Simulation eines Quantencomputers auf einem Höchstleistungsrechner und Analyse der Fehlertoleranz

Diplomarbeit von Nikolas Pomplun
Matrikel-Nr.: 192369

Betreuer
Prof. Dr. K.-E. Hellwig
Technische Universität Berlin
Institut für Theoretische Physik

Prof. Dr. Dr. Th. Lippert
Forschungszentrum Jülich GmbH
Zentralinstitut für Angewandte Mathematik

08.11.2005

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Der Leistungssteigerung klassischer Computer sind durch die Bauweise bedingte Grenzen gesetzt. Spätestens dann, wenn bei weiterer Miniaturisierung die Ausmaße der Chipstrukturen atomare Längeneinheiten erreichen und die physikalischen Vorgänge nur noch mit quantenmechanischen Gesetzen beschrieben werden können [1]. Ein Quantencomputer dagegen ist dieser Einschränkung nicht unterworfen und kann die Prinzipien der Quantenmechanik inhärent ausnutzen.

Deutsch zeigte 1985, daß eine auf reversibler Logik aufbauende Quanten-Turing-Maschine existiert, die nach den Gesetzen der Quantenmechanik funktioniert und jedes realisierbare System in endlicher Zeit perfekt simulieren kann [2]. Damit war klar, daß ein solcher Quantencomputer mehr Fähigkeiten besitzen muß als ein klassischer Computer, denn eine klassische Turing-Maschine ist nicht in der Lage, Quantensysteme effizient zu simulieren, ein Quantencomputer jedoch schon [3, 4].

Ein gesteigertes Interesse erfuhr die Quanteninformationstheorie Mitte der 90er Jahre, als unter anderem Shor und Grover zeigen konnten, daß die von ihnen gefundenen Quantenalgorithmen in der Lage sind, Probleme schneller zu lösen, als dies mit einem klassischen Algorithmus möglich wäre [5, 6]. Beispielsweise erzielt der Shor-Algorithmus zum Zerlegen von Zahlen in ihre Primfaktoren auf einem Quantencomputer eine exponentielle Geschwindigkeitssteigerung im Vergleich zu einem Zerlegungs-Algorithmus, der auf einem klassischen Computer abläuft.

Der Geschwindigkeitszuwachs entsteht dadurch, daß der Quantencomputer sich nicht wie im klassischen Fall zwingend in einem durch eine Bitkombination festgelegten Zustand befinden muß, sondern in eine Superposition von Basiszuständen der Quantenbits, den elementaren Informationseinheiten eines Quantencomputers, gebracht werden kann. Bei einem einzigen Rechenschritt können so alle 2^n Basiszustände eines n Qubit-Systems gleichzeitig manipuliert werden. Diese Eigenschaft wird Quantenparallelismus genannt (s. Abbildung 2.1).

Heutzutage existieren verschiedene experimentelle Realisierungen von Quantencomputern. Der Ionenfallen-Computer beispielsweise nutzt verschiedene Energiezustände eines Ionentyps zur Repräsentation von Quantenbits [8, 9], während der Nuclear-Magnetic-Resonance-(NMR) Computer [10, 11] hierfür die Kernspin-Niveaus von speziellen Molekülen verwendet. Zum Auslesen sind diese Spins von Einzelmolekülen allerdings zu schwach, so daß für die Quantenbit-Darstellung auf die Kernspin-Zustände eines Ensembles von bis zu 10^{20} Molekülen zurückgegriffen wird, deren Spins sich überlagern. Mit dieser Technik ist es gelungen, einen aus 7 Quantenbits bestehenden Quantencomputer zu realisieren, mit dem die Faktorisierung der Zahl 15 möglich war [12].

Um jedoch tatsächlich einen merklichen Leistungsunterschied gegenüber einem klassischen Computer zu erreichen, sind Quantencomputer mit mindestens einigen zehn Quantenbits erforderlich. Dabei stößt man an die Grenzen des derzeitig technisch Machbaren, denn ein Erhöhen der Anzahl der Quantenbits z.B. bei einem NMR-Computer hat zur Folge, daß sich die Energieniveaus der Zustände immer weiter zusammenschieben, wodurch eine unabhängige Adressierbarkeit der einzelnen Quantenbits immer schwieriger wird. Auch bei der Rechengeschwindigkeit ist man mit ca. 10^3 Operationen/s beim NMR-Computer [20] noch weit von der Taktfrequenz eines heutigen klassischen Computers entfernt, die bei ca. $4 \cdot 10^9$ Operationen/s liegt. Ionenfallen-Computer haben ähnliche Probleme, die gegenwärtig keine beliebige Vergrößerung des Systems zulassen.

Ein wichtiges Werkzeug zum Verständnis und zur Weiterentwicklung von Quantencomputern sind Simulationen von Quantensystemen auf klassischen Computern. Obwohl es dabei nur möglich ist, die Quanteneffekte eines Quantencomputers nachzubilden, nicht aber inhärent auszunutzen, bieten derartige Simulationen wichtige Anhaltspunkte, um das Verhalten von Quantencomputern vorhersagen zu können. Jedoch können auch hier keine beliebig großen Systeme simuliert werden, da der benötigte Speicher für den Zustandsvektor des Systems exponentiell mit der Anzahl der Quantenbits wächst. Schon der Zustandsvektor eines Systems mit 36 Quantenbits benötigt bei ausreichender numerischer Präzision einen Speicherplatz von einem Terabyte.

Ressourcen dieses Umfangs besitzen heutzutage nur parallele Höchstleistungsrechner, wie das in dieser Arbeit verwendete IBM Regatta p690+ System, das aus 41 sogenannten SMP-Knoten (Symmetric Multi Prozessor) besteht, die jeweils 32 Prozessoren und 128 Gigabyte Arbeitsspeicher vereinigen und so einen Gesamtspeicher von ca. 5,2 Terabyte und eine reale Rechenleistung von 6 Teraflops (6 Billionen Gleitkommazahlberechnungen pro Sekunde) erreichen.

Ein erster Schritt zur Simulation eines Quantencomputers ist dabei die Simulation eines idealen Quantencomputers, der - unabhängig von einer physikalischen Realisierung - keinen Einschränkungen eines realen Systems, wie Dekohärenz oder fehlerhaften Gattern, unterworfen ist. Dieser ideale Computer eignet sich als grundlegendes Modell, das später auf realistischere Systeme erweitert werden kann.

Ziel dieser Arbeit war es, einen idealen Quantencomputer auf einem Multiprozessorrechner zu simulieren, mit dem sich aus einem universellen Gattersatz auf modulare Art beliebige Quantenoperationen zusammensetzen lassen. Dabei wurde besonderes Augenmerk auf die Optimierung des Programmes hinsichtlich der Rechenzeit sowie auf die Minimierung der Kommunikation zwischen den Prozessoren gelegt, um ein möglichst gutes Skalierungsverhalten der Rechenzeiten bei wachsenden Systemgrößen zu gewährleisten. Die Speichergröße des IBM Regatta p690+ Systems ermöglichte es, einen Quantencomputer mit bis zu 37 Quantenbits zu simulieren.

Darüber hinaus wurden parametrisierte Gatterfehler und Dekohärenzeffekte simuliert und ihre Auswirkungen auf den Endzustand des Systems bei unterschiedlichen Systemgrößen und Fehlerwahrscheinlichkeiten an Hand der Hadamard-Operation untersucht, um Abschätzungen über eine Mindestfehlertoleranz machen zu können.

2 Der Quantencomputer

2.1 Warum Quantencomputer ?

Nach dem Moor'schen Gesetz [1], das in seiner heutigen Form ungefähr alle 24 Monate eine Verdopplung der Bauelemente auf einer Chipplatine bei gleichzeitiger Verkleinerung der Strukturgröße voraussagt, würden in ungefähr 10 Jahren Strukturgrößen von ca. 50 nm erreicht und so die Kohärenzlänge eines Elektrons unterschritten werden. Mit klassischen Gesetzen können dann keine Aussagen mehr über die physikalischen Vorgänge gemacht werden. Leistungssteigerungen einer solchen, auf Silizium-Strukturen basierenden Rechenmaschine durch weitere Miniaturisierung wären nicht möglich.

Ein alternatives Konzept einer Rechenmaschine, das auf einer reversiblen Logik beruht, geht auf Bennett [13] und Toffoli [14] zurück, die zeigten, daß sich alle irreversiblen Operationen der Bool'schen Logik in eine reversible Form einbetten lassen und aus diesen reversiblen Operationen ein universeller Gattersatz gebildet werden kann.

Die Verallgemeinerung der Church-Turing-Hypothese [15, 16] auf Quantensysteme durch Deutsch [2] in der Form, daß ein universeller Computer, eine Quanten-Turing-Maschine, in der Lage ist, jedes realisierbare System effizient zu simulieren, führt zu der Aussage, daß ein solcher universeller Computer auf der Quantenmechanik basieren muß, da nur ein Quantencomputer ein Quantensystem effizient simulieren kann [3, 4].

Zu den neu gewonnenen Eigenschaften, die den entscheidenden Unterschied zum klassischen Computer ausmachen, zählt vor allem die Superposition von Zuständen, die durch die Linearität der Schrödinger-Gleichung möglich wird und so z.B. parallele Berechnungen von Funktionswerten einer Funktion erlaubt, woraus eine erhebliche Geschwindigkeitssteigerung resultieren kann. Dieser Leistungszuwachs beim Quantencomputer kommt dabei nicht automatisch zustande, sondern erfordert einen auf das Problem zugeschnittenen Algorithmus. Dieser ist unter Umständen in der Lage, bestimmte Probleme schneller zu lösen, als ein klassischer Computer. Auch können quantenmechanische Zustände starke Korellationen ausbilden und sich verschränken, was bei der Fehlerkorrektur und Informationsübertragung genutzt werden kann. So kann zum Beispiel Kommunikation durch die Quantenkryptographie abhörsicher gemacht werden. Allerdings bringt das Konzept des Quantencomputings auch Schwierigkeiten mit sich, die entweder durch das Wesen der Quantenmechanik bedingt entstehen, z.B. beim Auslesen oder Messen des Zustandes, das einen Kollaps der Wellenfunktion und somit den Verlust von Informationen nach sich zieht, oder durch unzureichende technische Möglichkeiten, wie z.B. bei der Abschirmung oder der Beeinflussung des Systems.

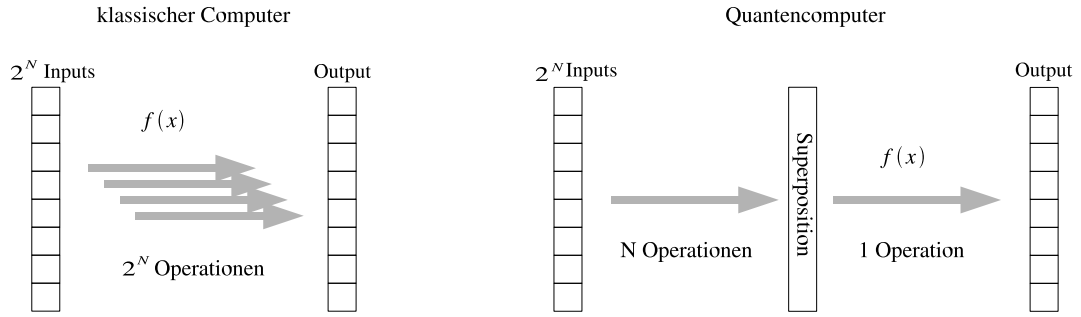

Abbildung 2.1: Quantenparallelismus (schematische Darstellung)

2.2 Die Logik des Quantencomputers

Ein Quantencomputer benutzt, im Gegensatz zur klassischen Bool'schen Logik, eine reversible Logik. Die Reversibilität hat zur Folge, daß der Computer während der Rechnung zusätzliche, für das Ergebnis unwichtige Informationen mitführt, die es erlauben, die Operationen wieder rückgängig zu machen. Bei einem klassischen Prozessor gehen diese überflüssigen Informationen verloren. Dabei steigt die Entropie und es entsteht Wärme, die zur Erhitzung des Prozessors führt. Eine vollständig reversible Entwicklung setzt natürlich ein perfekt isoliertes, abgeschlossenes System voraus, welches nur angenähert realisiert werden kann.

Die Gatter eines Quantencomputers werden durch reversible Operationen dargestellt, was sich bei der mathematischen Beschreibung in der Unitarität der Operatoren ausdrückt, da diese die Norm eines Vektors unverändert lassen und somit die Voraussetzungen für eine Wahrscheinlichkeitsinterpretation nicht verletzen.

So wie es in der Bool'schen irreversiblen Logik einen universellen Gattersatz gibt, auf den man alle Operationen zurückführen kann, gibt es auch für die reversible Logik des Quantencomputers ein Mindest-Set von Operationen, aus denen sich beliebige Operationen konstruieren lassen. Jede irreversible Operation kann auf Kosten zusätzlichen Speichers reversibel gemacht werden.

2.3 Der Geschwindigkeitszuwachs

Der Geschwindigkeitszuwachs gegenüber einem klassischen Computer hängt von einem geeigneten Algorithmus ab. Vergleiche beruhen hier auf Aussagen, wie schnell die Anzahl der zum Lösen eines Problems benötigten Rechenschritte einer Turingmaschine[1] mit der Größe des Problems anwachsen. Durch Grenzwertbetrachtungen für große Probleme ergeben sich asymptotische Funktionen, die abhängig von der Art des Algorithmus

[1] Eine universelle Turingmaschine ist nach der Church-Turing-Hypothese in der Lage, jedes berechenbares Problem zu lösen. Anders ausgedrückt: Sie kann jede andere Rechenmaschine simulieren und das Problem in maximal polynomial langsamerer Zeit berechnen. Daher bietet sich die Turingmaschine als Referenz für Algorithmen auf verschiedenen Systemen an.

verschiedenen Komplexitätsklassen zugeteilt werden können. In die Komplexitätsklasse **P** fallen Probleme, deren Rechenaufwand sich polynomial mit der Größe des Systems erhöht. Daneben gibt es noch die Komplexitätsklasse der **NP**-Probleme. Hier findet man alle Probleme, für die sich eine Lösung in polynomialer Zeit nachprüfen läßt.

Der Shor-Algorithmus zur Primfaktorenzerlegung großer Zahlen [5] ist exponentiell schneller als ein klassischer Algorithmus, da das Problem klassisch zu der Klasse der NP-Probleme gehört und der Rechenaufwand exponentiell mit der Größe der zu faktorisierenden Zahl ansteigt, der Rechenaufwand des Quantenalgorithmus jedoch nur polynomial. Weiterhin gibt es Algorithmen, wie den Grover-Algorithmus zur Datenbanksuche [6], die quadratisch beschleunigt werden können, und einige, die überhaupt nicht beschleunigt werden können, wie z.B. die Berechnung der Iteration einer Funktion [17].

Dies steht in enger Verbindung zur Superpositionseigenschaft der Quantenmechanik, die kein klassisches Analogon hat. Man kann sich diese, auch Quantenparallelismus genannte Eigenschaft bei der Entwicklung von Algorithmen zu Nutze machen, indem man z.B. das Eingabe-Register in einen Überlagerungszustand aller zu berechnenden Werte bringt und anschließend $f(x)$ nur einmal anwenden muß, wohingegen der klassische Computer den Funktionswert für jedes Argument x einzeln berechnen muß (s. Abbildung 2.1).

2.4 Verschiedene Arten von Quantencomputern

Bevor man zu den physikalischen Realisierungen eines Quantencomputers kommt, ist es sinnvoll, zum Verständnis des Grundprinzips zuerst einen idealen Quantencomputer zu betrachten. Dieser stellt ein grundlegendes Modell eines Quantenrechners dar, das nicht von einer bestimmten physikalischen Umsetzung abhängt und keinen technischen Unvollkommenheiten unterliegt. Das System ist vollständig abgeschlossen, es gehen keine Informationen verloren, und die Zeitentwicklung bleibt vollständig reversibel. Auch beeinflussen sich die Qubits nicht untereinander. Dieses ideale Modell dient als Vorstufe zur Simulation realistischer Modelle, indem es durch Zusatzterme im Hamiltonoperator erweitert werden kann. Im Hinblick auf die gegenwärtig zur Verfügung stehenden, klassischen Computer-Ressourcen ist es aber oft auch der einzige Weg, um durch Simulationen in akzeptabler Zeit Aussagen über das Verhalten eines Quantencomputers treffen zu können.

