangle$ geändert. Somit stimmt das erhaltene Ergebnis.

2.1.4 Toffoli Gate

Für die Erstellung des Toffoli Gates werden die selben Matrizen wie für das CNOT Gate benötigt. Da das Toffoli Gate aber ohnehin ein 3-Qubit Gate ist, ist der Code komplizierter und aufwendiger als jener der CNOT Transformation. Es wirken wieder die Projektionsmatrizen auf die zwei control Qubits und die Matrizen N und $\mathbb{1}$ auf das target Qubit (siehe Gleichung (15)). Zur leichteren Veranschaulichung wird das Beispiel eines 4-Qubit Zustands $|\psi\rangle = |a\rangle \otimes |b\rangle \otimes |c\rangle \otimes |d\rangle$ mit dem control Qubit $|a\rangle$ und $|c\rangle$ und dem target Qubit $|d\rangle$ gewählt. Die Matrix, welche anschließend auf den gesamten Zustandsvektor wirkt, wird so gebildet

$$M = P_{\otimes}\mathbb{1} \otimes P_1 \otimes N + P_1 \otimes \mathbb{1} \otimes P_0 \otimes \mathbb{1} + P_0 \otimes \mathbb{1} \otimes P_1 \otimes \mathbb{1} + P_0 \otimes \mathbb{1} \otimes P_0 \otimes \mathbb{1}.$$

Die Funktion, welche das Toffoli Gate wiederspiegelt, ist wie folgt aufzurufen:

$$[result\_qubit] = Toffoli(qubit, control1, control2, target)$$

Die Inputparameter sind ähnlich zum Cnot Gate.

qubit	Der Koeffizientenvektor des gegebenen Qubits.
control1	Der Index kennzeichnet das erste control Qubit. Wird kein Wert übergeben, so wird der Defaultwert auf 1 gesetzt.
control2	Der Wert des zweiten control Qubits. Bei keinem Übergabewert wird der Defaultwert auf 2 gesetzt.
target	Der Index des zu transformierenden Qubits. Falls dieser Input fehlt, wird der Defaultwert auf 3 gesetzt.

Der Ergebnisvektor ist wieder in $result\_qubit$ gespeichert.

Wieder wird als Beispiel der Anfangszustand auf $|0110\rangle$ und der Koeffizientenvektor somit auf (29) gesetzt. Der Code

$$Toffoli(qubit, 2, 3, 4)$$

wird in Matlab eingegeben und als Resultat erhält man

$$qubit = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^\top$$

Der Endzustand ist $|0111\rangle$ und stimmt mit den Erwartungen überein. Als control wurden das zweite und dritte Qubit gewählt, als target wird das vierte Qubit gesetzt. Auf Grund des Wertes $|1\rangle$ beider control Qubits, wird das Vierte von $|0\rangle$ auf $|1\rangle$ geändert.

2.1.5 Controlled Phase Gate

Die Funktion des Controlled Phase Gates ist der des Cnot Gates sehr ähnlich. Es muss ledeglich die Matrix N durch die Phase Matrix R_k, gegeben in (17), ersetzt werden.

Das Controlled Phase Gate wird mit der Funktion

$$[result\_qubit] = Cphase(qubit, control, target)$$

ausgeführt. Die Inputparameter sind durch

qubit	Der Koeffizientenvektor des gegebenen Qubits.
control	Der Index des control Qubits. Bei fehlendem Inputwert wird der Defaultwert auf 2 gesetzt.
target	Der Index des zu transformierenden Qubits. Falls dieser Input fehlt, wird der Defaultwert auf 1 gesetzt.

gegeben.

Der Outputparameter ist $result\_qubit$ und beinhaltet die Information der Koeffizienten nach dem Controlled Phase Gate.

Als Beispiel wird wieder der Anfangszustand $|0110\rangle$ gewählt, der Koeffizientenvektor ist dabei (29). In Matlab wird der Befehl

$$Cphase(qubit, 2, 3)$$

eingegeben. Der resultierende Vektor ist

$$qubit = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^{\top}$$

Das Ergebnis entspricht also genau der Transformation des Phase Gates auf das target Qubit (hier das Dritte). Das ist korrekt, da sowohl das control als auch das target Qubit den Wert $|1\rangle$ haben.

