Simulation eines Quantencomputers

Björn Butscher, Hendrik Weimer

Universität Stuttgart

22. März 2003

Inhaltsverzeichnis

1 Quantencomputing – Eine Einführung

Viele nützliche Internet-Anwendungen wie Internet-Banking, online Einkaufen oder vertrauliche E-Mails erfordern hohe Sicherheit. Diese wird erreicht, indem Verschlüsselungsverfahren eingesetzt werden, deren unbefugte Entschlüsselung bisher nicht in vertretbarer Zeit möglich ist. Allerdings wurde in letzter Zeit mit erhöhter Aufmerksamkeit die Entwicklung von Quantencomputern verfolgt, denn diese stellen eine Möglichkeit dar, alle bekannten Verschlüsselungsverfahren anzugreifen [1]. Quantencomputer können bestimmte Probleme wesentlich schneller als klassische Rechner lösen, da durch die Ausnutzung quantenmechanischer Effekte viele Berechnungen parallel durchgeführt werden können.

Doch wie funktioniert ein Quantencomputer genau? Klassische Computer kennen als kleinste Recheneinheit das Bit, welches die Zustände 0 und 1 annehmen kann. Beim Übergang in die Welt des Quantencomputings wird die klassische Vorstellung von diskreten, also eindeutigen Zuständen aufgegeben und erlaubt, dass sich eine Speichereinheit in beiden Zuständen gleichzeitig befindet. Man nennt die neue Speichereinheit Qubit. Ein Beispiel für eine experimentelle Realisierung eines Qubits ist die Polarisation des Lichts. So lässt sich horizontal polarisiertes Licht als 0 und vertikal polarisiertes Licht als 1 definieren. Liegt die Polarisationsachse des Lichts nicht in einer dieser beiden Achsen, so liegt eine Überlagerung von vertikaler und horizontaler Polarisation vor.

Bei einem Quantencomputer werden diese Überlagerungszustände *Superpositionen* genannt. Die Superposition stellt man als Linearkombination von Basiszuständen dar und schreibt den Zustand eines Qubits als Vektor:

$$\psi = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Dabei sind die angegebenen Vektoren die Basisvektoren (Basiszustände). Die Wahl der Basiszustände ist beliebig, jedoch wählt man meist ein orthonormales Basissystem. Ein derartiger Vektorraum wird auch Hilbertraum genannt. Im einfachsten Basissystem sind die beiden Basiszustände die beiden mögliche Zustände eines klassischen Bits:

$$0 \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle; \;\; 1 \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

Der Einfachheit halber schreibt man statt der kompletten Basisvektoren kurz $|0\rangle$ und $|1\rangle$. $\alpha, \beta \in \mathbb{C}$ sind die Wahrscheinlichkeitsamplituden des Qubits bezüglich des jeweiligen Basiszustands. Der Betrag des Amplitudenquadrats $|\alpha|^2$ ist die Wahrscheinlichkeit, dass das Qubit bei einer Messung im Zustand $|0\rangle$, also im klassischen Zustand 0, gefunden wird. Ebenso ist $|\beta|^2$ die Wahrscheinlichkeit, das Qubit bei einer Messung im Zustand $|1\rangle$ zu finden. Da das Qubit mit Sicherheit in einem der beiden Zustände ist, gilt $|\alpha|^2 + |\beta|^2 = 1$, also $||\psi|| = 1$. Der Zustandsvektor ist also ein Einheitsvektor.

Für die Darstellung von n Qubits ist ψ ein Vektor mit 2^n Basisvektoren, da sich das Quantensystem in einer Superposition aus allen klassischen n-Bit-Zuständen befinden kann. So schreibt man beispielsweise für eine gleichverteilte Superposition eines aus zwei Qubits bestehenden Quantensystems:

$$\psi = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$$

Daraus ergibt sich der sogenannte Quantenparallelismus, denn es ist damit möglich, Berechnungen nicht nur an einem klassischen Zustand, sondern an vielen klassischen Zuständen gleichzeitig vorzunehmen [2]. Man kann also eine Funktion $f(x)$ in nur einem Rechenschritt für mehrere x durchführen, was bei bestimmten Problemen zu einer enormen Verkürzung der Rechenzeit führt.

Um ein Modell für die Simulation eines Quantencomputers zu entwickeln, muss man die kohärente Entwicklung eines quantenmechanischen Zustands nachbilden, welche durch eine zeitabhängige Schrödinger-Gleichung beschrieben wird:

$$i\hbar\frac{\partial\psi(t)}{\partial t} = H\psi(t)$$

Der Hamilton-Operator H wird für ein ψ mit n Basisvektoren durch eine $n \times n$-Matrix angegeben und beschreibt die Dynamik des Systems. Man kann die Schrödinger-Gleichung auch etwas umformen:

$$\frac{\partial\psi(t)}{\psi(t)} = -\frac{iH}{\hbar}\partial t \quad \text{Integration nach } \partial t \text{ liefert} \quad \psi(t) = e^{-\frac{iHt}{\hbar}}\psi(0) = U(t)\psi(0)$$

Das heisst, dass es eine Matrix $U(t)$ gibt, die den Ausgangszustand $\psi(0)$ in den Endzustand $\psi(t)$ überführt. Eine zeitunabhängige logische Operation des Quantencomputers kann also durch eine Abbildung U dargestellt werden. Für weitere Operationen kann man einfach den Endzustand der ersten Operation als Anfangszustand der nächsten Operation nehmen und so eine quantenlogische Schaltung (Quantenalgorithmus) als Produkt solcher Abbildungen darstellen.