Es gibt verschiedene Ideen, einen Quantencomputer in die Praxis umzusetzen. Alle sind mit spezifischen Vor- und Nachteilen behaftet. Zu den aussichtsreichsten gehören der Ionenfallen- [8, 9] und der Nuclear-Magnetic-Resonance-Computer (NMR) [10, 11].

Im ersten Fall wird ein Qubit durch verschiedene Zustände eines lasergekühlten Ions in einer Ionenfalle realisiert. Kontrolliert wird das System durch Laserpulse. Die Vorteile liegen hier vor allem beim einfachen Auslesen der einzelnen Ionen und der Möglichkeit, das Systems gegenüber der Umgebung gut abzuschirmen. Dies spiegelt sich in einer langen Kohärenzzeit wider und erlaubt somit eine große Anzahl von durchführbaren Rechenoperationen. Die Kohärenzzeit ist das Zeitintervall, in dem die quantenmechanischen Korrelationen, die gerade den Rechenvorteil gegenüber dem klassischen Computer

ausmachen, noch erhalten sind.

Beim NMR-Computer bedient man sich der Kernspin-Einstellungen von bis zu 10^{20} Molekülen, um ein Qubit darzustellen. Ein einzelnes Molekül besteht aus Atomen verschiedener Elemente, um die Anzahl adressierbarer Energieniveaus zu erhöhen. Übergänge zwischen den Zuständen werden mit Radiofrequenzpulsen angeregt. Allerdings treten hier Probleme bei der gezielten Adressierbarkeit einzelner Qubits auf, da die Frequenzunterschiede der Resonanzfrequenzen nicht groß genug sind, um eine Beeinflussung der benachbarten Quantenbits zu verhindern.

2.5 Der Programmablauf eines Quantencomputers

Während man im Falle eines idealen Quantencomputers beim Ausführen des Programms nur von der Ausführung eines Algorithmus' auf einem definierten Anfangszustand spricht, werden beim realen Quantencomputer zusätzliche Schritte benötigt, die nicht zur eigentlichen Rechnung zählen. Diese dienen dazu, Daten zu präparieren oder auszuwerten oder um den Quantencomputer zu steuern. Diese Arbeiten können durchaus von einem klassischen Computer ausgeführt werden. Daher gliedert sich ein Programmablauf auf einem realen Quantencomputer in folgende Abschnitte (s. Abbildung 2.2):

1. *Initialisierungsphase:* Zu Beginn der Rechnung müssen die Qubits im Register des Computers initialisiert werden, d.h. sie müssen in einen bekannten Zustand, meistens den Grundzustand, gebracht werden. Diese Operation ist genau wie das Auslesen des Registers nicht reversibel. Die Initialisierung kann durch thermische Relaxation oder Laserkühlung erfolgen. Der Grundzustand kann dann durch Gatteroperationen in den für die Rechnung benötigten Anfangszustand transformiert werden. Nach der Initialisierung beginnt der eigentliche Algorithmus.

2. *Rechenphase:* Die Operationen auf den Quantenbits werden durch eine Beeinflussung des Systems von außen hervorgerufen. So etwa durch Radiofrequenzpulse bei einem NMR-Computer oder Laserlicht bei einem Ionenfallencomputer. Die entscheidenden Parameter zur Implementierung der Gatter sind die Stärke und Dauer des eingestrahlten Pulses. Während der gesamten Dauer der Rechnung gibt es auch ungewollte Wechselwirkungen, wie die der Qubits untereinander oder Dekohärenzeffekte bei der Wechselwirkung mit der Umgebung, die möglichst kompensiert werden müssen.

3. *Messung:* Zum Schluß muß der Endzustand des Systems wieder ausgelesen und in klassische Informationen umgewandelt werden. Dies ist eine nicht unitäre Operation, da die quantenmechanische Wellenfunktion zusammenbricht und bei der Projektion auf einen Eigenzustand des Systems unweigerlich Informationen verloren gehen.

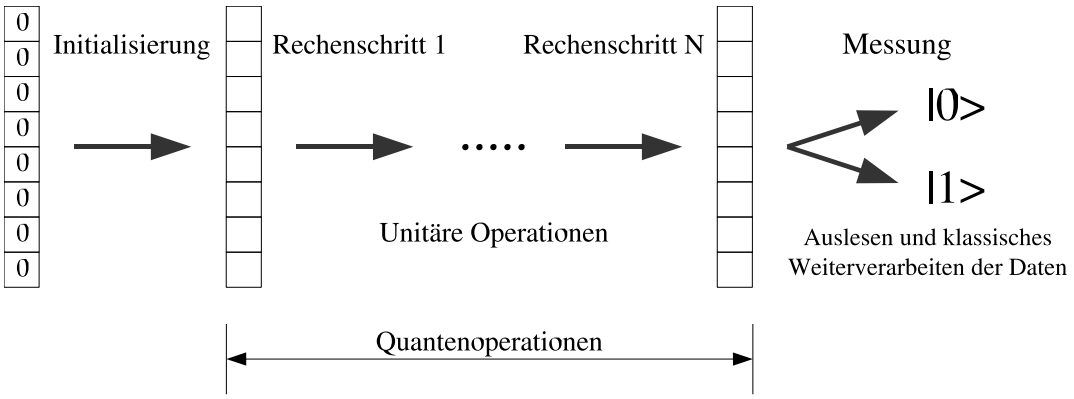

Abbildung 2.2: Ablaufschema eines Programmes auf einem Quantencomputer

2.6 Grundsätzliche Probleme

Obwohl bei der Entwicklung von Quantencomputern bereits viele Fortschritte erzielt wurden, ist die technische Umsetzung noch mit großen Schwierigkeiten verbunden [18]. Wichtige Kriterien der praktischen Umsetzbarkeit einzelner Modelle sind unter anderem die Länge der Kohärenzzeit und damit die Anzahl der durchführbaren Rechenoperationen sowie die präzise Adressierbarkeit der einzelnen Quantenbits.

Dekohärenzeffekte entstehen durch die Wechselwirkung des Systems mit der Umgebung. Sie äußern sich besonders in der Auflösung der Phasenbeziehung einzelner Zustände und führen so zu einem Informationsverlust über das System. Um dem entgegenzuwirken, ist es nötig, Fehlerkorrekturschemata einzuführen. Diese müssen auf Kosten zusätzlicher Qubits und Rechenzeit erkauft werden.

Die zum Ablauf eines Algorithmus auf dem Quantencomputer notwendigen Steuerpulse stellen eine weitere Fehlerquelle dar. Ungenauigkeiten bei der Länge und Intensität des Pulses können fehlerhafte Gatteroperationen hervorrufen. Unter Ungenauigkeiten bei der Focussierung leidet die Adressierbarkeit der Quantenbits und es entstehen ungewollte Auswirkungen auf benachbarte Qubits. Trennt man dagegen die Qubits räumlich zu weit, dann verliert man die gegenseitige Beeinflussung zwischen den Qubits, die zum Ausführen von 2-Qubit Operationen notwendig ist.

Ein fehlertoleranter Algorithmus benötigt mehrere physikalische Qubits zur Fehlerkorrektur, um ein logisches Qubit zu codieren [19]. Alle diese Probleme stellen hohe Anforderungen an die physikalische Realisierung eines Quantencomputers.

3 Quantenmechanische Grundlagen

Dieses Kapitel soll eine kurze Zusammenfassung der benötigten quantenmechanischen Grundlagen geben, die für das Verständnis eines Quantencomputers wesentlich sind. Detailliertere Ausführungen finden sich in [20] und [21] sowie in [22].

3.1 Zustände und Zustandsraum

Der Zustand eines quantenmechanischen Systems wird durch einen Vektor $|\Psi\rangle$ in einem komplexen Vektorraum mit Skalarprodukt, dem Hilbertraum $\mathcal{H}$, beschrieben, welcher als endlich-dimensional angenommen werden kann, da im weiteren nur Systeme mit endlich vielen Freiheitsgraden betrachtet werden. Der Hilbertraum wird durch die normierten Eigenvektoren gleichzeitig meßbarer Observablen aufgespannt, die eine orthogonale Basis bilden und jeden Zustandsvektor $|\Psi\rangle$ in der Form

$$|\Psi\rangle = a_1|\Psi_1\rangle + \ldots + a_n|\Psi_n\rangle \tag{3.1}$$

darstellen können. In der Dirac-Schreibweise wird ein komplexer Spaltenvektor durch einen Ket abgekürzt, der dazu konjugiert komplexe Zeilenvektor wird durch den entsprechenden Bra dargestellt:

$$|a\rangle = \begin{pmatrix} a_1 \\ a_2 \\ . \\ . \\ a_n \end{pmatrix} \tag{3.2}$$

$$\langle a| = (a_1^*, a_2^*, \ldots, a_n^*) \tag{3.3}$$

Die Komponenten des Zustandsvektors werden hier als Wahrscheinlichkeitsamplituden interpretiert, die Voraussagen über den Zustand des Systems nach einer Messung zulassen. Durch die Normierungsbedingung

$$\| \, |\Psi\rangle \, \|^2 := \langle\Psi|\Psi\rangle = \sum_i |a_i|^2 = 1 \tag{3.4}$$

wird sichergestellt, daß eine Wahrscheinlichkeitsinterpretation möglich ist.

Die Dynamik eines Quantensystems wird durch die Schrödinger-Gleichung (s. Gleichung 3.18) beschrieben. Aus ihrer Linearität folgt, daß auch Superpositionen einzelner Zustände der Form $|\Psi\rangle + |\Phi\rangle$ erlaubte Zustände sind.

3.2 Operatoren

Operatoren sind lineare Abbildungen zwischen zwei Vektorräumen V und W, für die gilt:

$$\begin{aligned} \mathbf{A} &: \mathcal{V} \rightarrow \mathcal{W} \\ \mathbf{A}(\sum_i a_i|\Psi_i\rangle) &= \sum_i a_i \mathbf{A}|\Psi_i\rangle \end{aligned} \tag{3.5}$$

Lineare Abbildungen aus einem m-dimensionalen in einen n-dimensionalen Vektorraum sind äquivalent zu einer $n \times m$-Matrix. Über die Eigenwertgleichung

$$\mathbf{A}|\Psi_i\rangle = a_i|\Psi_i\rangle \tag{3.6}$$

können die Eigenwerte des Operators als Lösungen der charakteristischen Funktion

$$c(\lambda) \equiv det|\mathbf{A} - \lambda\mathbf{I}| \tag{3.7}$$

beim Funktionswert

$$c(\lambda) = 0 \tag{3.8}$$

gefunden werden. Mit den dazugehörigen Eigenzuständen ist der Operator vollständig charakterisiert. Folgende Klassen von Operatoren sind besonders wichtig:

Observablen sind hermitesche oder selbst-adjungierte Operatoren, für die gilt:

$$\mathbf{A}^\dagger = \mathbf{A} \quad (\mathbf{A}^\dagger)_{ij} = (\mathbf{A})^*_{ji} \tag{3.9}$$

Sie entsprechen klassisch meßbaren Größen der Quantenmechanik. Ihre Eigenwerte, die Ergebnisse einer Messung, sind reell. Außerdem sind ihre Eigenvektoren paarweise orthogonal, so daß die Eigenvektoren aller gleichzeitig meßbaren Observablen eine Basis des Hilbertraumes bilden.

Unitäre Operatoren besitzen die Eigenschaft

$$\mathbf{U} \dagger \mathbf{U} = \mathbf{1} \Leftrightarrow \mathbf{U}\dagger = \mathbf{U}^{-1} \tag{3.10}$$

Sie erhalten die Norm und das Skalarprodukt von Vektoren und sind bei der zeitlichen Entwicklung eines Quantensystems von großer Bedeutung.

Projektoren bilden einen Zustand auf einen Unterraum des Systems ab, meistens aufgespannt von einem oder mehreren Eigenzuständen bestimmter Observablen. Ein Projektor auf einen Eigenzustand ist definiert als

$$\mathbf{P}_i := |a_i\rangle\langle a_i| \tag{3.11}$$

und bildet einen beliebigen Zustand $|\Psi\rangle$ auf ein Vielfaches von $|a_i\rangle$ ab. Sind alle $|a_i\rangle$ orthonormal, besitzt der Projektor folgende Eigenschaften:

$$\mathbf{P}_i\mathbf{P}_j = \delta_{ij}\mathbf{P}_j \tag{3.12}$$

$$\sum_{i=1}^{n}\mathbf{P}_i = \mathbf{1} \tag{3.13}$$

Damit lassen sich alle normalen Operatoren, die auch die Observablen mit einschließen, und für die $\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}\mathbf{A}^\dagger$ gilt, in der Spektralschreibweise

$$\mathbf{A} = \sum_{i=1}^{n}\lambda_i\mathbf{P}_i = \sum_{i=1}^{n}\lambda_i|a_i\rangle\langle a_i| \tag{3.14}$$

darstellen, die sich nur aus den Eigenwerten und -zuständen der Operatoren bzw. aus den Eigenwerten und Projektoren auf die Eigenräume zusammensetzt.

Dichtematrizen sind hermitesche, positive Operatoren, deren Spur gleich 1 ist:

$$\rho = \rho^\dagger \tag{3.15}$$

$$||\rho|\Psi\rangle|| = 0 \tag{3.16}$$

$$\mathrm{Tr}\rho = \sum_{i=1}^{k}\rho_{kk} = 1 \tag{3.17}$$

Sie eignen sich analog zum Zustandsvektor, um den Zustand eines Systems zu beschreiben, sind bei der Beschreibung von gemischten Zuständen und Mehrteilchensystemen jedoch einfacher zu handhaben und enthalten weniger redundante Informationen.

3.3 Zeitentwicklung

Die zeitliche Entwicklung eines quantenmechanischen, nach außen hin abgeschlossenen Systems wird durch eine lineare Differentialgleichung 1. Ordnung, die Schrödinger-Gleichung

$$\frac{d}{dt}|\Psi(t)\rangle = -\frac{i}{\hbar}\mathcal{H}|\Psi(t)\rangle \tag{3.18}$$

beschrieben. $\mathcal{H}$ ist der Hamilton-Operator oder Energie-Operator des Systems. Seine Eigenzustände werden stationäre Zustände genannt, da sich bei ihnen über die Zeit hinweg nur die globale Phase ändert:

$$|E\rangle \rightarrow e^{-i\frac{Et}{\hbar}}|E\rangle \tag{3.19}$$

Durch Integration gelangt man zu

$$|\Psi(t_2)\rangle = e^{-i\frac{\mathcal{H}(t_2-t_1)}{\hbar}}|\Psi(t_1)\rangle = \mathbf{U}(t_2-t_1)|\Psi(t_1)\rangle \tag{3.20}$$

als Lösung von Gleichung 3.18, wobei

$$\mathbf{U}(t_2 - t_1) = e^{-i\frac{\mathcal{H}(t_2-t_1)}{\hbar}} \tag{3.21}$$

definiert wird. $\mathbf{U}$ heißt Zeitentwicklungsoperator. Da alle Operatoren der Form $\mathbf{U} = e^{i\mathbf{K}}$ für hermitesche $\mathbf{K}$ unitär sind, gilt dies auch für $\mathbf{U}$. Die Entwicklung eines quantenmechanischen Systems wird also durch einen unitären Transformationsoperator beschrieben, der den Zustand des Systems zur Zeit t_1 mit dem Zustand des Systems zur Zeit t_2 verknüpft und nur von der Zeitdifferenz $t_2 - t_1$ abhängt. Weiterhin ergibt sich mit der Definition der Unitarität die Aussage

$$(\mathbf{U}(t))^{-1} = \mathbf{U}(-t) \tag{3.22}$$

Unitäre Zeitentwicklung ist reversibel.