2.2 Quanten Algorithmen

In diesem Kapitel werden zwei Programme zur Simulation zweier Quanten Netzwerke, der Quanten Adder und die Quanten Fourier Transformation, beschrieben. Die Algorithmen wurden in zwei seperaten Skripten erstellt. Dabei wird zu Beginn das 'Startqubit' erstellt. Danach werden nach der Reihe die benötigten Gates beziehungsweise deren Funktionen aufgerufen. Zum Schluss wird das letzte Output Qubit ausgegeben.

2.2.1 Quanten Adder

Wie bereits im Kapitel 1.3.1 beschrieben, werden in diesem Programm die Funktionen der Quantengates gemäß der Abb. 5 in folgender Reihenfolge aufgerufen:

$$\begin{aligned} Cnot(qubit, 2, 3) \\ Cnot(qubit, 1, 3) \\ Toffoli(qubit, 1, 2, 4) \end{aligned}$$

Bei ausführen dieses Skripts bei allen möglichen Inputwerten - diese betreffen nur die ersten zwei Qubits, da die Output-Qubits immer auf $|0\rangle$ gesetzt werden müssen - kommt man zu diesen Ergebnissen:

$$
\begin{aligned}
|0000\rangle &\rightarrow |0000\rangle \\
|0100\rangle &\rightarrow |0110\rangle \\
|1000\rangle &\rightarrow |1010\rangle \\
|1100\rangle &\rightarrow |1101\rangle
\end{aligned}
$$

Bei dem erhaltenen Ergebnis (rechts vom Pfeil) sind nur die beiden letzten Qubits verändert. Dies sind die vorher beschriebenen Output-Qubits, die ersten zwei Qubits bleiben unverändert, da sie ledeglich den Input wiederspiegeln. Betrachtet man das dritte Qubit, so ist, wie erhofft, die Bitsumme der ersten zwei Qubits erkennbar. Auch die logische AND Verknüpfung der zwei Inputwerte, dargestellt im vierten Qubit, liefert das gewünschte Ergebnis. Der erstellte Algorithmus erfüllt also seine Funktion und kann für weitere Rechnungen beziehungsweise Programme verwendet werden.

2.2.2 Quanten Fourier Transfomation

Im Allgemeinen werden Zustände mit n Qubits behandelt. Zur Vereinfachung wird das Programm für $n = 4$ Qubits realisiert. Das Netzwerk der Fourier Transformation für 4 Qubits ist in der Abb. 7 dargestellt.

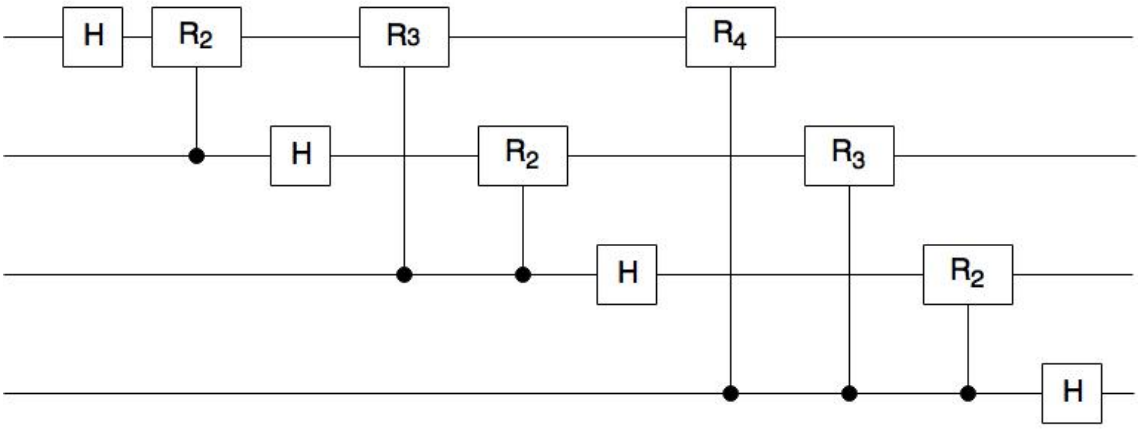

Abbildung 7: Netzwerk einer 4-Qubit Quanten Fourier Transformation bei fehlendem SWAP Gate. (Grafik wurde selbst erstellt)