Zudem ist der Hamilton-Operator, und damit auch U, unitär, das heisst, es existiert eine andere zu U inverse Matrix U^{-1}, die den Endzustand wieder in den Ausgangszustand zurückführt. Dies hat zur Folge, dass Quantencomputing reversibel sein muss, man kann also nicht einfach ein Qubit setzen oder löschen. Es gibt einige reversible quantenlogische Operationen (Gatter), die sich in einem Quantencomputer realisieren lassen. *Elementare Gatter* sind das NOT, das Controlled-NOT, das Hadamard und die Phasenverschiebung. Jede beliebige unitäre Operation U lässt sich als Produkt der elementaren Gatter darstellen, man bezeichnet die elementaren Gatter daher als *universal* [3].

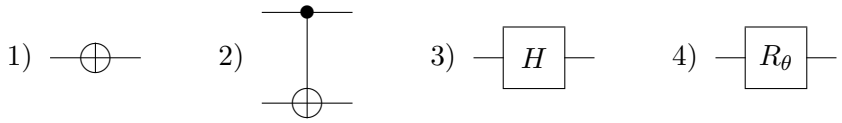

Abbildung 1: Symbole für NOT, Controlled-NOT, Hadamard und die Phasenverschiebung

Das NOT-Gatter (1) vertauscht die Wahrscheinlichkeitsamplituden der Zustände, bei denen das betreffende Qubit gesetzt bzw. gelöscht ist. Das Controlled-NOT (2) macht dasselbe mit dem zweiten Qubit, falls das erste Qubit gesetzt ist, ansonsten wird das System nicht verändert. Dieses Gatter ist also im Gegensatz zu den anderen ein 2-Qubit-Gatter. Während diese beiden Gatter auch klassisch implementierbar sind, findet man für das Hadamard-Gatter (3) und die Phasenverschiebung (4) keine Entsprechung. Ersteres dient dazu, Superpositionen zu erstellen und aufzuheben, während letztere die Phase,

also das Verhältnis von Real- und Imaginärteil der Amplituden verändert. In *Abb. 1* ist ein Formalismus zur Darstellung dieser Gatter aufgezeigt. Die waagerechten Linien geben dabei die Qubits an, auf die das Gatter angewendet wird. Für alle elementaren Gatter lässt sich auch die Operationsmatrix U angeben:

$$U_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad U_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad U_3 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \quad U_4 = \begin{pmatrix} e^{i\theta} & 0 \\ 0 & e^{-i\theta} \end{pmatrix}$$

Interessant wird es, wenn man ein Hadamard und ein Controlled-NOT kombiniert. Ein 2-Qubit-System soll sich zunächst im Zustand $\psi = 1\,|00\rangle$ befinden. Dann wird zunächst ein Hadamard auf das zweite Qubit durchgeführt. Es ergibt sich danach der neue Zustand $\psi' = \frac{1}{2}\sqrt{2}\,|00\rangle + \frac{1}{2}\sqrt{2}\,|01\rangle$. Nach der Anwendung eines Controlled-NOTs erhält man $\psi'' = \frac{1}{2}\sqrt{2}\,|00\rangle + \frac{1}{2}\sqrt{2}\,|11\rangle$. Das erstaunliche an diesem Zustand ist, dass die zwei Qubits nun aneinander gekoppelt sind, obwohl sie räumlich voneinander getrennt sind. Solche Zustände werden als *verschränkt* bezeichnet.

2 Simulation eines Quantencomputers

Gegenwärtig lassen sich Quantencomputer experimentell nur in sehr beschränktem Umfang mit wenigen Qubits realisieren. Die bisher umfangreichste Berechnung mit einem Quantencomputer wurde im Dezember 2001 mit 7 Qubits realisiert [4]. Um auch komplexere Quantenalgorithmen entwickeln und analysieren zu können, ist es unerlässlich, Simulatoren einzusetzen. Die Simulation eines Quantencomputers auf einem klassischen Computer soll hier beschrieben und eine Implementierung eines solchen Simulators vorgestellt werden. Die Anforderungen an einen solchen Simulator lassen sich zusammenfassen:

- Der Simulator muss universell sein, das heisst beliebige Quantenalgorithmen, die auf einem echten Quantencomputer lauffähig wären, müssen realisierbar sein.
- Die Simulation muss die physikalischen Gesetzmäßigkeiten, denen ein Quantencomputer unterliegt, möglichst genau abbilden.
- Die Ausführungsgeschwindigkeit (Performance) des Simulators muss möglichst hoch sein, damit auch komplexere Probleme mit akzeptablem Zeit- und Hardwareaufwand untersucht werden können.
- Sowohl die Ergebnisse als auch die Vorgehensweise bei der Simulation müssen nachvollziehbar sein, um eine wissenschaftliche Betrachtung zu ermöglichen. Das bedeutet, dass sämtliche Bestandteile des Simulators offen gelegt sein müssen.

Bislang existiert kein Simulator, der diese zentralen Punkte vollständig abdeckt [5]. Bisherige Simulatoren verfügen entweder über ein unzureichendes quantenmechanisches Modell oder benötigen schon für einfache Berechnungen einen enormen Zeit- und Hardwareaufwand. Mit der von uns entwickelten Lösung ist es erstmals möglich, beliebige

Quantenalgorithmen in einer genauen Simulation ohne großen Aufwand mit einem frei verfügbaren Simulator zu untersuchen, womit der Simulator die genannten Anforderungen erfüllt.

Für die Simulation eines Quantencomputers benötigt man drei grundlegende Komponenten. Man muss den Speicher des Quantencomputers verwalten und diesen Speicher durch die Anwendung von Gattern verändern können. Zudem muss es möglich sein, durch eine Messung den quantenmechanischen Zustand des Speichers in einen klassischen Zustand zu überführen.