In der Realität findet man kein vollständig abgeschlossenes Quantensystem, außer dem Universum selbst. Jedoch lassen sich stark abgeschirmte Systeme durch das Modell der unitären Zeitentwicklung gut beschreiben. Näherungsmethoden sind hier das einizige Mittel, da Bewegungsgleichungen für zeitabhängige Hamilton-Operatoren für mehr als zwei Dimensionen nicht mehr analytisch lösbar sind. Gerade zeitabhängige Hamilton-Operatoren müssen aber zur Beschreibung von Quantencomputern eingesetzt werden, um die Wechselwirkungen mit der Steuereinrichtung berücksichtigen zu können.

3.4 Messung

Nach der Kopenhagener Interpretation ist der Meßprozess in der Quantenmechanik ein nicht deterministischer Prozess. Alle Messungen, die an einem quantenmechanischen System gemacht werden, lassen dabei nur Aussagen über die Wahrscheinlichkeiten von Ereignissen zu. Ein System sei im Zustand $|\Psi\rangle$ präpariert. Nach einer Einzelmessung der Observablen $\mathbf{A}$ befindet sich das System nun mit der Wahrscheinlichkeit $|\ \langle a_i|\Psi\rangle\ |^2$ im Zustand

$$\frac{\mathbf{P}_{a_i}|\Psi\rangle}{\|\ \mathbf{P}_{a_i}|\Psi\rangle\ \|} \tag{3.23}$$

Hierbei bezeichnet $\mathbf{P}_{a_i}$ den Projektor auf den Eigenraum zum Eigenwert a_i. Voraussagen über den Ausgang einer Einzelmessung sind nicht möglich. Erst durch Ensemblemessungen an unabhängigen, gleichpräparierten Systemen ist eine Aussage über die Wahrscheinlichkeiten $|\ \langle a_i|\Psi\rangle\ |^2$ möglich. Bei einer Ensemblemessung erhält man den durchschnittlichen Erwartungswert

$$\langle \mathbf{A} \rangle := \langle \Psi|\mathbf{A}|\Psi\rangle \tag{3.24}$$

Messungen sind im Gegensatz zur Zeitentwicklung nicht deterministisch und nicht reversibel. Durch sie kommt es zu einer Reduktion der Wellenfunktion und zum Informationsverlust über den Zustand. Hier gibt es bisher noch keine zufriedenstellende Lösung zur Einbettung des Messprozesses in die Dynamik der Quantenmechanik, obwohl viele verschiedene Ansätze existieren [23].

3.5 Gemischte Zustände

Ein gemischter Zustand beinhaltet eine gewisse Unkenntnis über den Zustand des Systems. Dies ist der Fall bei der Beschreibung eines Ensembles von Quantensystemen, bei dem die einzelnen Zustände verschieden stark besetzt sein können. Mischzustände entstehen aber auch aus reinen Zuständen durch nicht-unitäre Operationen, d.h. bei Operationen, bei denen Information verloren geht und die Entropie ansteigt, wie zum Beispiel durch Dekohärenz oder Dissipation. Daher sind Mischzustände auch keine quantenmechanischen Superpositionen und als nicht-kohärente Überlagerungen anzusehen.

Der Unterschied zwischen einem Zustandsgemisch und einer Superposition besteht mathematisch darin, daß für einen gemischten Zustand

$$\rho = \sum_i p_i |\Psi_i\rangle\langle\Psi_i| \tag{3.25}$$

$\sum_i p_i = 1$ mit $p_i > 0$ gelten muß im Gegensatz zu $\sum_i p_i^2 = 1$ bei einer Superposition.

3.6 Dichtematrix

Die Beschreibung gemischter Zustände ist mit dem Zustandsvektoransatz zwar möglich, aber sehr aufwendig. Besser geeignet ist der Dichtematrixformalismus. Der Zustand eines Systems, das sich mit der Wahrscheinlichkeit p_i in einem der reinen Zustände $|\Psi_i\rangle$ befindet, entspricht der Dichtematrix

$$\rho \equiv \sum_i p_i |\Psi_i\rangle\langle\Psi_i| \quad \text{mit} \quad \sum_i p_i = 1 \tag{3.26}$$

Dies ist ein hermitescher, positiver Operator, dessen Spur gleich 1 ist. Umgekehrt ist die Zerlegung einer Dichtematrix ρ in reine Zustände nicht eindeutig.

Eine allgemeine Dichtematrix eines 1-Qubit-Zustandes läßt sich aus einer Linearkombination der Spin-Matrizen σ_i und des Einheitsoperators $\mathbf{1}$ in der Form

$$\frac{1}{2}\left(\mathbf{1} + \frac{2}{\hbar}\vec{P}\cdot\vec{\sigma}\right) \tag{3.27}$$

zusammensetzen. $\vec{P}$ ist dabei ein reellwertiger 3-dimensionaler Vektor. Da die σ_i hermitesch und spurlos sind, gilt für beliebige Linearkombinationen aus σ und $\frac{1}{2}\mathbf{1}$, daß diese hermitesch sind und ihre Spur gleich 1 ist. Die Eigenwerte von (3.27) sind

$$\lambda_\pm = \frac{1}{2}(\mathbf{1} \pm |\vec{P}|) \tag{3.28}$$

Damit ist die Matrix positiv, falls $|\vec{P}| \leq 1$ gilt.

Die Diagonalität der Matrix hängt von der Wahl der Basis ab und hat so nur für ausgewählte, willkürlich ausgezeichnete Situationen eine physikalische Aussage. Dagegen gilt unabhängig von der Wahl der Basis, daß ein reiner Zustand vorliegt, wenn gilt:

$$\rho^2 = \rho \tag{3.29}$$

und ein gemischter Zustand, falls dies nicht erfüllt ist.

Für die Zeitentwicklung und Messung des Systems im Dichtematrixformalismus gelten dieselben Gesetze wie für den Zustandsvektoransatz. Die Zeitentwicklung wird durch die unitäre Abbildung

$$\mathbf{U} : \rho = \sum_i p_i |\Psi_i\rangle\langle\Psi_i| \rightarrow \sum_i p_i U|\Psi_i\rangle\langle\Psi_i|U^\dagger = U\rho U^\dagger \tag{3.30}$$

beschrieben. U ist dabei der in Gleichung (3.21) beschriebene Zeitentwicklungsoperator.

Bei einer Messung des Quantensystems im Zustand ρ erhält man mit der Wahrscheinlichkeit

$$p(m) = tr(M_m^\dagger M_m \rho) \tag{3.31}$$

den Eigenwert m des Operators M, und der Zustand nach der Messung ist

$$\rho = \frac{M_m \rho M_m^\dagger}{tr(M_m^\dagger M_m \rho)} \tag{3.32}$$

3.7 Zusammengesetzte Systeme

Setzt man mehrere Qubits zu einem System zusammen, dann werden aus mathematischer Sicht die Hilberträume der einzelnen Qubits durch das Tensorprodukt $\otimes$ verbunden. Ist V ein Vektorraum der Dimension n und W ein Vektorraum der Dimension m, dann hat der zusammengesetzte Raum $V \otimes W$ die Dimension nm. Ein Element von $V \otimes W$ ist wieder eine Linearkombination von Tensorprodukten der einzelnen Vektorräume.

$$|v_1\rangle \otimes |w_1\rangle + |v_2\rangle \otimes |w_2\rangle \in V \otimes W \quad |v_1\rangle, |v_2\rangle \in V \quad |w_1\rangle, |w_2\rangle \in W \tag{3.33}$$

Lineare Operatoren können in gleicher Weise zu einem Operator des Gesamtsystems zusammengesetzt werden, wobei die Teiloperatoren nur auf ihrem jeweiligen Vektorraum operieren.

$$(A \otimes B)(|v\rangle \otimes |w\rangle \equiv A|v\rangle \otimes B|w\rangle \tag{3.34}$$

Zusammengesetzte Operatoren haben in Matrixschreibweise folgende Gestalt: A sei eine $n \times m$-Matrix, B eine $p \times q$-Matrix. Die Matrix $A \otimes B$ hat dann folgende Darstellung:

$$A \otimes B \equiv \begin{pmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & A_{22}B & \dots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{m1}B & A_{m2}B & \dots & A_{mn}B \end{pmatrix} \tag{3.35}$$

An jedem Eintrag der Matrix A findet sich nun eine $p \times q$ große Untermatrix $A_{xy}B$.

4 Theorie des Quantencomputers

4.1 Ein Quantenbit

Die Grundbausteine eines Quantencomputers sind einzelne Quantenbits, aus denen wiederum die einzelnen Register des Quantencomputers zusammengesetzt sind. Ein geeignetes physikalisches System zur Realisation eines Quantenbits muß mindestens zwei unterscheidbare Zustände besitzen. Außerdem muß die Superpositionsfähigkeit der beiden Zustände gewährleistet sein. Beispielsweise ist ein System, in dem das Vorhandensein eines Teilchens z.B. den Zustand $|1\rangle$ beschreibt und das Nicht-Vorhandensein den Zustand $|0\rangle$ ungeeignet, da Superpositionen aus zwei Zuständen mit unterschiedlichen Teilchenanzahlen für massebehaftete Teilchen nicht möglich sind. Als vielversprechend erwiesen haben sich dagegen der Spin von Elektronen und Atomkernen oder aber die Polarisation von Photonen. Um die zeitliche Entwicklung des Systems beschreiben zu können, muß außerdem der Hamilton-Operator des Systems bekannt sein.

Zur mathematischen Beschreibung eines Quantenbits wird ein Spin-$\frac{1}{2}$-System gewählt. Im einfachsten Fall, dem eines einzelnen Qubits, spannen die zwei orthogonalen Basisvektoren

$$\begin{aligned} |0\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ |1\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

einen 2-dimensionalen, komplexen Hilbertraum auf, in dem das Quantenbit durch den Zustandsvektor

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \qquad \alpha, \beta \in \mathbb{C} \tag{4.1}$$

dargestellt werden kann. Basis des Hilbertraumes seien die Eigenzustände des Spin-Operators σ_z, $|0\rangle = \binom{1}{0}$ und $|1\rangle = \binom{0}{1}$. Diese Wahl geschieht willkürlich und der mathematischen Einfachheit halber. Durch die Bedingung

$$|\alpha|^2 + |\beta|^2 = 1 \quad \alpha, \beta \in \mathbb{C} \tag{4.2}$$

wird der Zustandsvektor auf Einheitslänge normiert, was für eine probabilistische Theorie unabdingbar ist, da bei einer Messung auf jeden Fall ein Ereignis eintreten muß.

Der physikalische Zustand des Systems wird durch einen globalen Phasenfaktor nicht verändert, da dieser beim Berechnen des Erwartungswertes herausfällt. $|\Psi\rangle$ und $e^{i\alpha}|\Psi\rangle \quad (\alpha \in \mathcal{R})$ sind also äquivalent. Dagegen sind die relativen Phasen zwischen den Zuständen sehr wichtig: $|\Psi\rangle + |\Phi\rangle$ und $|\Psi\rangle + e^{i\alpha}|\Phi\rangle \quad (\alpha \neq 0)$ sind verschiedene Zustände.

Gleichung 4.1 stellt eine Superposition der beiden logischen Basiszustände dar. Hier wird der Unterschied zum klassischen Bit deutlich, welches sich als ein Spezialfall aus dem Quantenbit ergibt. Während beim klassischen Bit entweder α oder β genau gleich 1 sind, werden im quantenmechanischen Fall alle α und β zugelassen, die Gleichung 4.2 erfüllen. $|\alpha|^2$ und $|\beta|^2$ sind die Wahrscheinlichkeiten, mit denen sich das Quantenbit nach einer Messung von S_z in einem der beiden Zustände $|0\rangle$ oder $|1\rangle$ wiederfindet.

4.1.1 Bloch-Sphäre

Da die globale Phase des Zustandsvektors keinen physikalischen Einfluß auf das System hat, kann der Parameter α in Gleichung 4.1 durch geeignete Transformationen immer reell gemacht werden. Damit haben wir drei reelle, unabhängige Koordinaten, die den Zustand eindeutig festlegen. Durch eine Koordinatentransformation der Art

$$\begin{aligned} \alpha &= cos(\frac{\theta}{2}) \\ \beta &= e^{i\phi} sin(\frac{\theta}{2}) \qquad\qquad 0 < \theta < \pi \qquad 0 < \phi < 2\pi \end{aligned} \tag{4.3}$$

kann man den Zustand eines Qubits in der Bloch-Sphäre in Abhängigkeit der Winkel θ und ϕ graphisch veranschaulichen (s. Abbildung 4.1). Reine Zustände liegen auf der Oberfläche der Einheitskugel, gemischte Zustände dagegen im Inneren. Eine andere Art, die Bloch-Sphäre zu definieren, ist der Vektor $\vec{P}$ aus Kapitel 3.6. Alle möglichen Vektoren $\vec{P}$ mit $|\vec{P}| \leq 1$ spannen die Bloch-Sphäre auf. 1-Qubit-Operationen können hierbei als Drehungen des Bloch-Vektors um den Koordinatenursprung aufgefaßt werden.

4.2 Mehrere Quantenbits

Betrachtet man 2 Quantenbits, so wird der nun 4-dimensionale Hilbertraum durch die orthonormalen Basiszustände $|0\rangle_A$,$|1\rangle_A$,$|0\rangle_B$ und $|1\rangle_B$ aufgespannt. Befinden sich 2 Quantensysteme A und B jeweils in den Zuständen $|\Psi\rangle_A$ und $|\Phi\rangle_B$ so befindet sich das Gesamtsystem in einem Produktzustand der Form $|\Psi\rangle_A \otimes |\Phi\rangle_B$.

Dabei ermöglicht das Tensorprodukt eine Definition der Korrelation oder der Verschränkung der beiden Qubits. Wenn der Zustand des Systems nicht als Produktzustand der einzelnen Teilsysteme ausgedrückt werden kann, ist der Zustand verschränkt. Ist im Gegensatz dazu der Zustand als Produktzustand darstellbar, wird er separabel genannt.

Eliminiert man durch partielle Spurbildung das System B, dann erhält man abhängig davon, ob der Gesamtzustand verschränkt war oder nicht, unterschiedliche Ergebnisse. Befand sich das System vor der Messung in einem nicht verschränkten Zustand, also in einem Produktzustand, dann ergibt die Untersuchung der reduzierten Dichtematrix des

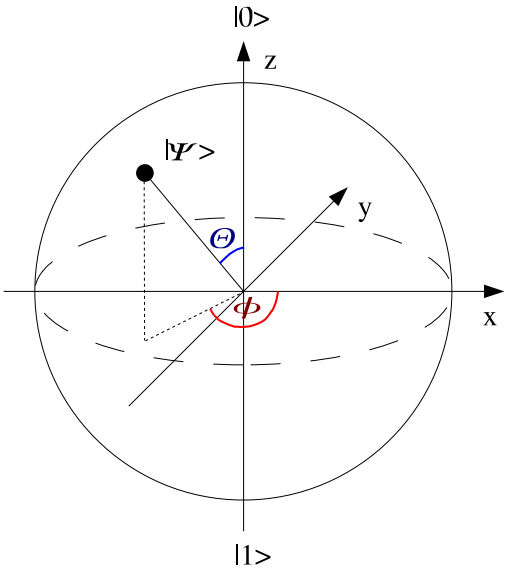

Abbildung 4.1: Bloch Sphäre: Der Zustand des Qubits kann in Abhängigkeit der Winkel θ und ϕ dargestellt werden

Systems A, daß sich dieses System in einem reinen Zustand befindet. War der Gesamtzustand verschränkt, so ist System A nach der partiellen Spurbildung in einem gemischten Zustand. Der ursprüngliche reine Zustand von System A vor der Interaktion mit System B ist verloren gegangen.