Im Vergleich zur Abb. 6 sind in dieser Darstellung die Controlled Phase Gates mit demselben target Qubit nicht direkt hintereinander geschalten. Das macht jedoch keinen Unterschied, da die target Qubits noch keiner anderen Transformation unterzogen wurden. Betrachtet man zum Beispiel das R_3 Gate welches auf das erste Qubit, in der Grafik die oberste Linie, wirkt, so ist sofort ersichtlich, dass es völlig egal ist, ob es vor oder nach dem Hadamard Gate des zweiten Qubits angeführt wird, da die Hadamard Transformation weder das erste, noch das dritte Qubit beeinflusst.

Nun werden im Skript, nach erstellen eines beliebigen Ausgangszustandes, die Funktionen der in der Grafik angegebenen Gates von links nach rechts aufgerufen

$$
\begin{aligned}
Hadamard(qubit, 1)\\
Cphase(qubit, 2, 1)\\
Hadamard(qubit, 2)\\
Cphase(qubit, 3, 1)\\
Cphase(qubit, 3, 2)\\
Hadamard(qubit, 3)\\
Cphase(qubit, 4, 1)\\
Cphase(qubit, 4, 2)\\
Cphase(qubit, 4, 3)\\
Hadamard(qubit, 4).
\end{aligned}
$$

Um das korrekte Ergebnis zu erhalten, wird am Ende noch die Swap Transformation durchgeführt. Dazu wird eine Funktion aufgerufen in der, im Falle eines 4 Qubit Zustandes, die Matrix (30) auf den Koeffizientenvektor angewendet wird. In Matlab wird dieses Gate mit mehreren Cnot Gates realisiert.

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\tag{30}
$$

Anhand eines Beispiels wird die Funktionsfähigkeit des erstellten Programmes gezeigt. Nach Fertigstellung des Skripts wurden die Ergebnisse von vielen verschiedenen Inputwerten auf ihre Richtigkeit überprüft. In dieser Arbeit wird jedoch konkret nur die Richtigkeit eines Ergebnisses überprüft.

Der gewählte Anfangszustand ist $|1010\rangle$, im Skript wird der Inputvektor dementsprechend auf

$$\vec{q}_{start} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^\top \tag{31}$$

gesetzt. Der resultierende Koeffizientenvektor, nach durchlaufen des Programmes, ist folgender:

$$\vec{q}_{result} = \begin{pmatrix} 0.2500 + 0.0000i \\ -0.1768 - 0.1768i \\ 0.0000 + 0.2500i \\ 0.1768 - 0.1768i \\ -0.2500 + 0.0000i \\ 0.1768 + 0.1768i \\ -0.0000 - 0.2500i \\ -0.1768 + 0.1768i \\ 0.2500 + 0.0000i \\ -0.1768 - 0.1768i \\ 0.0000 + 0.2500i \\ 0.1768 - 0.1768i \\ -0.2500 + 0.0000i \\ 0.1768 + 0.1768i \\ -0.0000 - 0.2500i \\ -0.1768 + 0.1768i \end{pmatrix} \tag{32}$$

Nimmt man den gleichen Anfangszustand $|1010\rangle$, welcher der Dezimalzahl 10 entspricht, und setzt ihn in die Gleichung (21) ein, erhält man

$$F\,|10\rangle = \frac{1}{\sqrt{16}} \sum_{k=0}^{15} e^{2\pi i 10k/15}\,|k\rangle \tag{33}$$

da man für $n = 4$ und somit auch für $N = 2^n = 16$ einsetzen kann. Die Variable j ist auf Grund der Dezimaldarstellung von $|1010\rangle$ gleich 10.