2.1 Implementierung des Speichers

Der Speicher des Quantencomputers, das Quantenregister ψ, lässt sich als Zustandsvektor schreiben, wobei die Basisvektoren jeder mögliche klassische Zustand sind, den das System einnehmen kann. Für ein n-Qubit-Register bedeutet dies:

$$\psi = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle + \cdots + \alpha_{2^n-1}|2^n-1\rangle \qquad (\alpha_j \in \mathbb{C})$$

In allen komplexeren Quantenalgorithmen werden zusätzliche Qubits für Zwischen- und Endergebnisse benötigt. Diese werden jedoch nicht am Anfang in eine Superposition gebracht, was zur Folge hat, dass während der gesamten Berechnung sehr viele Basiszustände eine Wahrscheinlichkeitsamplitude von Null besitzen. Daher lässt sich effiziente Datenstruktur für das Register erreichen, indem man die Basiszustände, deren Wahrscheinlichkeitsamplituden Null sind, weglässt. Diese Vereinfachung erfordert aber, dass sowohl die Bezeichnungen der einzelnen Basiszustände als auch die Gesamtzahl der von Null verschiedenen Basiszustände gespeichert werden müssen. In der Praxis ist es zudem hilfreich, wenn man direkt auf die Anzahl der Qubits des Registers zugreifen kann. Man kann die Datenstruktur für das Register formulieren, welche sich als Feld mit den entsprechenden Kopfdaten im Simulator speichern lässt:

Anzahl der Qubits			
Anzahl der Basiszustände mit Amplituden ungleich Null (N)			
1. Basiszustand	*2. Basiszustand*	$\cdots$	*N-ter Basiszustand*
Amplitude	Amplitude	$\cdots$	Amplitude
Bezeichnung	Bezeichnung	$\cdots$	Bezeichnung

Abbildung 2: Datenstruktur für ψ im Simulator

2.2 Implementierung der Gatter

Anfangs wurde gezeigt, dass sich die Anwendung eines Gatters im Quantencomputer als Matrixprodukt aus der Operationsmatrix U des Gatters und dem Zustandsvektor ψ des Registers formulieren lässt. Nach Durchlaufen eines Gatters erhält man den neuen Zustand $\psi' = U\psi$. Allerdings ist das Matrixprodukt in der Praxis aus zwei Gründen nicht sinnvoll realisierbar. Zum einen ist es schwierig, aus einer Operationsmatrix für mehr als ein Qubit

eine Operationsmatrix für sämtliche Qubits des Registers zu finden. Zum anderen sind sehr viele Elemente der Operationsmatrix Null, weshalb andere Verfahrensweisen deutlich schneller zum gewünschten Ergebnis führen.

Deswegen betrachtet man die Auswirkungen eines Gatters auf einen einzelnen Basiszustand und führt dies für alle Basiszustände nacheinander aus. Besonders einfach wird es, wenn die Operationsmatrix eine Permutationsmatrix ist, da die Basiszustände lediglich umbenannt werden müssen. Als Beispiel sei hier die Anwendung eines NOT-Gatters auf das 7. Bit[1] (Operations-Bit) gezeigt:

$$\alpha|0\rangle + \beta|1\rangle + \cdots \xrightarrow{NOT(7)} \alpha|128\rangle + \beta|129\rangle + \cdots$$

Es reicht also aus, bei jeder Bezeichnung der Basiszustände das 7. Bit zu invertieren. Analog lässt sich auch das Controlled-NOT-Gatter implementieren. Für das Phasenverschiebungs-Gatter kann ähnlich verfahren werden, die Amplituden der Basiszustände müssen mit einer komplexen Zahl multipliziert werden. Diese Zahl ist abhängig davon, ob in der Bezeichnung des Basiszustandes das Operations-Bit gesetzt ist oder nicht.

Schwieriger wird es, wenn eine Zeile oder Spalte der Operationsmatrix mehrere Werte enthält. Dazu betrachtet man für ein beliebiges 1-bit-Gatter die Basiszustände stets paarweise. Man beginnt mit dem ersten Basiszustand im Speicher des Simulators und sucht denjenigen Basiszustand, bei dem nur das Operationsbit invertiert, die Bitmaske ansonsten jedoch identisch ist. Diese Suche lässt sich erheblich vereinfachen, indem man zu Beginn der Operation eine Hashtabelle aufstellt, in der sämtliche Basiszustände des Speicherfelds indiziert sind. Der Schlüssel sind die Basiszustände, die im Register gespeichert sind, und eine multiplikative Hashfunktion sorgt für möglichst wenig Kollisionen [6].

Hat man ein solches Paar an Basiszuständen gefunden, lässt sich der Zustand nach der Operation für diese beiden Basiszustände berechnen:

$$\psi' = U\psi = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a\alpha + b\beta \\ c\alpha + d\beta \end{pmatrix}$$

α ist die Wahrscheinlichkeitsamplitude des Basiszustandes $|x\rangle$, bei dessen binärer Repräsentation das Operations-Bit gelöscht ist. Analog dazu ist β ist die Amplitude des Basiszustandes $|y\rangle$, den man erhält, indem man das Operations-Bit von x setzt. Wenn man diese Vorgehensweise für alle Basiszustände im Speicher des Simulators anwendet, wird das gewünschte Gatter simuliert.

Somit wurde eine Methode entwickelt, mit der man jedes beliebige 1-Qubit-Gatter simulieren kann. Damit kann im Prinzip jeder Quantenalgorithmus untersucht werden.