4.3 Quantengatter

Sobald die Qubits initialisiert sind, braucht man eine Möglichkeit, sie zu manipulieren. Diese Operationen werden analog zum klassischen Computer als Gatter bezeichnet, die im quantenmechanischen Fall unitären Abbildungen entsprechen. Die Unitarität ist dabei Voraussetzung für die Reversibilität des Quantencomputers.

4.3.1 1-Qubit-Gatter

Die allgemeinste Form eines 1-Qubit-Gatters entspricht einer unitären 2x2-Matrix mit den Eigenwerten $+1$ und -1. Jede dieser Matrizen kann aus den Pauli-Matrizen und

dem Einheitsoperator

$$\begin{aligned} \mathbf{X} &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \mathbf{Y} &= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \\ \mathbf{Z} &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\ \mathbf{1} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned} \tag{4.4}$$

in Form von

$$\mathbf{R} = \mathbf{1} - n_x\mathbf{X} - n_y\mathbf{Y} - n_z\mathbf{Z} \quad |\vec{n}| = 1 \tag{4.5}$$

zusammengesetzt werden. Während es im klassischen Fall nur eine nicht triviale 1-Qubit-Operation gibt, nämlich das NOT-Gatter, existieren im quantenmechanischen Fall unendlich viele Operationen auf einem Quantenbit, die Drehungen des Zustandsvektors um verschiedene Achsen und Winkel bewirken.

Rotationen werden in der Quantenmechanik durch den Drehimpulsoperator $e^{in_\alpha S_\alpha}$ erzeugt. Die darin auftretenden Spinmatrizen sind mit den Pauli-Matrizen durch die Beziehung $S_\alpha = \frac{\hbar}{2}\sigma_\alpha$ verknüpft, voraus sich für eine Drehung um einen Winkel θ um eine beliebige Achse $\vec{n}$ ergibt:

$$\mathbf{R}_{\vec{n}}(\theta) = e^{i\theta\frac{2}{\hbar}\vec{n}\vec{\mathbf{S}}} = \mathbf{1}cos(\theta) + i\frac{2}{\hbar}\vec{n}\vec{\mathbf{S}}sin(\theta) \tag{4.6}$$

Zu den grundlegenden Operationen gehört die $\frac{\pi}{2}$-Rotation um die x- und y-Achse (Gleichung 4.4). Dabei kommt dem X-Operator die Rolle eines NOT-Gatters zu, das die Amplituden der Basisvektoren vertauscht:

$$\mathbf{X}|\Psi\rangle = \mathbf{X}(\alpha|0\rangle + \beta|1\rangle) = \beta|0\rangle + \alpha|1\rangle \tag{4.7}$$

Das Z-Gatter aus Gleichung 4.4 bewirkt eine relative Phasenverschiebung zwischen den Basisvektoren um den Winkel π. Allgemein generiert

$$e^{i\theta\mathbf{Z}} = \begin{pmatrix} e^{i\theta} & 0 \\ 0 & e^{-i\theta} \end{pmatrix} \tag{4.8}$$

eine Phasenverschiebung um 2θ. Hieraus lassen sich zwei weitere Phasengatter ableiten, die als Spezialfall des Z-Gatters anzusehen sind. Das S-Gatter mit der Eigenschaft $\mathbf{S}^2 = \mathbf{Z}$:

$$\mathbf{S} = \mathbf{T}^2 = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{4.9}$$

und das $\frac{\pi}{8}$-Gatter, auch oft mit T bezeichnet, für das gilt: $\mathbf{T}^2 = \mathbf{S}$

$$\mathbf{T} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} = e^{i\frac{\pi}{8}} \begin{pmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{pmatrix} \tag{4.10}$$

Das Hadamard-Gatter

$$\mathbf{H} = \frac{1}{\sqrt{2}}(X + Z) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4.11}$$

überführt den Zustand $|0\rangle$ nach $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ und den Zustand $|1\rangle$ nach $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, also in eine gleichteilige Superposition der Zustände. Damit ist es die Quadratwurzel des NOT- bzw. X-Gatters.

4.3.2 2-Qubit-Gatter

Um ein Quantenbit in Abhängigkeit des Zustandes eines anderen Quantenbits zu beeinflussen, werden 2-Qubit-Gatter benötigt, wobei man die involvierten Quantenbits zur logischen Unterscheidung in Kontroll- und Zielqubit unterteilt. Das Controlled-NOT-Gatter (CNOT) zum Beispiel invertiert das Zielqubit, wenn das Kontrollqubit im Zustand $|1\rangle$ ist. Die folgende Wahrheitstabelle zeigt die Aktion des CNOT-Gatters. Dabei wird das vordere Quantenbit als Kontrollbit und das hintere Quantenbit als Zielbit angenommen.

$$\begin{aligned} 00 &\rightarrow 00 \\ 01 &\rightarrow 01 \\ 10 &\rightarrow 11 \\ 11 &\rightarrow 10 \end{aligned} \tag{4.12}$$

Hier kann man gut erkennen, wie die irreversible Logik in den reversiblen Fall eingebettet ist. Der Zustand des Zielbits nach der CNOT-Operation entspricht einem klassischen exklusiven ODER (XOR), nur bleibt hier auch das Kontrollbit erhalten, die nötige Information also, um die Operation rückgängig zu machen, was durch eine weitere CNOT-Operation bewerkstelligt werden kann:

$$(\mathrm{CNOT})^2 = \mathbf{1} \tag{4.13}$$

Die Matrixdarstellung des CNOT-Gatters in der $|00\rangle, |01\rangle, |10\rangle, |11\rangle$- Basis sieht folgendermaßen aus

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.14}$$

Ausgedrückt in 2×2-Matrizen, läßt sich erkennen, daß das CNOT-Gatter eine Erweiterung des $\mathbf{X}$-Gatters ist:

$$\mathrm{CNOT} = \begin{pmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{X} \end{pmatrix} \tag{4.15}$$

Ebenso kann man z.B. das Phasengatter **Z** nehmen und daraus einen kontrollierten Phasenshift auf dem Zielbit in Abhängigkeit des Zustands des Kontrollbits bewirken:

$$\begin{pmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{Z} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & e^{-i\theta} \end{pmatrix} \tag{4.16}$$

Auf diese Weise läßt sich aus jeder 1-Qubit-Operation **U** eine kontrollierte Operation **CU** machen.

4.3.3 Universeller Gattersatz und Darstellung

Wie beim klassischen Computer sind alle Operationen durch wenige Operationen, einen universellen Gattersatz, reproduzierbar. In der Bool'schen Logik ist ein universelles Gatterpaar zum Beispiel das NAND- und COPY-Gatter. In der Quantenmechanik kann jedes beliebige Gatter als Folge von 1-Qubit-Operationen und dem CNOT-Gatter realisiert werden. Weiterhin genügen sogar nur das Hadamard- zusammen mit dem S- und T-Gatter als 1-Qubit-Operationen, um jede Operation bis zu beliebiger Genauigkeit anzunähern ([20]).

Um das Ablaufschema eines Quantencomputers graphisch darzustellen, zeichnet man für jedes Quantenbit eine horizontale Linie. Die zeitliche Entwicklung verläuft von links nach rechts. Durch die Symbole auf den Linien wird die Art der Operation beschrieben. Vertikal überlappende Gatter können simultan ausgeführt werden. In Abbildung 4.2 sind die Symbole einiger wichtiger Quantengatter wiedergegeben. Beim Phasengatter entspricht der Parameter k dem Winkel $\frac{\pi}{2^k}$ der Phasenverschiebung. Die Gatter S und T entsprechen einem Phasengatter mit den k-Werten 2 bzw. 3.

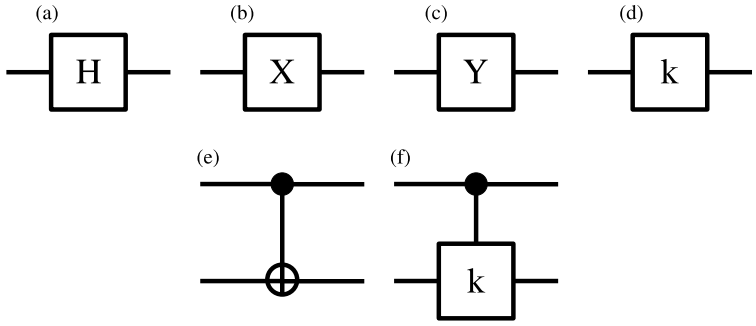

Abbildung 4.2: Darstellung der Quantengatter: (a) Hadamard,(b) und (c) Drehung um $\frac{\pi}{2}$ um x- und y-Achse, (d) Phasenshift, (e) CNOT, (f) kontrollierter Phasenshift

4.3.4 Zusammengesetzte Operationen

Aus dem Gattersatz in Abbildung 4.2 kann man nun weitere Gatter kombinieren. So kann man z.B. Kontroll- und Zielbit des CNOT-Gatters invertieren, indem man vor und nach der CNOT Operation auf beide Qubits ein Hadamard-Gatter anwendet. Durch drei CNOT-Gatter, bei denen abwechselnd Kontroll- und Zielbit vertauscht werden, lassen sich die Zustände zweier Qubits vertauschen. Dies entspricht der SWAP-Operation: Darüber hinaus gibt es Gatter, die auf mehr als zwei Quantenbits operieren, wie zum

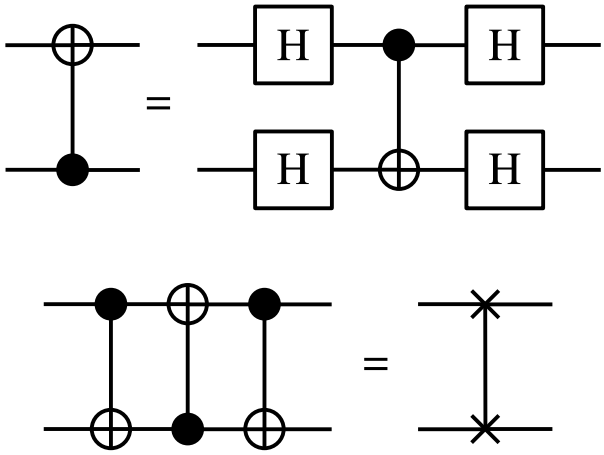

Abbildung 4.3: CNOT-Gatter mit vertauschtem Kontroll- und Zielbit / SWAP-Gatter

Beispiel das Toffoli- oder C^2NOT- Gatter, das das Zielbit nur invertiert, wenn sich beide Kontrollbits im Zustand $|1\rangle$ befinden oder das Fredkin-Gatter, das die Zustände seiner zwei Zielbits vertauscht, wenn das Kontrollbit im Zustand $|1\rangle$ ist:

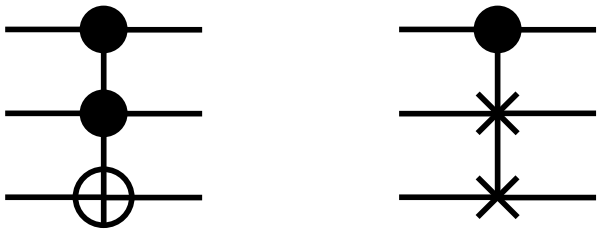

Abbildung 4.4: Toffoli- und Fredkin-Gatter

4.4 Fehlerhafte Gatter und Dekohärenz

Ein realer Quantencomputer muß in der Lage sein, die mathematischen Operationen mit großer Genauigkeit zu implementieren. Jedes physikalische Bauteil weicht jedoch bis zu einem bestimmten Maß vom idealen mathematischen Fall ab. Daher muß bei der

Betrachtung eines realen Quantencomputers das Auftreten von Fehlern berücksichtigt werden.

Alle auftretenden Fehler lassen sich dabei in zwei Kategorien unterteilen. Zum einen Fehler, die durch ungewollte Wechselwirkung mit der Umgebung entstehen und so zu Dekohärenzeffekten führen, und zum anderen Fehler, die durch fehlerhaftes Ausführen der Gatter-Operationen entstehen und so bei wiederholtem Ausführen der Operationen zu einem anwachsenden Genauigkeitsverlust führen (s. Kapitel 8).

Die Fehlerkorrektur bietet eine Möglichkeit, Fehler zu erkennen und zu beheben, indem der Zustand eines Qubits in mehreren physikalischen Qubits verschlüsselt wird. So wird die Wahrscheinlichkeit eines unentdeckten Fehlers, der im Verlauf der Rechnung zu einem falschen Ergebnis führt, im Vergleich zum unverschlüsselten Qubit stark reduziert.

4.4.1 Dekohärenz-Modell

Ist das Quantensystem nicht perfekt von seiner Umgebung abgeschirmt, fangen System und Umgebung an, Korrelationen untereinander auszubilden, die sich in einen zunehmenden Verlust der Phasenbeziehungen des Systemzustandes äußern.

Eine einfache Art zur Simulation von Kohärenzeffekten bietet das Modell des *depolarizing channel*, das mit einer Wahrscheinlichkeit von $1-p$ das Qubit unverändert läßt und mit einer Wahrscheinlichkeit von $\frac{p}{3}$ jeweils eine der Operationen $\mathbf{X}$, $\mathbf{Z}$ oder $\mathbf{Y}$ auf das Qubit anwendet. Die Wahrscheinlichkeit p, mit der eine der Pauli-Matrizen auf das Qubit angewendet wird, ist ein Maß für die Stärke der Dekohärenz durch Depolarisierung.

Ein *Bit-Flip* vertauscht die Amplituden vor den Basiszuständen $|0\rangle$ und $|1\rangle$ und entspricht daher dem X-Gatter. Ein *Phase-Flip* verschiebt die relative Phase zwischen den Zuständen $|0\rangle$ und $|1\rangle$ um den Winkel π und entspricht damit dem Z-Gatter. Das Y-Gatter entspricht einem gemeinsamen *Bit-* und *Phase-Flip*.

Dieses Dekohärenzmodell kann mathematisch als Operation auf der Dichtematrix des Systems beschrieben werden. Diese entspricht in der Quanteninformationstheorie einer Informationsübertragung durch einen sogenannten *depolarizing channel*. Dabei wird die Dichtematrix ρ der Transformation

$$\rho \rightarrow (1-p)\mathbf{1}\rho\mathbf{1}^\dagger + \frac{p}{3}\mathbf{X}\rho\mathbf{X}^\dagger + \frac{p}{3}\mathbf{Y}\rho\mathbf{Y}^\dagger + \frac{p}{3}\mathbf{Z}\rho\mathbf{Z}^\dagger \tag{4.17}$$

unterzogen. Physikalisch betrachtet entsprechen die Operationen $\mathbf{X}$, $\mathbf{Z}$ und $\mathbf{Y}$ einem *Bit-Flip*, einem *Phase-Flip* bzw. einer Kombination aus beiden.

4.4.2 Modell fehlerhafter Operationen

Fehlerhafte Gatteroperationen entstehen durch Ungenauigkeiten der Steuerung des Quantencomputers. Z.B. weisen Radio- oder Laserpulse, die beim NMR- bzw. beim Ionenfallen-Computer zur Ausführung der Rechenoperationen benutzt werden, Ungenauigkeiten in der Dauer und Stärke der eingestrahlten Pulse auf.

Um eine Beschreibung der Auswirkungen fehlerhafter Gatter geben zu können, kann man als einfaches Modell die entsprechende Matrix der Operation in verschiedene Rotationen und Phasenshifts zerlegen. So gewinnt man eine parametrisierte Darstellung der Operation aus einzelnen elementaren Gattern in Abhängigkeit der Drehwinkel. Für den speziellen Fall der Hadamard-Operation wird dies in Kapitel 8.1 durchgeführt.