Schreibt man die Summe aus und bestimmt man direkt den Wert der vorkommenden Exponentialfunktionen, kommt man zu folgendem Ergebnis:

$$\begin{aligned} F\,|10\rangle =& \frac{1}{4}\big[\,|0\rangle + (-0.7071 - 0.7071i)\,|1\rangle + i\,|2\rangle + (0.7071 - 0.7071i)\,|3\rangle \\ & - |4\rangle + (0.7071 + 0.7071i)\,|5\rangle - i\,|6\rangle + (-0.7071 + 0.7071)\,|7\rangle \\ & + |8\rangle + (-0.7071 - 0.7071i)\,|9\rangle + i\,|10\rangle + (0.7071 - 0.7071i)\,|11\rangle \\ & - |12\rangle + (0.7071 + 0.7071i)\,|13\rangle - i\,|14\rangle + (-0.7071 + 0.7071i)\,|15\rangle\,\big] \end{aligned} \tag{34}$$

Im nächsten Schritt wird der Vorfaktor ausmultipliziert und die Dezimaldarstellung wird in die Binärdarstellung umgeschrieben.

$$\begin{aligned} F\,|1010\rangle =& 0.2500\,|0000\rangle + (-0.1768 - 0.1768i)\,|0001\rangle + 0.2500i\,|0010\rangle + (0.1768 - 0.1768i)\,|0011\rangle \\ & -0.2500\,|0100\rangle + (0.1768 + 0.1768i)\,|0101\rangle - 0.2500i\,|0110\rangle + (-0.1768 + 0.1768)\,|0111\rangle \\ & +0.2500\,|1000\rangle + (-0.1768 - 0.1768i)\,|1001\rangle + 0.2500i\,|1010\rangle + (0.1768 - 0.1768i)\,|1011\rangle \\ & -0.2500\,|1100\rangle + (0.1768 + 0.1768i)\,|1101\rangle - 0.2500i\,|1110\rangle + (-0.1768 + 0.1768i)\,|1111\rangle\,\big] \end{aligned} \tag{35}$$

Von dieser Darstellung können die Koeffizienten abgelesen und mit den Vektoreinträgen (Gleichung (32)) in der gegebenen Reihnfolge verglichen werden. Das Ergebnis ist exakt dasselbe. Dadurch wurde gezeigt, dass das erstellte Programm die gewünschte Funktion erfüllt.

Ein anderes Beispiel, welches sich auch ohne längere Rechnung relativ leicht überprüfen lässt, ist die Wahl des Anfangszustandes $|0000\rangle$. Da alle control Qubits der diversen Controlled Phase Gates vor dem entsprechenden Hadamard Gate geschalten sind, ist das erwartete Ergebnis ledeglich eine Hadamard Transformation jedes Qubits. Das sollte im Endeffekt eine Superposition aller möglichen Zustände mit derselbelben Wahrscheinlichkeit, also demselben Vorfaktor, $(1/\sqrt{2})^4 = 0.25$ ergeben. Der erstellte Algorithmus liefert bei dem Startvektor $\vec{q}_{start}$ den resultierenden Koeffizientenvektor $\vec{q}_{result}$.

$$\vec{q}_{start} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \rightarrow \quad \vec{q}_{result} = \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} \tag{36}$$

Interessant ist außerdem das Ergebnis einer Superposition zweier Zustände $|j\rangle$ als Anfangsvektor. Zur rechnerischen Bestimmung des Resultats muss nun, wie in Gleichung (22) dargestellt, zusätzlich eine weitere Summe über j gelöst werden. Bei einem Input von $|\psi\rangle = 1/\sqrt{2}\,|0100\rangle + 1/\sqrt{2}\,|1100\rangle$ liefert das erstellte Programm den Koeffizientenvektor $\vec{q}_{result}$.