2.3 Messungen

Ein Quantenalgorithmus überführt einen Anfangszustand in einen Endzustand ψ. Um ein Ergebnis zu erhalten, muss eine Messung des Quantenregisters durchgeführt werden. Da die Messung selbst ein Eingriff in das Quantensystem ist, erhält man nicht den quantenmechanischen Zustand ψ, sondern einen klassischen Zustand. Die Wahrscheinlichkeit, einen beliebigen klassischen Zustand j zu messen, beträgt $p = |\alpha_j|^2$, wobei α_j die Wahrscheinlichkeitsamplitude des Basiszustandes $|j\rangle$ ist.

[1] Die Zählweise beginnt bei 0, welche das kleinste Bit angibt (least significant bit)

Im Simulator wurde die Messung des Registers realisiert, indem zunächst eine gleichverteilte Zufallszahl f mit $0 \leq f < 1$ erzeugt wird. Danach wird beginnend beim ersten Basiszustand des Speicherfelds das Amplitudenquadrat $|\alpha_j|^2$ jedes nachfolgend gespeicherten Basiszustandes $|j\rangle$ von f abgezogen, bis $|\alpha_j|^2 \leq f$ ist. Als Ergebnis ergibt sich dann j.

Manchmal ist es aber erforderlich, nicht das gesamte Register zu messen, sondern nur einzelne Qubits. Die Messung des k-ten Qubits wird in der Simulation erreicht, indem zunächst das Ergebnis der Messung (0 oder 1) bestimmt wird. Dazu wird die Wahrscheinlichkeit berechnet, dass ψ ein Zustand ist, bei dessen binärer Repräsentation das k-te Bit gelöscht ist:

$$p = \sum_j \{|\alpha_j|^2 \mid (j \text{ AND } 2^k) = 0\}$$

Danach wird die Zufallszahl f wie vorher initialisiert, und f mit p verglichen. Ist $f > p$, lautet das Ergebnis der Messung 1, ansonsten 0. Die Messung eines Qubits sorgt aber dafür, dass die Verschränktheit von ψ teilweise aufgehoben und die Dimension des Hilbertraums halbiert wird. Konkret heisst dies, dass das Quantenregister ein Qubit weniger an Information enthält. Die Reduzierung des Registers wird erreicht, indem die Wahrscheinlichkeitsamplituden der Basiszustände, deren k-tes Bit invers zum erhaltenen Ergebnis ist, auf Null gesetzt werden. Da ψ ein Einheitsvektor ist, muss dieser noch anschließend normiert werden.

2.4 Der Faktorisierungsalgorithmus von Shor

2.4.1 Grundlagen

Schon in der Antike war bekannt, dass sich jede natürliche Zahl eindeutig in ein Produkt aus Primzahlen zerlegen lässt. Seitdem haben sich viele Mathematiker mit der Frage beschäftigt, wie sich diese Primfaktoren berechnen lassen. Praktische Bedeutung hat die Faktorisierung in der Kryptoanalyse, denn der RSA-Algorithmus, eines der am weitesten verbreiteten Verschlüsselungsverfahren, basiert darauf, dass sich die Primfaktoren einer Zahl nicht effizient berechnen lassen. Die Rechenzeit für alle bekannten klassischen Algorithmen wächst exponentiell mit der Eingabegröße, dem Logarithmus der zu faktorisierenden Zahl n. Der beste bekannte klassische Algorithmus (Number Field Sieve) besitzt die Laufzeit $O(e^{1.9(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}})$. Die O-Notation gibt bis auf einen konstanten Faktor an, welche Laufzeit ein Algorithmus im ungünstigsten Fall benötigt.

Daher war es ein Durchbruch, als Peter Shor 1994 den ersten Algorithmus vorstellte, der dieses Problem in polynomialer Zeit löst [7]. Dieser Algorithmus benötigt einen Quantencomputer und wurde daher als Beispiel für einen Quantenalgorithmus in unserem Simulator implementiert.

Der Algorithmus von Shor berechnet die Periode r der Abbildung $a \mapsto x^a \bmod n$, dem Rest aus der Division von x^a durch n, wobei $x < n$ beliebig ist. Es soll die Annahme gelten, dass n wie im RSA-Verfahren aus genau zwei Primfaktoren p und q besteht. Falls n aus mehr Primfaktoren besteht, können diese nach und nach abgespalten werden. Mit dieser Annahme kann man versuchen, die Faktoren zu bestimmen [8]:

$$(x^{\frac{r}{2}} - 1)(x^{\frac{r}{2}} + 1) = x^r - 1 = kn$$

kn ist ein Vielfaches von n, daher ergibt sich über den größten gemeinsamen Teiler $p = \mathrm{ggT}(x^{\frac{r}{2}} - 1, n)$ und $q = \mathrm{ggT}(x^{\frac{r}{2}} + 1, n)$. Allerdings sieht man, dass dieser Ansatz nicht immer zum Ziel führt, zum Beispiel wenn r ungerade ist. In diesem Fall wählt man einfach ein anderes x und wiederholt die Berechnung.

Zur Faktorisierung wird zunächst das Quantenregister in eine Superposition aus $q = 2^m$ Basiszuständen gebracht, wobei m die nächstgrößere Ganzzahl des Zweierlogarithmus von n^2 ist ($m = \lceil \log_2 n^2 \rceil$). Danach wird für sämtliche a gleichzeitig der zugehörige Wert berechnet, was durch die Quantenparallelisierung möglich ist. Aus den erhaltenen Werten wird über eine Quanten-Fourier-Transformation (QFT) die Periode bestimmt.

2.4.2 $x^a \bmod n$

Das Verfahren zur Berechnung von $x^a \bmod n$ basiert im wesentlichen auf dem von Beckman et al. entwickelten Algorithmus [9]. Dieser stellt x und n durch klassische Bits dar und speichert nur a in einem Quantenregister. Der Aufbau der Schaltelemente des Quantencomputers kann somit durch einen klassischen Computer in Abhängigkeit von x und n gesteuert werden. Diese Vereinfachung reduziert den Aufwand an Quanten-Schaltelementen und Qubits erheblich und führt somit zu einer Geschwindigkeitssteigerung.