5 Multi-Prozessor-Rechner

5.1 Architekturen verschiedener Multi-Prozessor-Rechner

Es gibt eine Vielzahl von verschiedenen Architekturkonzepten für Multi-Prozessor-Rechner. In Abhängigkeit von der Art, in der Daten und Befehle verarbeitet werden, kann man verschiedene Systeme klassifizieren. Bei der Befehlsverarbeitung unterscheidet man zwischen dem *Single-Instruction-Mode (SI)*, bei dem jeder Prozessor die gleichen Befehle ausführt, und dem *Multiple-Instruction-Mode (MI)*, bei dem den Prozessoren verschiedene Aufgaben zukommen. Greifen alle Prozessoren auf die gleichen Daten zu, spricht man vom *Single-Data-Mode (SD)* bzw. vom *Multiple-Data-Mode (MD)*, wenn auf unterschiedliche Daten zugegriffen wird.

Durch Kombination dieser Modi ergeben sich die *Flynn'schen Klassifikationen* für die grundlegenden Hardwarekonzepte: SISD, SIMD, MISD und MIMD. Hier soll nur auf die MIMD-Klasse eingegangen werden, da die heute gängigsten Großrechner wie z.B. der IBM Blue Gene oder der IBM Regatta p690+ (JUMP), auf dem die Simulationen dieser Arbeit durchgeführt worden sind, in diese Klasse fallen.

Zum weiteren Verständnis der Unterschiede zwischen den einzelnen Systemen und ihren jeweils spezifischen Problemen bei der Programmierung ist nun eine tiefergehende Beschäftigung mit dem internen Aufbau und vor allem mit dem Speicherkonzept dieser Rechner unabdingbar.

Die zentralen Bauelemente sind die Prozessoren, die aus der Befehle verarbeitenden CPU (Central Processing Unit) und den Datenregistern bestehen. Dabei müssen die Daten natürlich erst in die Register gelangen, um dann verarbeitet zu werden. Jedem dieser Prozessoren ist ein sogenannten Cache zugeordnet, ein im Vergleich zum Arbeitsspeicher relativ kleiner, schneller Speicher, in dem die zuletzt benötigten Speicherinhalte aus dem Arbeitsspeicher gepuffert werden. Wird nun oft hintereinander dieselbe Speicheradresse benötigt, und liegt diese noch vom vorherigen Aufruf im Cache, kann der Prozessor die Daten einfach aus dem Cache in seine Register laden und muß nicht auf den Arbeitsspeicher zugreifen. Die Idee dahinter wird klar, wenn man sich verdeutlicht, daß bei einen Prozessor des JUMPs der Zugriff auf den Arbeitsspeicher ungefähr 200 ns dauert, der Zugriff auf den Cache aber nur ungefähr 2-10 ns und der Zugriff auf ein Prozessorregister sogar nur rund 1 ns. Es wird daher angestrebt, die benötigten Daten so nah wie möglich am Prozessor zu halten, um eine optimale Geschwindigkeit zu erreichen.

An der Art, wie nun der Zugriff der Prozessoren auf den Arbeitsspeicher erfolgt, un-

terscheidet man zwischen einem Multi-Prozessor- und einem Multi-Computer-System. Beim Multi-Prozessor-System greifen alle Prozessoren über ein Verbindungsnetzwerk auf einen gemeinsamen Arbeitsspeicher zu. Dieses eng gekoppelte Netzwerk nennt man auch *Shared Memory Model (SM)* oder *Uniform Memory Adress Model (UMA)* (s. Abbildung 5.1). Hier liegt der Vorteil bei der hohen Zugriffsgeschwindigkeit auf den Speicher, gleichzeitig ergibt sich aber auch das Problem, daß es keine Zugriffsüberschneidungen auf eine Speicheradresse geben darf, d.h. brauchen alle Prozessoren das gleiche Speicherdatum, dann kommt man nicht um einen seriellen Zugriff umhin. Außerdem wird bei zu großen Prozessorzahlen der Speicherbus, also die Schnittstelle zwischen Speicher und Prozessor zum Flaschenhals, d.h. das Netzwerk kann die Prozessoren nicht schnell genug mit Daten speisen. Bis zu einem gewissen Maß kann dieser Effekt durch Einsatz von mehrstufigen Cachespeichern kompensiert werden, da diese den Speicherbus entlasten können. Einen Weg, diesen Engpaß zu umgehen, bietet das Multi-Computer-System (s. Abbildung 5.2), auch *Distributed Memory Model (DM)* oder *NonUniform Memory Adress Model (NUMA)* genannt, basierend auf einem lose gekoppelten Zusammenschluß der Prozessoren. Hier besitzt jeder Prozessor seinen eigenen Arbeitsspeicher, auf den er uneingeschränkten Zugriff hat. Die gesamte Prozessor-/Speichereinheit ist über ein Netzwerk mit den anderen Prozessor-/Speichereinheiten verbunden, was den Datenaustausch zwischen den Arbeitsspeichern nötig macht und einen erhöhten Programmieraufwand erfordert. Daneben gibt es Mischformen, bei denen die beiden Konzepte vereinigt werden und sozusagen Shared Memory Modelle in ein Distributed Memory Modell eingebettet werden [s. Abbildung 5.3).

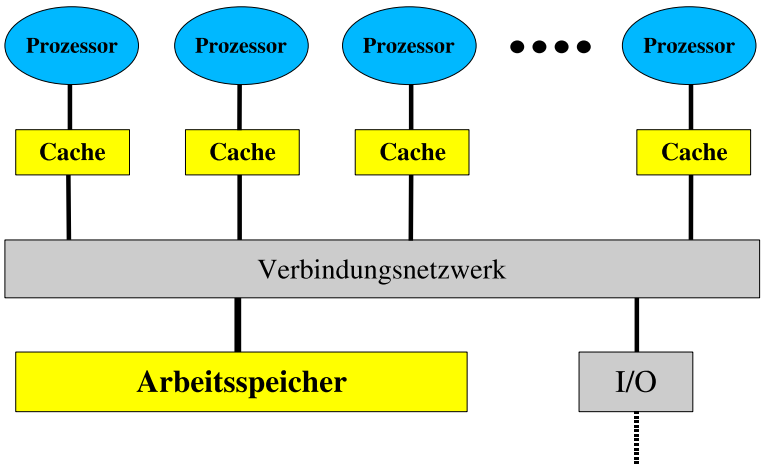

Abbildung 5.1: Das Shared Memory Model

5.2 Der IBM Regatta p690+ (JUMP)

Der **Jü**lich **M**ulti**p**rozessor-Rechner oder kurz *JUMP* wurde in seiner jetzigen Konfiguration Anfang des Jahres 2004 in Betrieb genommen. Zu diesem Zeitpunkt einer der

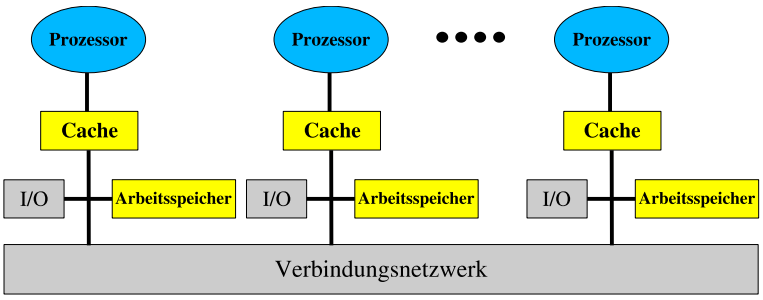

Abbildung 5.2: Das Distributed Memory Model

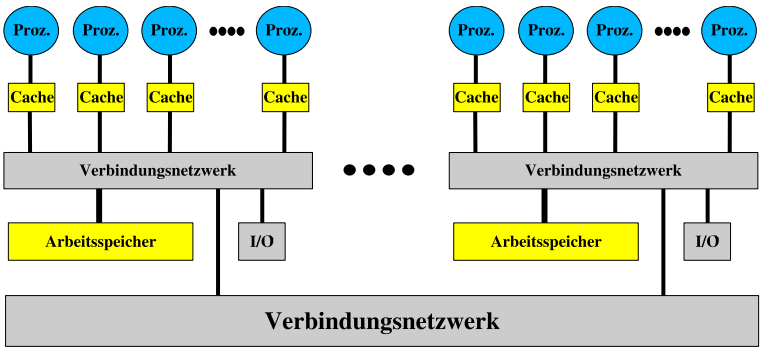

Abbildung 5.3: Mischform aus Distributed und Shared Memory Model

schnellsten Großrechner weltweit mit einer Spitzengeschwindigkeit von 8.9 Teraflops (1 Flop = 1 Fließkommaoperation pro Sekunde), rangiert er heute (Juni 2005) nur noch auf Platz 52 der weltschnellsten Großrechner [24]. Dabei ist zu beachten, daß diese Spitzengeschwindigkeiten im normalen Rechenbetrieb nicht erreicht werden können, da dies einen Programmcode voraussetzt, der nur aus Fließkommaarithmetik besteht, was in Bezug auf ein realistisches, sinnvolles Programm gesehen nicht erreichbar ist. Die tatsächliche, durchschnittliche Leistung liegt bei einem um ca. 30-50% niedrigeren Wert.

Das dem JUMP zu Grunde liegende Architektur-Konzept entspricht der im letzten Abschnitt erwähnten Mischform aus *Shared-* und *Distributed-Memory*-Modell. 32 Prozessoren des Typs Power4+ mit einer Taktfrequenz von 1.7 GHz teilen sich einen gemeinsamen Arbeitsspeicher von 128 GB und bilden einen Shared-Memory-Knoten (auch Symmetric Multi Processing (SMP) Knoten genannt). Darauf existiert eine 3-stufige Cache Struktur, um die schnelle Datenversorgung der Prozessoren zu gewährleisten und den Engpaß zum Arbeitsspeicher zu entlasten. Jeder Prozessor hat einen internen 96 KB großen Level 1 Cache, jeweils 2 Prozessoren sind wiederum von einen 1.5 MB großen Level 2 Cache umgeben und alle 32 Prozessoren sind schließlich in einen 512 MB großen Level 3 Cache eingebettet, was in Abbildung 5.4 nochmal graphisch veranschaulicht wird. Ein solcher Knoten hat eine Spitzenleistung von 218 Gigaflops. 41 dieser Knoten sind nun in Form eines *Distributed-Memory*-Modells durch ein sogenanntes High-Performance-

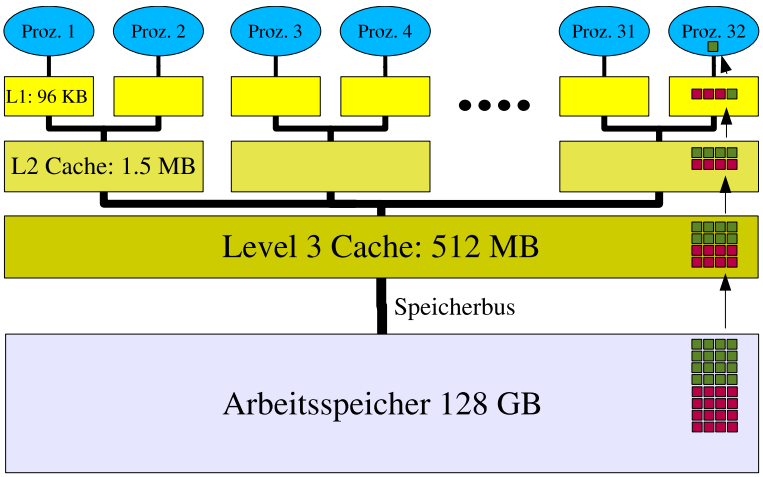

Abbildung 5.4: Cache-Struktur eines JUMP-Knotens: Wird ein Datum vom Prozessor benötigt, wird zunächst im nächsthöheren Cachelevel geschaut, ob die Daten von vorherigen Aufrufen noch vorhanden sind, bevor letztendlich auf den Arbeitsspeicher zugegriffen wird. Werden Daten von unten 'aufgefüllt', dann immer in ganzen Blöcken (grüne Kästchen), um bei Zugriff auf benachbarte Daten (z.B. in Schleifen) möglichst wenig Speicheroperationen ausführen zu müssen.

Switch (HPS) Netzwerk miteinander verbunden. Dieses Netzwerk hat einen gemessenen Datendurchsatz von mindestens 1400 MB pro Sekunde und pro Verbindung zwischen 2 Knoten bei einer Latenzzeit von weniger als 6,5 μs. Insgesamt stehen also 1312 Prozessoren mit einem Gesamt-Arbeitsspeicher von 5,2 Terabyte zur Verfügung. Daneben gibt es noch weitere Serviceknoten, die für die Joborganisation und die Ein- und Ausgabe der Daten zuständig sind.

5.3 Kommunikation zwischen den Prozessoren

Um die Leistung von Multi-Prozessor-Systemen nutzbar zu machen, müssen die Prozessoren miteinander kommunizieren und Daten austauschen. Es haben sich zwei Standards entwickelt, um dem Programmierer die Kontrolle über die einzelnen Prozessoren bei *Shared-* und *Distributed-Memory*-Modellen zu ermöglichen. Diese werden benötigt, um eine möglichst große Portabilität des Codes zu gewährleisten und ihn so auch für Parallelrechner mit anderer Speicherarchitektur nutzbar zu machen.

OpenMP ermöglicht dabei die Steuerung der Prozessoren auf einem SMP-Knoten, mit den *Message Passing Interface (MPI)*-Routinen können sowohl *Shared-* als auch *Distributed-Memory*-Systeme kontrolliert werden. Es handelt sich hierbei um festgelegte Programmbibliotheken, die Unterprogramme, Funktionen, Variablen und Befehle bereitstellen, mit denen die die Kommunikation und der Datenaustausch zwischen den Prozessoren eines Multi-Prozessor-Rechners gesteuert werden können. Am weitesten entwickelt

sind diese Bibliotheken für Fortran und C++.

Eine wichtige Tatsache besteht darin, daß in der Programmierung eines SMP-Knotens OpenMP und MPI in einer Mischform auftreten können. Dabei werden Gruppen von Kommunikationsprozessen zu MPI-*Tasks* zusammengefaßt, die einen Teil des Arbeitsspeichers zugewiesen bekommen und wie eigene logische Knoten behandelt werden, zwischen denen auch Kommunikation zum Datenaustausch eingesetzt werden muß. Die Prozesse innerhalb eines Tasks, *Threads* genannt, werden mit OpenMP kontrolliert und können gemeinsam auf den Speicher des Tasks zugreifen. Dies kann von Vorteil sein, da die Kommunikation zwischen den Threads mit OpenMP versteckt abläuft. Trotzdem lassen sich manche Probleme, trotz zusätzlicher Kommunikation, schneller mit MPI lösen.

Bei der gleichen Anzahl gegebener Prozessoren gibt es daher verschiedene Möglichkeiten, OpenMP und MPI zu kombinieren. Bei 32 verwendeten Prozessoren auf einem Knoten ist jede Kombination möglich, sofern für die Anzahl der MPI-Tasks p und der Threads t gilt: $p \times t = 32$. Ab zwei involvierten Knoten muß es mindestens 2 MPI-Tasks mit 32 Threads geben, maximal können es 64 Tasks mit einem Thread pro Task sein. Anders herum allerdings funktioniert dies nicht. 64 Prozessoren auf 2 Knoten können nicht mit OpenMP als ein Shared-Memory System angesteuert werden. Diese unterschiedlichen Konfigurationen haben einen gravierenden Einfluß auf die Laufzeit des Programmes, worauf in Kapitel 7 eingegangen wird.

Im folgenden soll kurz auf die Vor- und Nachteile beider Schnittstellen und deren Anwendungsgebiete eingegangen werden

5.3.1 OpenMP

Die OpenMP-Befehle koordinieren die Prozessoren eines Tasks, die auf einen gemeinsamen Arbeitsspeicher zugreifen und somit untereinander nur versteckt kommunizieren müssen.

Die am meisten genutzte OpenMP Anwendung ist sicherlich die Parallelisierung von Schleifen. Hierbei werden die Schleifendurchläufe für verschiedene Schleifenindizes auf die Threads verteilt. Das setzt voraus, daß keine zwei Prozessoren gleichzeitig auf ein Speicherdatum zugreifen und die Daten, die in verschiedenen Durchläufen verarbeitet werden, nicht voneinander abhängen dürfen und so jeder Prozessor sein Stück der Schleife unabhängig bearbeiten kann. Iterative Schleifen, bei denen jeder Durchlauf das Ergebnis des Vorherigen benötigt, scheiden daher aus. Hier muß eine andere Form des Parallelisierens gefunden werden.