$$\vec{q}_{start} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1/\sqrt{2} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1/\sqrt{2} \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \rightarrow \quad \vec{q}_{result} = \begin{pmatrix} 0.3536 \\ 0 \\ -0.3536 \\ 0 \\ 0.3536 \\ 0 \\ -0.3536 \\ 0 \\ 0.3536 \\ 0 \\ -0.3536 \\ 0 \\ 0.3536 \\ 0 \\ -0.3536 \\ 0 \end{pmatrix} \tag{37}$$

In diesem Fall wurde der Zustand $|\psi\rangle$ so gewählt, dass nur das erste Qubit der beiden

Basiszustände der Superposition unterschiedlich ist. Die einzelnen Qubits sind also

$$
\begin{aligned}
|j_0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
|j_1\rangle &= |1\rangle \\
|j_2\rangle &= |0\rangle \\
|j_3\rangle &= |0\rangle \,.
\end{aligned}
$$

Die Bezeichnung der Qubits wurde dabei analog zur Abbildung 6 gewählt. Nach der Hadamardtransformation des ersten Qubits wird $|j_0\rangle$ zu $|0\rangle$. Nun ist erkennbar, dass die Controlled Phase Gates keine Transformation durchführen, da entweder die control Qubits $|0\rangle$ sind, oder, im Falle des R_2 Gates angewandt auf $|j_0\rangle$, zwar das control Qubit auf $|1\rangle$, das target Qubit aber gleich $|0\rangle$ ist. Somit wird nur mehr das Hadamard Gate auf die restlichen Qubits angewandt.
Auf Grund des Swap Gates ist der resultiernde Zustand $|j_3 j_2 j_1 j_0\rangle$, wobei sich die Werte von $|j_0\rangle$ - $|j_3\rangle$ durch die Transformationen, im Vergleich zum Anfangswert, verändert haben. Da $|j_0\rangle$ nun keine Superposition von $|0\rangle$ und $|1\rangle$ ist, enthält der Vektor $\vec{q}_{result}$ in jeder zweiten Zeile eine 0. Das Minuszeichen kommt aus der Hadamard Transformation von $|j_1\rangle$.

Der erstellte Code wäre auch für eine beliebige Zahl an Qubits anwendbar. Da das Programm aber an einem klassischen Computer läuft, ist die Speicherkapazität und die damit verbundene maximale Zahl an Qubits beschränkt. Ein n Qubit Zustand benötigt 2^n x 2^n Matrizen, daher kann n nicht allzu große Werte annehmen.

3 Zusammenfassung

Der programmierte Algorithmus führt eine 4-Qubit Quanten Fourier Transformation durch und liefert das erwartete Ergebnis. Auf Grund der Erstellung der einzelnen Quantengates mittels Tensorprodukten, sind diese auf n-Qubit Zustände anwendbar. Dadurch lässt sich das Programm der Fourier Transformation leicht für eine größere Anzahl an Qubits verwenden. Diese könnte man nutzen, um den Shor Algorithmus zur Faktorisierung ganzer Zahlen zu simulieren. Diese Simulation basierend auf einem klassischen Computer bedarf weiterhin exponentiellen Zeitaufwand. Durch die Nutzung eines Quantencomputers kann dies in eine polynominelle Laufzeit umgewandelt werden und für die Primfaktorenzerlegung verwendet werden. Da Methoden zur Datenverschlüsselung auf die Unlösbarkeit der Primfaktorenzerlegung einer großen Zahl beruhen, würde die Realisierung des Shor Algorithmus völlig neue Kryptographieverfahren erfordern.

Literatur

[1] Michael A.Nielson and Isaac L.Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 8 edition, 2005.

[2] Enrico Arrigoni. Quantencopmputer. Vorlesungsunterlage, 2009/2010.

[3] Matthias Homeister. *Quantum Computing verstehen*. Springer Verlag, 4 edition, 2015.

[4] Mikio Nakahara and Tetsuo Ohmi. *Quantum Computing*. Taylor and Francis Group, 4 edition, 2008.

[5] Jr. Samuel J. Lomonaco. A lecture on shor's quantum factoring algorithm. `https://arxiv.org/abs/quant-ph/0010034`, 2000.