Der Speicher des Quatencomputers ist in mehrere Register aufgeteilt. Dabei unterscheidet man zwischen dem nur temporär benötigten *Scratch-Space* und dem dauerhaft benötigten *Working-Space*. Ausgehend von einem auf Null initialisierten Speicher bringt man den Working-Space aus $M = \lceil \log_2 n^2 \rceil$ Qubits mittels Hadamard-Gatter in eine gleichverteilte Superposition:

$$|a\rangle = \frac{1}{\sqrt{2^M}} \sum_{j=0}^{2^M-1} |j\rangle$$

Für die einzelnen Schritte zur Berechnung von $x^a \bmod n$ werden drei zusätzliche Register mit je $m = \lceil \log_2 n \rceil$ Qubits für den Scratch-Space benötigt. Zwei weitere Qubits sind zur Ablaufsteuerung erforderlich. Für die Faktorisierung von n benötigt man also $M + 3m + 2$ Qubits.

Die Potenzierung wird durch eine wiederholte Multiplikation realisiert. Die Multiplikation wird wiederum auf eine vielfache Addition zurückgeführt. Die Addition wird mittels einer Quantenvolladdierer-Schaltung vollzogen, wobei das Ergebnis modulo n in den Scratch-Space geschrieben wird. Nach jedem Durchlauf der Multiplikation wird der Scratch-Space wieder auf Null gesetzt, damit dieser weiterhin benutzt werden kann. Da Quantencomputing reversibel ist, erreicht man dieses, indem man die Umkehrfunktion der Multiplikation durchführt.

Die Umkehrfunktion wird realisiert, indem zum einen alle Gatter in umgekehrter Reihenfolge durchlaufen werden und zum anderen durch ihre inversen Operationen ersetzt werden. Da alle in dieser Schaltung verwendeten Gatter selbstinvers sind, muss man lediglich die Umkehrung der Reihenfolge durchführen. Nachdem sämtliche Durchläufe der Multiplikation abgeschlossen sind, ist der Zustand des Speichers:

$$|a\rangle|x^a \bmod n\rangle$$

Im Simulator wurde dieses Verfahren mit $O((\ln n)^3)$ elementaren Gattern realisiert. Daraus ergibt sich für die Laufzeit der gesamten Faktorisierung ebenfalls $O((\ln n)^3)$.

2.4.3 Quanten-Fourier-Transformation

Nachdem $x^a \bmod n$ berechnet wurde, muss die Periode dieser Abbildung bestimmt werden. Dies wird über eine Quanten-Fourier-Transformation (QFT) erzielt. Generell ist eine Fourier-Transformation eine Methode, die eine beliebige Abbildung in eine frequenzabhängige Abbildung umwandelt. Aus einer Abbildung mit der Periode r entsteht eine neue Abbildung, die nur bei Vielfachen der Frequenz $\frac{1}{r}$ von Null verschieden ist.

Die QFT ist die Umsetzung der Diskreten Fourier-Transformation (DFT) für einen Quantencomputer und lässt sich mit folgender Vorschrift implementieren [10]:

$$\sum_{a=0}^{q-1} f(a)|a\rangle \rightarrow \sum_{b=0}^{q-1} \bar{f}(b)|b\rangle \quad \text{mit} \quad \bar{f}(b) = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} e^{\frac{2\pi i a b}{q}} f(a)$$

Diese Vorschrift lässt sich mit $O((\ln n)^2)$ elementaren Gattern realisieren [11]. Nach der QFT wird das Quantenregister gemessen.

2.4.4 Klassische Nachbearbeitung

Aus dem gemessenen Ergebnis kann man noch nicht direkt die gesuchte Periode ablesen, allerdings erfordern die weiteren Schritte keinen Quantencomputer mehr, da sie effizient klassisch durchführbar sind. Durch die Messung des Quantenregisters erhält man einen ganzzahligen Wert c, aus welchem sich die gesuchte Periode r berechnen lässt. Sei q die Anzahl der möglichen Basiszustände des Registers, dann gilt für den Quotienten $\frac{c}{q}$, dass er in der Nähe eines Vielfachen der Frequenz $\frac{d}{r}$ liegt. Der Quotient $\frac{d}{r}$ lässt sich in polynomialer Zeit über Kettenbruchbildung annähern. Die Abbruchbedingung für diese Annäherung ist erfüllt, wenn [7]:

$$\left| \frac{c}{q} - \frac{d}{r} \right| \leq \frac{1}{2q}$$

Da r nun bestimmt wurde, kann auch die Primfaktorzerlegung von n berechnet werden. Für die Berechnung des größten gemeinsamen Teilers wurde das Verfahren nach Euklid verwendet, da es ebenfalls in polynomialer Zeit ($O(\ln n)$) zum Ziel führt [12].

2.5 Dekohärenz und Fehlerkorrektur

2.5.1 Dekohärenz und andere Fehlerquellen

Der bisherige Ansatz zur Simulation eines Quantencomputers geht davon aus, dass das zu simulierende System perfekt ist. In der Praxis hingegen treten Fehler auf, die zum einen durch die experimentelle Realisierung des Quantencomputers verursacht werden, zum anderen gibt es aber auch Fehler, die auf die generelle Arbeitsweise eines Quantencomputers zurückzuführen sind.