Ein Problem, das bei der Laufzeitmessung mit verschieden Task-/Thread-Konfigurationen (s. Kapitel 7) deutlich wurde, verursachte allerdings eine Verlangsamung des Programmes. Ist das Datenfeld so groß, daß die Prozessoren nicht ausreichend Platz zur Verfügung haben, um eine lokale Kopie ihres Teiles des Feldes zu erzeugen, bremsen sich die Prozessoren beim Zugriff auf dieses globale Feld gegenseitig aus. Dies liegt an dem Speicherkontrollchip, der nicht genug simultane Lese- und Schreibzugriffe auf den Speicher erlaubt. Bei den in dieser Arbeit durchgeführten Simulationen zeigt sich dieses Phänomen umso ausgeprägter, je mehr Threads zu einem Task zusammengefaßt werden.

5.3.2 MPI

Die Befehle der MPI-Bibliothek ermöglichen die Kommunikation zwischen verschiedenen MPI-Tasks. Dabei ist der maximale Datendurchsatz durch die Bandbreite des Netzwerkes gegeben, das die Knoten verbindet. In der Netzwerk-Topologie des JUMP sind die Knoten in einem 2D-Torus angeordnet, wobei jeder Knoten jeweils vier Verbindungen zu seinen nächsten Nachbarn besitzt, die eine maximale Datengeschwindigkeit von 1,4 GB/s zwischen zwei Knoten erlauben. Beim Verschicken von großen Datenmengen, die im Fall des Simulationsprogrammes bis zu 16 Gigabyte pro Knoten und Programmschritt umfassen, wird das Netzwerk zum Flaschenhals des Systems und trägt erheblich zu der Laufzeit des Programmes bei. Der Datentransfer zwischen MPI-Tasks auf einem physikalischen Knoten läuft schneller ab, da die Daten hier nicht das Netzwerk passieren müssen.

Eine große Rolle für die Übertragungsgeschwindigkeit spielt auch die Paketgröße. MPI schickt bei jedem Paket neben den eigentlichen Informationen auch noch protokollarische Daten, wie zum Beispiel Datentyp und Größe des Paketes. Werden viele kleine Pakete anstatt weniger großer verschickt, macht sich dies in einem Geschwindigkeitsverlust bemerkbar, da bei den kleinen Paketen ein relativ zu der Datenmenge großer sogenannter Overhead von Protokollinformation entsteht, der bei den großen Paketen nicht ins Gewicht fällt. Zusätzlich erhöht sich die Kommunikationsdauer beim Verschicken vieler kleiner Pakete durch die Summe der Latenzzeiten der Senderoutinen, der Zeiten, die vom Aufruf der Routine bis zum eigentlichen Beginn des Datentransfers vergehen. Wird stattdessen ein großes Paket gesendet, trägt die Latenzzeit nur einmal zur Kommunikationsdauer bei.

Damit Prozessoren während des Sende- und Empfangsprozesses nicht bis zum Abschluß der Operation warten müssen, gibt es die Möglichkeit, einen Transfer zu initialisieren und während des Sendens oder Empfangens, das nun nur noch die Aufmerksamkeit des Speicherkontrollchips fordert, Daten zu verarbeiten. Allerdings darf auf die Daten natürlich erst zugegriffen werden, sobald der Datenversand abgeschlossen ist. Trotz des Versteckens der Kommunikation hinter der Rechnung läßt sich dadurch die Kommunikationszeit nicht beliebig verkleinern, da vor jedem Datenaustausch Zeit bei der Synchronisation der Kommunikationsprozesse verloren geht. Ein geschicktes Ablegen der Daten im Speicher und ein Algorithmus, der möglichst wenig Kommunikation benötigt, sind daher das Mittel der Wahl, um Aufwand und Zeit für die Kommunikation möglichst gering zu halten.

6 Simulation eines idealen Quantencomputers

6.1 Funktionalität des Programmes

Die Entwicklung des Quantencomputersimulators im Rahmen dieser Arbeit erfolgte innerhalb der Arbeitsgruppe Quantencomputer am Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich. Ziel war es, ein Simulationsprogramm auf einem Parallelrechner zu erstellen, um das Verhalten von Quantensystemen zu modellieren, die die gegenwärtig technisch realisierbaren Systeme um einige Zehn Qubits an Größe übertreffen.

Grundlegende Vorgaben bei der Programmierung waren, erstens, ein System simulieren zu können, das sich theoretisch auf beliebig viele Quantenbits erweitern läßt, und zweitens, die Portabilität des Programmes zwischen verschiedenen Rechnersystemen zu gewährleisten.

Multiprozessorrechner bieten heutzutage eine Möglichkeit, Speicherressourcen dieser Größenordnung bereit zu stellen und die Simulationszeit eines Algorithmus' in einem realistischen Rahmen zu halten. Dem Parallelisieren des Programmes kommt dabei eine wichtige Bedeutung zu, denn eine minimale Laufzeit kann nur durch eine gleichmäßige Ausnutzung der Rechenleistung aller involvierten Prozessoren erreicht werden. Auch beeinflußt der Datentransfer zwischen den Prozessoren maßgeblich die Laufzeit des Programmes, so daß eine Optimierung der Kommunikation große Priorität besitzt.

Mit dem hier entwickelten Programm können beliebig große Quantensysteme simuliert werden. Dabei ist es nur durch die zur Verfügung stehenden Speicherressourcen begrenzt. Der JUMP-Rechner erlaubt es, Simulationen von Systemen mit maximal 37 Qubits durchzuführen (s. Kapitel 5.2).

Das Programm ist in der Lage, auf einem gegebenen Anfangszustand des Quantensystems einen beliebigen Algorithmus durchzuführen, der sich aus den elementaren 1- und 2-Qubit-Operationen X, Y, Phase, H, CNOT und C-Phase in modularer Form zusammensetzen läßt. Jedes dieser elementaren Gatter wird programmtechnisch auf verschiedene Weise realisiert, da der Rechenaufwand und das Kommunikationsaufkommen stark von der Art der Operation abhängen.

Schwerpunkt der hier vorliegenden Arbeit war die Implementierung und Optimierung der Rechen- und Kommunikationsroutinen für die einzelnen Quantengatter, da sich bei Algorithmen mit mehreren Hundert Operationen schon kleine Laufzeitunterschiede einer

Operation bemerkbar machen. Im Hinblick auf die Simulation realistischer Quantencomputer gewinnt die Optimierung der Operationslaufzeiten sogar noch an Bedeutung, da hier der Rechenaufwand zur numerischen Berechnung von $\mathbf{U} = e^{-i\mathbf{H}t}$ um Größenordnungen höher ist, als im Fall des idealen Quantencomputers. Ziel war es, zu untersuchen, ob die Rechenzeit des Simulationsprogrammes bei ansteigender Systemgröße linear skaliert, und so Simulationen großer Systeme in annehmbarer Zeit durchgeführt werden können. Dies ist exemplarisch am Beispiel des Hadamard- und des CNOT-Gatters geschehen (s. Kapitel 7).

Als Erweitung des idealen Systems wurden darüber hinaus parametrisierte Gatterfehler sowie Dekohärenzeffekte in die Simulation eingebaut, und deren Auswirkungen auf das Hadamard-Gatter untersucht (s. Kapitel 8).

6.2 Programmablauf

Zur besseren Beschreibung werden die Quantenbits eines Quantencomputers mit 0 beginnend durchnumeriert. Einem 36-Qubit System stehen daher die Qubits 0-35 zur Verfügung. Befindet sich das System also z.B. im Basiszustand $|...100\rangle$, dann sind Qubit 0 und 1 im Zustand $|0\rangle$ und Qubit 2 im Zustand $|1\rangle$.

6.2.1 Parameterübergabe und Initialisierung

Beim Start des Programmes müssen folgende Parameter an das Programm übergeben werden:

- ***Anzahl der Qubits des Systems***: Die Anzahl der Qubits des Systems (`nbits`) bestimmt die Größe des Zustandsvektors und damit auch den benötigten Arbeitsspeicher. Indirekt wird so auch die minimale Anzahl der benötigten Rechenknoten festgelegt.

- ***Systemkonfiguration***: Die Anzahl der MPI-Tasks und der darin enthaltenen OpenMP-Threads auf jedem Knoten legt fest, wie viele Einträge des Zustandsvektors einem MPI-Task zugeordnet werden (`spn`). Dadurch steht fest, welche Qubitkombinationen welchem Kommunikationstyp (Kap. 6.6) entsprechen.

- ***Anfangszustand***: Das simulierte Quantensystem startet in dem wohldefinierten Basiszustand $|000...000\rangle$. Aus diesem Zustand kann es durch Gatteroperationen in jeden beliebigen Zustand transformiert werden. Die bei einem realen Quantencomputer erforderliche Initialisierungsphase, die alle Qubits in einen bekannten Anfangszustand bringt, entfällt.

- ***Algorithmus***: Der auszuführende Algorithmus wird aus einer Datei eingelesen. Ein Eintrag dieser Datei beinhaltet die Art der Operation und die Nummer des oder der Qubits, auf die die Operation wirken soll.

- ***Paketgröße für Kommunikation***: Die Paketgröße hat starken Einfluß auf die Datenübertragungsgeschwindigkeit zwischen den MPI-Prozessen (s. Kapitel 5.3.2). Bei der dynamischen Lastverteilung (s. Kapitel 6.8.3) kann dabei eine optimale Paketgröße gefunden werden, die möglichst wenig Kommunikationsoverhead (große Pakete) mit möglichst feiner Lastverteilung kombiniert (kleine Pakete).

- ***Anzahl der gleichzeitig gesendeten Pakete***: Die Angabe ist nur bei der dynamischen Lastverteilung (s. Kapitel 6.8.3) nötig und dient dort, wie die Paketgröße, zur Minimierung der Kommunikationsdauer.

- ***Dekohärenzwahrscheinlichkeit***: Mit dem aus Kapitel 4.4.1 bekannten Parameter p wird die Wahrscheinlichkeit dafür angegeben, daß nach einer Operation ein Bit-Flip, eine Phase-Flip oder eine Kombination aus beidem auf das Qubit angewendet wird.

- ***Standardabweichung des Gatterfehlers***: Der Parameter σ als Standardabweichung legt die Größe des Gauss-verteilten Fehlers fest, um den der wirkliche Drehwinkel eines fehlerhaften Gatters vom idealen Fall abweicht (s. Kapitel 4.4.2).

6.2.2 Ausführen des Algorithmus'

Nach der Initialisierungsphase des Programmes, die das Anlegen aller benötigten Variablen beinhaltet, wird der Algorithmus gatterweise abgearbeitet. Bei jedem Schritt des Algorithmus' wird das entsprechende Unterprogramm aufgerufen, das die Operation durchführt. Diesem werden die Indizes der involvierten Qubits sowie der Zustandsvektor übergeben.

Bei der Simulation eines idealen Quantencomputers wird jede Operation als ein Zeitschritt betrachtet. Die Evolution verläuft zwar nach der Schrödinger-Gleichung, da aber keine Wechselwirkungen von außen und unter den Spins berücksichtigt werden, ist es nicht notwendig, die einzelnen Operationen in kleine Zeitschritte aufzuteilen, um Wechselwirkungen detailgenau zu simulieren.

Dieser Ablauf kann als Grundlage zur Implementierung realistischer Quantencomputermodelle dienen, bei dem eine dynamische Zeitentwicklung beginnend mit einem generischen Ising-Modell angestrebt ist. Hierbei wird die Zeitentwicklung durch numerische Näherungsverfahren wie z.B. den Suzuki-Trotter-Algorithmus implementiert, wie auch schon für kleinere Systeme geschehen in [25, 26].

Bei der Simulation von fehlerhaften Gattern wird diese ideale Zeitentwicklung beibehalten, da es sich in diesem Fall um einen statistischen Fehler handelt, bei dem es keine Rolle spielt, ob dieser in einem Schritt nach der Operation hinzugefügt wird, oder in mehreren Einzelschritten entsteht, die sich zu einem Gesamtfehler aufaddieren.

Die Auswirkung der Dekohärenz wird bei dem Programm nach jeder abgeschlossenen Operation simuliert, indem der Zustand des Systems einer Transformation nach Gleichung 4.17 unterzogen wird, so daß auch hier keine Unterteilung in explizite Zeitschritte nötig ist.

6.2.3 Messung des Zustandsvektors

Nach Abschluß des Algorithmus' wird der Zustandsvektor an eine Meßroutine übergeben, die für jedes Qubit den Erwartungswert im Bezug auf die drei Spin-Operatoren $\mathbf{S}_\alpha$ ausrechnet und diesen zusammen mit den Wahrscheinlichkeiten, sich in einem der Basiszustände der Operatoren zu befinden, ausgibt. Hierbei ist man natürlich nicht der Restriktion des quantenmechanischen Messprozesses unterworfen, denn die Amplituden des Zustandsvektors sind wohlbekannt.

Bei den Messungen im nicht-idealen Fall werden nach jeder Operation Daten des Zustandsvektors ausgegeben, um eine zeitliche Entwicklung der Fehlereffekte aufzeichnen zu können. Die Berechnung der Differenz zweier Zustände zur Fehlerbestimmung geschieht dabei durch Nachbearbeitung der Daten.

6.2.4 Zeitmessung

Zu jeder aufgerufenen Operation wird die benötigte Zeitdauer lokal auf jedem Prozessor aufaddiert. Dies geschieht zum einen in einer funktionellen Einteilung unter Berücksichtigung der Art der Operation, des Kommunikationsmusters und der Prozessoraufgabe innerhalb der Lastverteilung, und zum anderen nach den Indizes der beteiligten Quantenbits. Zusammen mit der Anzahl der Operationen lassen sich so Aussagen über die durchschnittlichen Rechenzeiten in Abhängigkeit einzelner Parameter treffen.

Die erhaltenen Meßwerte werden nach Abschluß des Programmes durch Shell-Skripte zu den benötigten Mittelwerten für die gewünschten Teilbereiche zusammengefaßt.

Die Zeit einer Operation setzt sich dabei aus drei Teilen zusammen, der Zeit für die Rechenroutinen, der für die Kommunikationsroutinen und einer Restsystemzeit, die für die Ausführung der Instruktionen benötigt wird, die sich keiner der beiden Kategorien zuordnen lassen. Bei den Simulationen wird diese Systemzeit nicht explizit gemessen, sondern berechnet sich aus der Differenz der Gesamtlaufzeit und der Summe der Rechen- und Kommunikationszeit.

6.3 Speicherbedarf des Systems

Bei einem System mit n Qubits besitzt der Zustandsvektor dieses Systems 2^n Einträge. Jeder Eintrag des Vektors beinhaltet die komplexe Amplitude des dazugehörigen Basiszustandes. Real- und Imaginärteil werden dabei in zwei Variablen des Typs `double_precision` abgespeichert. Eine Variable diesen Typs belegt einen Speicher von 8 Byte und ermöglicht die Darstellung einer Floatingpoint-Zahl mit einer Genauigkeit von 15 Dezimalstellen. Eine solche Genauigkeit ist nötig, um die bei numerischen Operationen entstehenden Rundungsfehler, die sich über den Algorithmus hinweg akkumulieren können, möglichst gering zu halten.

Insgesamt belegt der Zustandsvektor einen Speicherplatz von $2^n \cdot 2 \cdot 8$ Bytes. Jedem physikalischen Rechnerknoten bleiben von den 128 GB gemeinsamen Speichers 112 GB zum Ausführen von Programmen, da 16 GB für das Betriebssystem reserviert sind. Der verfügbare Speicher kann nun unter den MPI-Tasks aufgeteilt werden. Bei der maximalen

Anzahl von 32 MPI-Tasks pro Knoten steht jedem Prozessor also ein Arbeitsspeicher von 3,5 GB zu, in dem alle Variablen des Programmes sowie das Programm selbst abgelegt werden.