Ein Beispiel für einen implementationsabhängigen Fehler wäre die Anwendung eines Laserpulses, dessen Frequenz und Zeitdauer nur begrenzt genau sind. Solche Fehler lassen sich mit einer leicht modifizierten Operationsmatrix simulieren. Allerdings kann man davon ausgehen, dass sich diese Art von Fehler im Mittel ausgleichen und daher nicht weiter ins Ergebnis einfließen [13].

Wesentlich signifikanter sind die Fehler, die unmittelbar mit dem Quantencomputing an sich verbunden sind. Das Hauptproblem ist hierbei die Wechselwirkung mit der Umgebung, welche dafür sorgt, dass das Quantensystem den angestrebten Zustand verlässt. Dieses Phänomen wird als *Dekohärenz* bezeichnet und ist das größte Hindernis auf dem Weg, einen praktikablen Quantencomputer zu entwerfen [14]. Diese Wechselwirkung mit der Umgebung lässt sich durch eine zufällige Phasenverschiebung auf jedes Qubit des Registers simulieren [15]. Der Winkel θ der Phasenverschiebung ist dabei normalverteilt mit der Varianz λ. Da aber ein Computer nur gleichverteilte Zufallswerte zur Verfügung stellt, werden für die Simulation aus diesen mit der Polarmethode normalverteilte Zufallszahlen berechnet [12]. λ ist ein Maß für die Anfälligkeit des Systems für Dekohärenz bei einer einzigen Operation. Der Gesamtfehler P, der angibt mit welcher Wahrscheinlichkeit aufgrund von Dekohärenz ein falsches Ergebnis gemessen wird, liegt für ein l-Qubit-System nach u Rechenschritten bei $P = 1 - e^{-lu\lambda}$ [10].

Mit diesem Wissen kann nun untersucht werden, ob ein Algorithmus bei entsprechender Dekohärenzrate überhaupt noch in polynomialer Zeit abläuft. Auch ist es möglich, Verfahren zur Fehlerkorrektur in den Algorithmus einzubauen und mit dem Simulator zu untersuchen. Als Beispiel sei in *Abb. 3* das Ergebnis des Shor-Algorithmus ohne und mit Berücksichtigung von Dekohärenz gezeigt. Die beiden Graphen zeigen die Wahrscheinlichkeitsverteilung für die einzelnen klassischen Zustände, die als Ergebnis gemessen werden können. Ohne Dekohärenz ist die Wahrscheinlichkeit, bei einer Messung ein Vielfaches der Frequenz zu erhalten, relativ hoch ($p = 0.63$). Im Vergleich dazu lässt sich mit der gewählten Dekohärenzrate λ, welche ungefähr einem auf Elektronen-Spins basierenden Quantencomputer enstpricht [10], ein solches Ergebnis mit der Wahrscheinlichkeit $p' = 0.31$ messen. Aus diesen beiden Graphen ergibt sich für den Gesamtfehler $P = \frac{p}{p'} = 0.49$. Der Shor-Algorithmus benötigt für $q = 128$ und $n = 21$ im Simulator $u = 6235$ Rechenschritte, wobei das Ergebnis aus $l = \log_2 q = 7$ Qubits gemessen wird. Damit lässt sich der theoretische Wert für P berechnen, der mit $\lambda = 1.53 \cdot 10^{-5}$ bei $P = 1 - e^{-lu\lambda} = 0.51$ liegt.

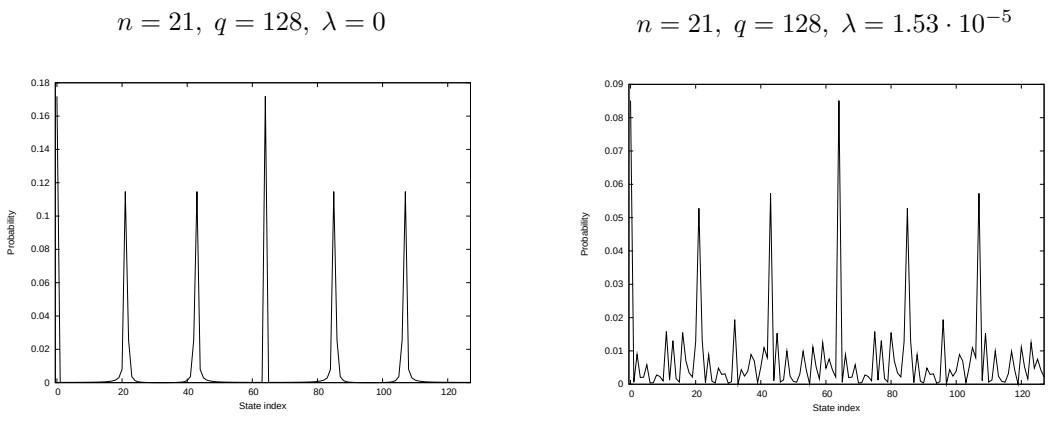

Abbildung 3: Auswirkungen von Dekohärenz auf den Faktorisierungsalgorithmus von Shor

2.5.2 Quanten-Fehlerkorrektur-Verfahren

Um trotz der durch Dekohärenz auftretenden Fehler noch effizient Rechenoperationen auf einem Quantencomputer auszuführen zu können, muss man Verfahren zur Fehlerkorrektur einsetzen. Jedoch unterscheiden sich diese Verfahren von klassischen auf Redundanz basierenden Verfahren, da ein Qubit nicht kopiert werden kann (no cloning theorem) [16]. Zudem verursacht Dekohärenz keine gewöhnlichen Bitfehler, sondern Abweichungen in der Phase der einzelnen Basiszustände.