Die bei der Simulation durchgeführten Operationen wirken auf einen Zustandsvektor, nicht auf eine Dichtematrix, da dies den Rahmen des verfügbaren Speichers sprengen würde. Durch diese Einschränkung ist das System auf reine Zustände limitiert. Einzelne Zustandskomponenten gemischter Zustände müssen daher durch mehrere unabhängige Programmaufrufe separat berechnet werden.

Ein System mit 32 Qubits benötigt 64 GB, um den Zustandsvektor im Arbeitsspeicher abzulegen. Daneben werden weitere 32 GB reserviert, die als Zwischenspeicher für nicht lokale Elemente anderer MPI-Tasks dienen. Dies ist das größte System, das auf einem Knoten simuliert werden kann, da hier 96 der verfügbaren 112 Gigabyte Arbeitsspeicher benutzt werden. Jede weitere Vergrößerung des simulierten Systems erfordert eine Verdoppelung des verfügbaren Speicherplatzes in Form von zusätzlichen Knoten. Für ein System von 33 Qubits werden zwei Knoten benötigt, für 34 Qubits vier Knoten usw. Das größte berechnete System umfaßte 36 Qubits auf 16 Knoten mit einem Zustandsvektor der Größe 1 TB.

Die maximale Größe des Systems ist daher durch den verfügbaren Speicher begrenzt, nicht durch die Rechenleistung.

Der Zustandsvektor des Gesamtsystems muß also in Pakete der Größe $2^{32} \cdot 2 \cdot 8 = 64$ GB zerlegt werden, die jeweils auf einem Knoten liegen. Innerhalb des Knotens werden die 64 GB auf die existierenden MPI-Prozesse verteilt, so daß der lokale Teil des Zustandsvektors eines MPI-Prozesses entsprechend kleiner ist.

Der Zustandsvektor wird in den Variablen `psi_r` und `psi_i` gespeichert. Um Teile des Zustandsvektors anderer Tasks speichern zu können, ohne den lokalen Zustand überschreiben zu müssen, verfügt jeder Task über 2 zusätzliche Speicher `chi1_r`, `chi1_i`, `chi2_r` und `chi2_i` der Größe $\frac{1}{4}$`spn`. Der lokale Teil des Zustandsvektors zusammen mit den beiden Kommunikationsspeichern belegen somit $\frac{6}{7}$ des verfügbaren Arbeitsspeichers eines jeden MPI-Tasks und somit auch des gesamten Systems.

6.4 Aufteilung des Zustandsvektors unter den MPI-Tasks

Wie gesehen umfaßt das größtmögliche simulierbare System auf einem physikalischen Knoten, begrenzt durch den Speicherplatz, 32 Qubits. Nun kommt es darauf an, wieviele MPI- und OpenMP-Prozesse auf diesem Knoten gestartet werden. Angenommen, auf einen Rechnerknoten wird nur ein MPI-Prozess gestartet, dann hat dieser Zugriff auf den gesamten Zustandsvektor im Speicher. Bei zwei MPI-Prozessen wird der verfügbare Speicher geteilt und somit hat jeder Prozess nur noch direkten Zugriff auf die Hälfte des gesamten Zustandsvektors, und zwar der erste auf die Einträge $|000\ldots\rangle$ bis $|011\ldots\rangle$ und der zweite auf die Einträge $|100\ldots\rangle$ bis $|111\ldots\rangle$ (s. Abbildung 6.1). Damit ist jeder Task weiterhin in der Lage, Operationen auf allen Qubits außer dem höchstwertigen, un-

abhängig von den anderen Prozessoren, auszuführen. Natürlich müssen trotzdem beide Prozesse alle Einträge ihres Zustandsvektors der gleichen Matrixmultiplikation unterziehen. Bei Operationen, die das höchstwertige Qubit bearbeiten, ist Kommunikation notwendig, da in diesem Fall Daten aus verschiedenen Speicherbereichen miteinander verknüpft werden müssen. Die Art und das Volumen dieser Kommunikation hängen in erster Linie von der Art der Operation ab und beeinflussen wesentlich die Laufzeiten des Programmes, da bei großen zu versendenden Datenmengen die Kommunikation weitaus langsamer abläuft als die Prozessoren Daten verarbeiten können und so Wartezeiten entstehen können. Je mehr MPI-Prozesse auf einem Knoten laufen, desto eher ist Kom-

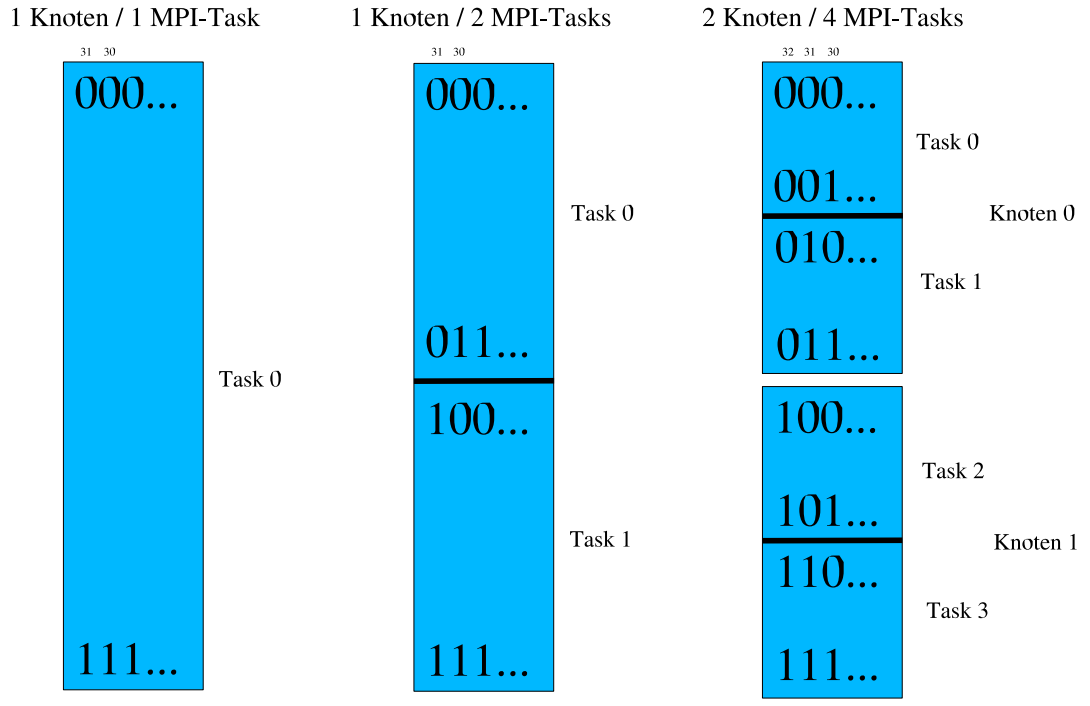

Abbildung 6.1: Aufteilung des Zustandsvektors bei verschieden vielen MPI-Prozessen

munikation nötig, oder anders herum: Je mehr Prozesse, desto weniger Qubits können ohne Kommunikation bearbeiten werden. Bei 32 MPI-Prozessen auf einem Knoten und einer Systemgröße von mindestens 32 Qubits könnte jeder Prozessor Operationen auf den Qubits 0-26 selber durchführen, für Operationen auf den Qubits 27-31 wäre Kommunikation zwischen den Tasks auf einem Knoten nötig. Für mehr als 32 Qubits gilt innerhalb der Knoten natürlich das gleiche. Die Qubits mit den Nummern größer als 32 liegen über mehrere Knoten verteilt, sobald diese Qubits bearbeitet werden sollen, ist Kommunikation zwischen den Knoten unumgänglich. Da die Kommunikation in der Regel im Vergleich zur Rechengeschwindigkeit der Prozessoren vergleichsweise langsam abläuft, versucht man möglichst, diese Kommunikationsengpässe zu umgehen; dies kann durch geschicktes Verteilen der Daten im Speicher passieren oder auch durch eine geeignete Implementierung des Algorithmus', bei der, falls möglich, die meisten Operationen

auf den Qubits 0-26 ausgeführt werden und eine Operation, die Qubit Nummer 35 anspricht, möglichst selten vorkommt. Daß es sich dennoch lohnt, auch viele MPI-Prozesse innerhalb eines physikalischen Knotens zu starten und so ein hohes Kommunikationsaufkommen in Kauf zu nehmen, wird durch die durchgeführten Testmessungen bestätigt (s. Kapitel 7). Durch das Aufteilen des Zustandsvektors auf mehr MPI-Prozesse wird dieser in kleinere lokale Abschnitte unterteilt. Die maximale Entfernung zweier Vektoreinträge, auf die die Prozessoren eines MPI-Prozesses zugreifen müssen, verringert sich daher, was den Zugriff auf die Daten beschleunigen kann. Weiterhin sinkt durch die Erhöhung der MPI-Prozesse die Anzahl der darin enthaltenen Prozessoren. Dies führt dazu, daß der Zugriff auf den Speicherbereich eines Prozesses bei einer OpenMP-Parallelisierung weniger stark segmentiert erfolgen kann und so einen weiteren Geschwindigkeitsgewinn ermöglicht.

6.5 Reduktion der Operationsmatrizen auf 2x2- und 4x4-Matrizen

Wie in Abschnitt 6.3 gesehen, ist der Speicherplatz auf einem Knoten durch den Zustandsvektor schon zu $\frac{6}{7}$ belegt. Eine lineare Abbildung, die auf den Vektor angewendet werden soll, entspräche dabei einer Matrix der Größe $2^{32} \times 2^{32}$ mit komplexen Einträgen. Bei 16 Byte pro Eintrag (Real- und Imaginärteil $\grave{a}$ 8 Byte) wird schnell klar, daß der gegebene Speicherplatz nicht ausreicht, auch nur eine einzige Matrix dieser Größe zu speichern, verschiedene Matrizen für verschiedene Operationen schon gar nicht. Daher muß eine andere Methode gefunden werden, die Operationen auf dem System durchzuführen.

Dazu schaut man sich am Beispiel des NOT-Gatters in einem System bestehend aus 3 Qubits an, wie die Operationsmatrizen des NOT-Gatters aussehen, abhängig davon, auf welches Quantenbit sie wirken sollen.

$$\begin{aligned} \mathbf{1} \otimes \mathbf{1} \otimes \mathbf{X} \quad & \text{für Qubit 1} \\ \mathbf{1} \otimes \mathbf{X} \otimes \mathbf{1} \quad & \text{für Qubit 2} \\ \mathbf{X} \otimes \mathbf{1} \otimes \mathbf{1} \quad & \text{für Qubit 3} \end{aligned} \tag{6.1}$$

Bei einer NOT-Operation auf Qubit 2 des Beispielsystems nimmt der Operator nach

Kapitel 3.7 folgende Gestalt an:

$$NOT_2 = \mathbf{1} \otimes \mathbf{X} \otimes \mathbf{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{6.2}$$

Bei jeder der Operationsmatrizen handelt es sich um sehr dünn besetzte unitäre Matrizen, die jeweils nur einen Eintrag pro Zeile bzw. Spalte haben, alle anderen Einträge haben den Wert Null, d.h. alle Einträge des Zustandsvektors werden entweder mit sich selbst oder mit einem anderen Eintrag verknüpft. Es läßt sich eine gewisses Abhängigkeit

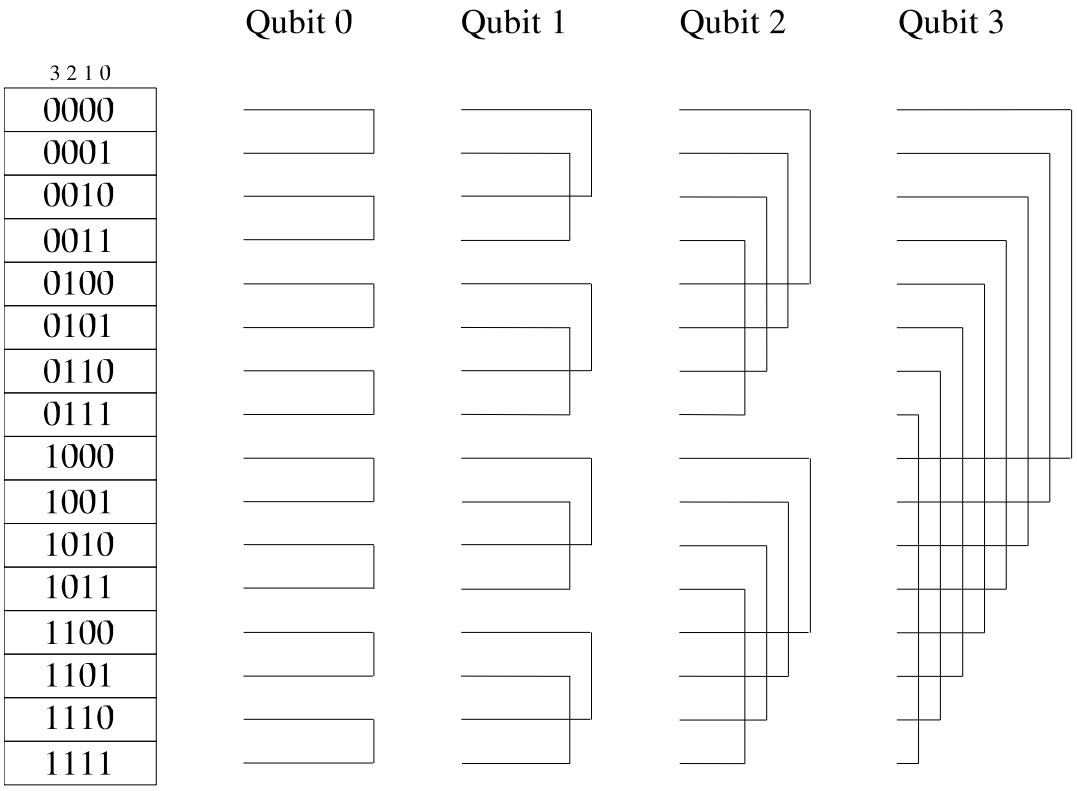

Abbildung 6.2: Zusammengehörige Einträge bei 1-Qubit-Operationen

zwischen dem Quantenbit, auf das die Operation wirken soll und den Indizes der Zustandes in binärer Darstellung erkennen, die jeweils miteinander verknüpft werden müssen.

Bei einer Operation auf Qubit 0 werden hintereinanderliegende Paare von Basisvektoren miteinander verknüpft. Bei einer Operation auf Qubit 1 werden immer einen Zustand voneinander entfernte Einträge paarweise miteinander verknüpft, bei Operationen auf Qubit 2 beträgt die Entfernung 4 Einträge (s. Abbildung 6.2). Insgesamt kann also jede der NOT-Operationen aus Gleichung (6.1) in Operationen aufgeteilt werden, die jede für sich nur 2 Basisvektoren miteinander verknüpfen und so durch 2×2-Matrizen darstellbar sind [28].

Allgemein werden durch eine 1-Qubit-Operation immer die Basiszustände paarweise miteinander verknüpft, deren binärer Index an allen Stellen übereinstimmt, außer an der Position des zu bearbeitenden Qubits:

$$\begin{aligned} &|\ldots * *0 * * \ldots\rangle \\ &|\ldots * *1 * * \ldots\rangle \end{aligned} \tag{6.3}$$

Ein $*$ stellt hierbei einen übereinstimmenden Wert dar.

Eine 1-Qubit-Operation, angewendet auf Qubit Nummer 2 des Beispielsystems, verknüpft also die Zustände $|000\rangle$ und $|010\rangle$, $|001\rangle$ und $|011\rangle$, $|100\rangle$ und $|110\rangle$ sowie $|101\rangle$ und $|111\rangle$ miteinander. Hierfür ist nur die elementare 2×2-Operationsmatrix notwendig, die auf jedes Eintragspaar angewendet werden muß.