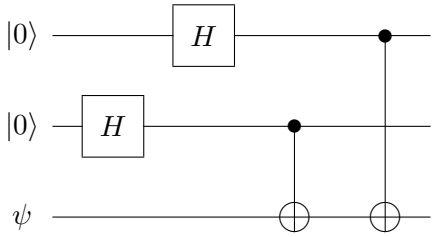

Abbildung 4: Schaltung für die Kodierung eines Qubits zur Fehlerkorrektur

Deshalb muss man spezielle Quanten-Fehlerkorrektur-Verfahren einsetzen, ein solches ist in *Abb. 4* gezeigt. Zur Korrektur von Fehlern durch Dekohärenz transformiert man zunächst die Basis des Systems mit Hadamard-Operationen. Durch diese Basistransformation werden die von der Dekohärenz verursachten Phasenfehler in Bitfehler umgewandelt [15]. Danach verschränkt man die Qubits mit zwei Controlled-Not-Operationen. Nachdem die gewünschte Berechnung durchgeführt wurde, dekodiert man das System, indem die Schaltung zur Kodierung rückwärts durchlaufen werden. Die Fehlerkorrektur wird dann durch die Messung der beiden zusätzlichen Qubits durchgeführt. Da diese beiden Qubits im Idealfall am Ende wieder auf $|0\rangle$ gesetzt wären, enthalten diese Qubits ausschließlich die Information über die angesammelten Fehler. Durch einer Messung dieser Qubits werden diese Fehler aus dem Qubit ψ entfernt. Somit wurde das Quantensystem gegen Fehler durch Dekohärenz geschützt [17].

Um ganze Schaltungen gegen Fehler zu schützen, muss man die gewünschten Gatter sowohl auf die ursprünglichen als auch die zugehörigen Qubits zur Fehlerkorrektur anwenden [18]. Der dadurch entstehende Laufzeitverlust ist proportional zur Anzahl der zu kodierenden Qubits, wodurch ein polynomialer Algorithmus weiterhin effizient abläuft. Erste Tests mit dem von uns entwickelten Simulator zeigen, dass das hier vorgestellte Verfahren zur Fehlerkorrektur geeignet ist, um ein Quantensystem zu stabilisieren. Eine Umsetzung des Verfahrens für komplette Quantenalgorithmen ist in Arbeit.

2.6 Simulation von Spin-$\frac{1}{2}$-Systemen

Ein weiteres interessantes Anwendungsgebiet für Quantencomputer ist die Simulation von quantenmechanischen Systemen. Bereits 1982 merkte der amerikanische Physiker Richard Feynman an, dass eine Computersimulation eines quantenmechanischen Systems effizient nur auf einem Computer möglich ist, der selbst ein Quantensystem zur Berechnung nutzt [19]. Diese Idee war der erste Grundstein zur Entwicklung des Quantencomputers.

Als Beispiel für ein solches quantenmechanisches System soll die Wechselwirkung zwischen zwei Teilchen mit einem magnetischen Impuls (Spin) dienen. Dies kann beispielsweise ein Elektron und ein Proton im Wasserstoff-Atom sein. Die magnetische Wechselwirkung zwischen zwei Teilchen mit dem Spin $\frac{1}{2}$ lässt sich mit den Pauli'schen Spin-Matrizen σ_x, σ_y und σ_z darstellen.

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Um eine Simulation des Systems zu ermöglichen, muss der Hamilton-Operator H berechnet werden. Da dieser nicht von der Wahl der Koordinatenachsen abhängen darf, benötigt man eine invariante Kombination der Spin-Matrizen. Diese lässt sich über das Tensorprodukt erreichen [20]:

$$H = A(\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y + \sigma_z \otimes \sigma_z) = \begin{pmatrix} A & 0 & 0 & 0 \\ 0 & -A & 2A & 0 \\ 0 & 2A & -A & 0 \\ 0 & 0 & 0 & A \end{pmatrix}$$

A ist der Energiebetrag, der die Eigenzustände von H unterscheidet. Da H als symmetrische Matrix diagonalisierbar ist, kann man die Operationsmatrix U berechnen:

$$U = e^{-\frac{iHt}{\hbar}} = \begin{pmatrix} e^{-i\frac{At}{\hbar}} & 0 & 0 & 0 \\ 0 & \frac{1}{2}e^{-i\frac{At}{\hbar}} + \frac{1}{2}e^{i3\frac{At}{\hbar}} & \frac{1}{2}e^{-i\frac{At}{\hbar}} - \frac{1}{2}e^{i3\frac{At}{\hbar}} & 0 \\ 0 & \frac{1}{2}e^{-i\frac{At}{\hbar}} - \frac{1}{2}e^{i3\frac{At}{\hbar}} & \frac{1}{2}e^{-i\frac{At}{\hbar}} + \frac{1}{2}e^{i3\frac{At}{\hbar}} & 0 \\ 0 & 0 & 0 & e^{-i\frac{At}{\hbar}} \end{pmatrix}$$

Die Schwierigkeit besteht nun darin, eine Schaltung aus elementaren Gattern zu finden, die diese Operationsmatrix umsetzt. Die Operationsmatrix U kann man auch so interpretieren, dass einerseits eine globale Phasenänderung um $e^{-i\frac{At}{\hbar}}$ auf alle Basiszustände wirkt. Andererseits ist im Mittelteil von U eine relative Phasenänderung um $e^{i3\frac{At}{\hbar}}$ enthalten, welche sich aber nur auf die Basiszustände $|01\rangle$ und $|10\rangle$ auswirkt. Für die relative Phasenänderung ist es nötig, zuerst die verschränkten Zustände aufzuheben, dieses kann man mit einem Controlled-NOT und einem Hadamard erreichen. Dann wird die relative Phasenänderung durch Phasenverschiebungs-Gatter in Kombination mit Controlled-NOTs durchgeführt. Danach werden die verschränkten Zustände wieder hergestellt und die globale Phasenänderung auf die gleiche Weise durchgeführt. Um zu zeigen, dass die Schaltung in *Abb. 5* tatsächlich die Operationsmatrix umsetzt, berechnet man das Matrixprodukt aus allen Operationen der Schaltung, mit $\theta = \frac{At}{\hbar}$. Als Ergebnis erhält man schließlich U. Dadurch ist gezeigt, dass sich auf einem Quantencomputer tatsächlich Quantensysteme simulieren lassen.

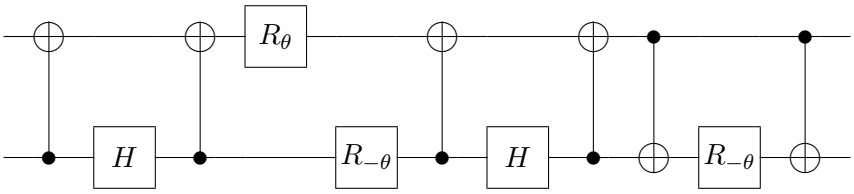

Abbildung 5: Schaltung zur Simulation von zwei Spin-$\frac{1}{2}$-Teilchen

3 Zusammenfassung und Ausblick

Der hier vorgestellte Simulator wurde als C-Bibliothek realisiert und ist unter dem Namen `libquantum` unter `http://www.enyo.de/libquantum` verfügbar. Er deckt als weltweit erster die anfangs aufgestellten Anforderungen vollständig ab. Durch eine universelle Schnittstelle ist es möglich, jeden Quantenalgorithmus in der Simulation zu testen. Mit der Berücksichtigung der Dekohärenz des Quantencomputers kann man Verfahren zur Fehlerkorrektur untersuchen und entwickeln, was für den praktischen Einsatz von Quantencomputern der entscheidende Punkt ist. Bisherige Simulatoren wie OpenQubit[2] oder QCL[3] bieten keine Möglichkeit, die Auswirkungen von Dekohärenz zu simulieren. Die einzigen Simulatoren, die dies bisher unterstützen, benötigen auf einem Mehrprozessor-Großrechner bereits mehrere Stunden, um die Zahl 15 mit einem Quantenalgorithmus zu faktorisieren[13]. Diese Aufgabe wird von dem von uns entwickelten Simulator auf einem handelsüblichen PC in weniger als einer Sekunde bewältigt.

Durch die Implementierung als C-Bibliothek ist eine hohe Portabilität und Verfügbarkeit auf verschiedensten Systemen gewährleistet. Der Simulator wird als Freie Software unter der GNU General Public License (GPL) veröffentlicht, womit eine Nachprüfbarkeit der hier vorgestellten Ergebnisse sichergestellt werden kann.

Literatur

[1] Michael Mehring: *Kernspins knacken Code*, Physik Journal, 2/1, S.17–18, 2002

[2] David Deutsch: *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proceedings of the Royal Scociety of London A, Nr. 400, S.97–117, 1985

[3] Adriano Barenco et al. *Elementary gates for quantum computation*, Physical Review Letters A, Nr. 52, S.3457–3467, 1995

[4] Lieven M. K. Vandersypen et al.: *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, Nature, Nr. 447, S.883–887, 2001

[2] `http://www.ennui.net/~quantum/`
[3] `http://tph.tuwien.ac.at/~oemer/qcl.html`

[5] Julia Wallace: *Quantum Computer Simulators*, International Journal of Computing Anticipatory Systems, Nr. 10, S.230–245, 2000

[6] Donald E. Knuth: *The Art of Computer Programming, Band 3: Sorting and Searching*, Addison-Wesley, 1973

[7] Peter W. Shor: *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, `quant-ph/9508027`[4], 1997

[8] Gary L. Miller: *Riemann's hypothesis and tests for primality*, Journal of Computer and System Sciences, Nr. 13, S.300–317, 1976

[9] David Beckman, Amalavoyal N. Chariy, Srikrishna Devabhaktuniz, John Preskill: *Effcient networks for quantum factoring*, `quant-ph/9602016`, 1996

[10] Jozef Gruska: *Quantum Computing*, McGraw-Hill, 1999

[11] Don Coppersmith: *An Approximate Fourier Transform Useful in Quantum Factoring*, IBM Research Report RC 19642, 1994

[12] Donald E. Knuth: *The Art of Computer Programming, Band 2: Seminumerical Algorithms*, Addison-Wesley, 1981

[13] Jumpei Niwa, Keiji Matsumoto, Hiroshi Imai: *General-Purpose Parallel Simulator for Quantum Computing*, `quant-ph/0201042`, 2002

[14] Isaac L. Chuang et al.: *Quantum Computers, Factoring and Decoherence*, `quant-ph/9503007`, 1995

[15] Michael A. Nielsen, Isaac L. Chuang: *Quantum Computation and Quantum Information*, Cambridge University Press, 2002

[16] Daniel Gottesman: *An Introduction to Quantum Error Correction*, `quant-ph/0004072`, 2000

[17] Andrew Steane: *Multiple Particle Interference and Quantum Error Correction*, `quant-ph/9601029`, 1995

[18] Wojciech H. Zurek, Raymond Laflamme, *Quantum Logical Operations on Encoded Qubits*, `quant-ph/9605013`, 1996

[19] Richard P. Feynman: *Simulating physics with computers*, International Journal of Theoretical Physics, Nr. 21, S.467–488, 1982

[20] Richard P. Feynman, Robert B. Leighton, Matthew Sands: *Vorlesungen über Physik, Band 3: Quantenmechanik*, Oldenbourg, 1999

[4] `http://www.arxiv.org/archive/quant-ph`