Der Vollständigkeit halber soll hier kurz erwähnt werden, daß sich, anders herum, durch Umsortieren des Zustandsvektors Operationen auf verschiedenen Qubits durch dieselbe Matrix bewerkstelligen lassen. Die Operation $\mathbf{1} \otimes \mathbf{1} \otimes \mathbf{X}$ kann also eine NOT-Operation auf allen drei Qubits des Systems bewirken, je nachdem, ob der Zustandsvektor in der Form

$$\begin{pmatrix} a_{000} \\ a_{001} \\ a_{010} \\ a_{011} \\ a_{100} \\ a_{101} \\ a_{110} \\ a_{111} \end{pmatrix}, \quad \begin{pmatrix} a_{000} \\ a_{010} \\ a_{001} \\ a_{011} \\ a_{100} \\ a_{110} \\ a_{101} \\ a_{111} \end{pmatrix} \quad \text{oder} \quad \begin{pmatrix} a_{000} \\ a_{100} \\ a_{001} \\ a_{101} \\ a_{010} \\ a_{110} \\ a_{011} \\ a_{111} \end{pmatrix} \tag{6.4}$$

vorliegt. Auf Grund der Tatsache, daß eine Umsortierung des Zustandsvektors bei jeder Operation jedoch sehr viel Rechenzeit und Kommunikation beansprucht, falls die zu verknüpfenden Zustände verteilt auf verschiedenen Knoten liegen, wird ein anderes Verfahren benutzt, die gewünschten Zustände zusammenzubringen, auf das in Kapitel 6.6 eingegangen wird.

Im Fall der 2-Qubit-Operationen läßt sich analog zum 1-Qubit-Fall die Operationsmatrix des gesamten Systems durch 4×4-Matrizen ausdrücken, durch die jede 2-Qubit-Operation elementar darstellbar ist, denn die Matrix der Gesamtoperation ist wieder nur mit einem Eintrag pro Zeile und Spalte besetzt, so daß auch hier nur maximal vier Einträge des Zustandsvektors miteinander verknüpft werden. Welche vier Einträge des Zustandsvektors das sind, hängt auch hier von den Indizes der zwei involvierten Quantenbits ab. Abbildung 6.3 verdeutlicht die Zusammengehörigkeit von jeweils vier Einträgen

des Zustandvektors für einige Qubitkombinationen. Auch hier muß die 4×4-Matrix auf

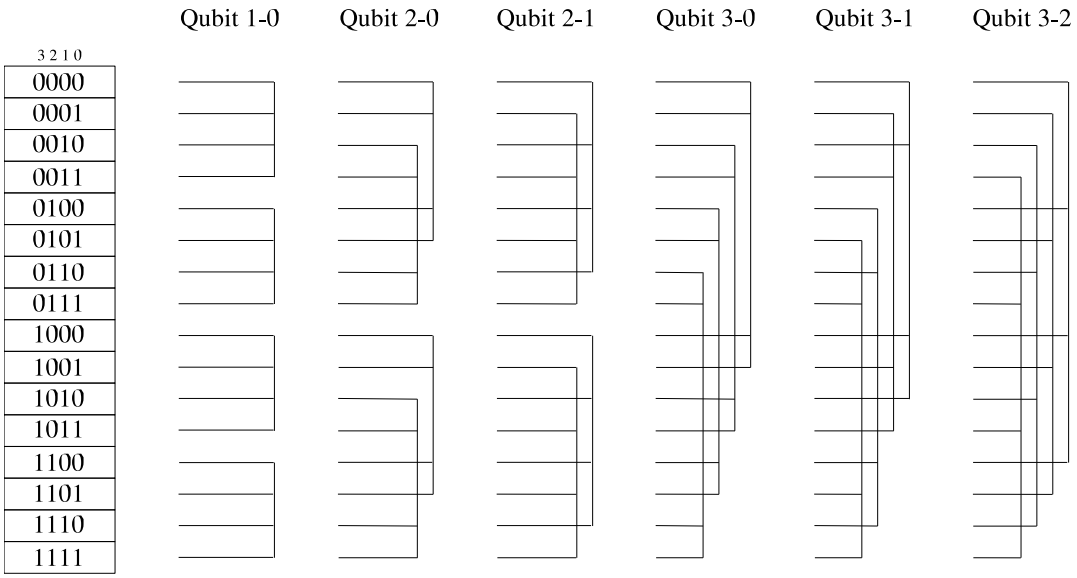

Abbildung 6.3: Zusammengehörige Einträge bei 2-Qubit-Operationen

jede Vierergruppe von Einträgen des Zustandsvektors angewendet werden, deren binäre Indizes an allen Stellen übereinstimmen, sich an den Positionen der Qubits, auf die die Operation wirken soll, jedoch unterscheiden. Die zusammengehörigen Einträge des Zustandsvektors sind also

$$\begin{aligned} &|\ldots * *0 * *0 * * \ldots\rangle \\ &|\ldots * *0 * *1 * * \ldots\rangle \\ &|\ldots * *1 * *0 * * \ldots\rangle \\ &|\ldots * *1 * *1 * * \ldots\rangle \end{aligned} \tag{6.5}$$

wobei ein $*$ übereinstimmende Werte darstellt.

Zusammenfassend kann man sagen, daß es möglich ist, jede Multiplikation einer $n \times n$-Matrix mit dem Zustandsvektor des Systems durch $\frac{n}{2}$- bzw. $\frac{n}{4}$-faches Anwenden von 2×2- bzw. 4×4-Matrizen auf alle Zweier- bzw. Vierergruppen von Einträgen des Zustandsvektors zu ersetzen, solange die Matrizen entsprechend schwach besetzt sind. So läßt sich das Speichern der gesamten Operationsmatrix für ein zusammengesetztes System umgehen. Der Speicheraufwand für 2×2- und 4×4-Matrizen ist dagegen verschwindend gering.

Insgesamt werden durch dieses Verfahren sogar weniger numerische Rechenoperationen benötigt, um die Matrixmultiplikation eines Rechenschrittes durchzuführen. Multiplikation eines n-dimensionalen Vektors mit einer $n \times n$-dimensionalen Matrix erfordert im Allgemeinen n^2 Multiplikationen und Additionen. Die $\frac{n}{2}$- bzw. $\frac{n}{4}$-fache Anwendung von 2×2- bzw. 4×4- Matrizen erfordert dagegen nur $2n$ bzw. $4n$ Multiplikationen

und Additionen. Speziell wird die Matrixmultiplikation nur für die von Null verschiedenen Elemente in Abhängigkeit der Gatteroperation fest im Programm implementiert, welches die Anzahl der durchzuführenden Rechenoperationen bei den hier auftretenden, schwach besetzten Matrizen nochmals reduziert.

6.6 Nomenklatur für Kommunikationsprozesse

Im folgenden wird das Schema beschrieben, nach dem zusammengehörige Einträge des Zustandsvektors bei einer bestimmten Operation in Abhängigkeit von den involvierten Qubits im Speicher verteilt liegen. Dabei werden programmspezifische Größen eingeführt, die für die Kommunikationsroutinen nötig sind. Die Unterteilung dieser Routinen in verschiedene Gruppen motiviert sich durch die unterschiedlichen Anforderungen der einzelnen Operationen an Rechen- und Kommunikationsaufwand und dient zum besseren Verständnis der Laufzeitanalysen.

Wie in Kapitel 6.5 beschrieben, muß bei einer 1-Qubit-Operation auf Qubit x die Operationsmatrix auf die Gruppen von Einträgen angewendet werden, deren Indizes sich im Bitmuster an der Stelle x unterscheiden, sonst jedoch identisch sind. Bei einer 2-Qubit-Operation auf den Qubits x und y muß die Operationsmatrix entsprechend auf die Gruppen von Einträgen angewendet werden, deren Indizes sich nur an den Stellen x und y voneinander unterscheiden.

Dabei werden den Einträgen innerhalb dieser Zweier- bzw. Viergruppen die Positionen 1-2 bzw. 1-4 zugeordnet. Bei den 1-Qubit-Operationen entsprechen die Einträge der Position 1 den Amplituden mit den Indizes $a_{...**0**...}$, die der Position 2 den Amplituden der Indizes $a_{...**1**...}$. Analog dazu verhält es sich bei den 2-Qubit-Operationen. Der Position 1 entsprechen dabei die Einträge mit den Indizes $a_{...**0**0**...}$, der Position 2 die Einträge der Indizes $a_{...**0**1**...}$ usw. Hierbei stellen die $*$ nun beliebige Werte dar. Die Charakterisierung der Zustände nach ihren Positionen vereinfacht nicht nur die Bezeichnung der Zustände, sondern auch die zum Auffinden und Verschicken von Einträgen benutzten Routinen.

Zusammengehörige Einträge bei einer 1-Qubit-Operation auf Qubit 0 liegen paarweise hintereinander im Speicher. Erhöht sich der Index des Qubits, auf das die Operation wirkt, um 1, so verdoppelt sich der Abstand zwischen den Eintragspaaren der Positionen 1 und 2. In der graphischen Darstellung der zusammengehörigen Einträge überlappen sich nun die Verbindungslinien zweier Eintragspaare. Weiteres Erhöhen des Qubitindexes hat eine weitere Verdopplung des Eintragsabstands zur Folge (s. Abbildung 6.2).

Im Fall der 2-Qubit-Operationen gestaltet sich dies etwas komplizierter: Für den Fall, daß die Operation auf die Qubits 1 und 0 wirkt, werden jeweils vier hintereinander liegende Zustände benötigt. Wird nun der Index des ersten Qubits von 1 auf 2 erhöht, ändert sich das Muster so, daß sich der Abstand zwischen den Positionen 1 und 3 der jeweiligen Vierergruppen von zwei auf vier Einträge erhöht. Gleichzeitig überlappen sich in der Darstellung nun jeweils zwei Vierergruppen. Ein weiteres Erhöhen des ersten Qubitindex bringt eine weitere Verdopplung des Abstands der Positionen 1 und 3 sowie ein Überlappen von vier Eintragsgruppen mit sich, der Abstand zwischen den Positionen

1 und 2 sowie 3 und 4 bleibt jedoch gleich. Beim Erhöhen des Index des zweiten Qubits um eins stellt man nun eine Verdoppelung des Abstands zwischen den Positionen 1 und 2 sowie 3 und 4 fest. Gleichzeitig liegen nun mehrere Einträge gleicher Positionen hintereinander (s. Abbildung 6.3).

Einige Variablen des Programmes sind für das Verständnis der Funktionsweise von zentraler Bedeutung, daher soll auf sie kurz an Hand einer 2-Qubit-Operation eingegangen werden. Die Definition der Größen bei 1-Qubit-Operationen erfolgt analog mit dem einzigen Unterschied, daß die Positionen 1 und 2 den 2-Qubit-Positionen 1 und 3 entsprechen. Die Positionen 2 und 4 entfallen hier. Benennt man den Index des ersten

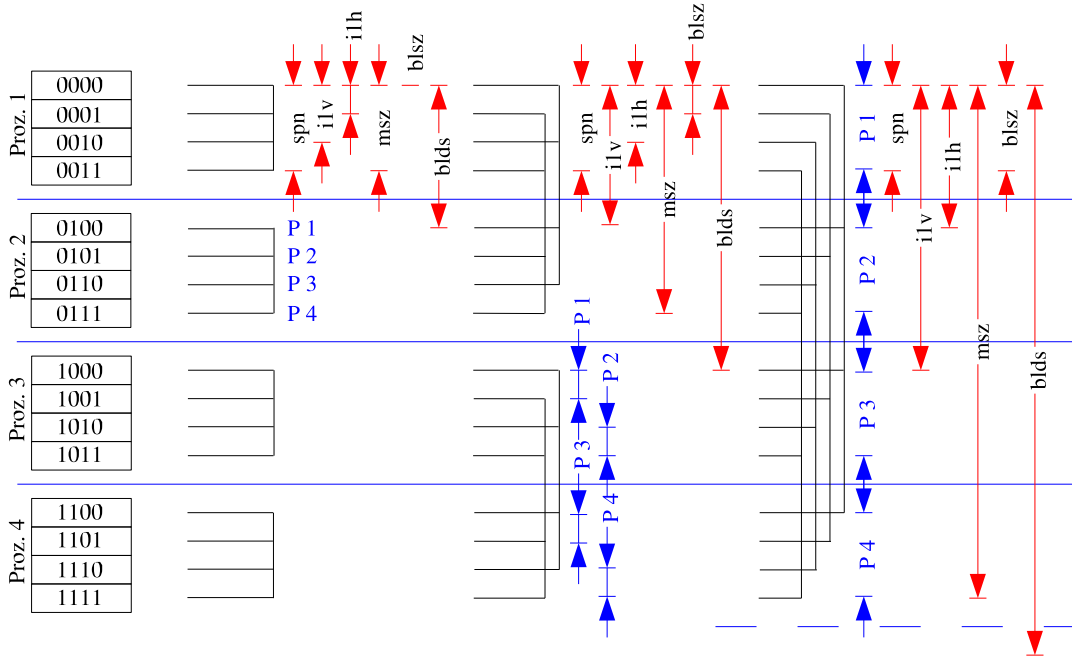

Abbildung 6.4: Charakteristische Mustergrößen der Kommunikationsprozesse. Die Abstände i1v, i1h und blds sind hierbei um einen Eintrag verlängert dargestellt, so daß die Position des um einen Mitten-, Seiten- oder Blockabstand entfernten Eintrags ersichtlich wird.

Qubits mit x, beginnend von 1 bis $n-1$ und den Index des zweiten Qubits mit y mit den Werten 0 bis $x-1$, ergibt sich der Mittenabstand zwischen Position 1 und 3 einer Vierergruppe als Funktion von x zu $\texttt{i1v} = 2^x$, der Seitenabstand zwischen den Positionen 1 und 2 sowie 3 und 4 als Funktion von y zu $\texttt{i1h} = 2^y$. Dabei ist zu beachten, daß der Seitenabstand eines Musters nie größer als die Hälfte des Mittenabstands sein kann, was der Bedingung $y \leq x - 1$ entspricht. Hieraus lassen sich alle weiteren wichtigen Größen des Musters ableiten. Die Namensgebung der einzelnen Größen gleicht der der Variablen, die auch im Programm Verwendung finden. Die Zahl der hintereinander liegenden Einträge einer Position ist $\texttt{blsz} = 2^y$ und wird Blocklänge genannt. Sich überlappende Gruppen von Einträgen werden zu einem Muster zusammengefaßt, das die Länge $\texttt{msz} = 2 \cdot \texttt{i1v}$

hat. Innerhalb eines Musters gibt es `blno` = `msz`/`blsz`/4 Blöcke gleicher Position, deren erste Einträge um jeweils `blds` = 2 · `i1h` auseinanderliegen. Abbildung 6.4 verdeutlicht diese Größen.

Auf einem physikalischen Knoten finden 2^{32} Einträge des Zustandsvektors Platz. Abhängig von der Zahl der MPI-Prozesse auf einem Knoten p besitzt jeder MPI-Task einen lokalen Zustandsvektor der Länge `spn` = $2^{32-\log_2 p}$. Bei einer maximalen Aufspaltung, bei der jeder MPI-Task einen Prozessor enthält (p=32), beträgt die Länge des lokalen Zustandsvektors 2^{27} Einträge.

Bei der Aufteilung des Zustandsvektors auf verschiedene Tasks kommt es bei verschiedenen Operationen dazu, daß sich Einträge des Zustandsvektors im Speicher verschiedener MPI-Tasks befinden, die zur Anwendung der Operationsmatrix in denselben Speicher transportiert werden müssen. Hier erlaubt ein Vergleich der Abstände innerhalb eines Musters mit der Anzahl der lokal abgelegten Vektoreinträge, die auftretenden Fälle für 1- und 2-Qubit-Operationen zu klassifizieren (s. Abbildungen 6.5 und 6.6). Ist `spn` ≥ 2 · `i1v`, dann befinden sich alle zusammengehörigen Einträge im Speicher

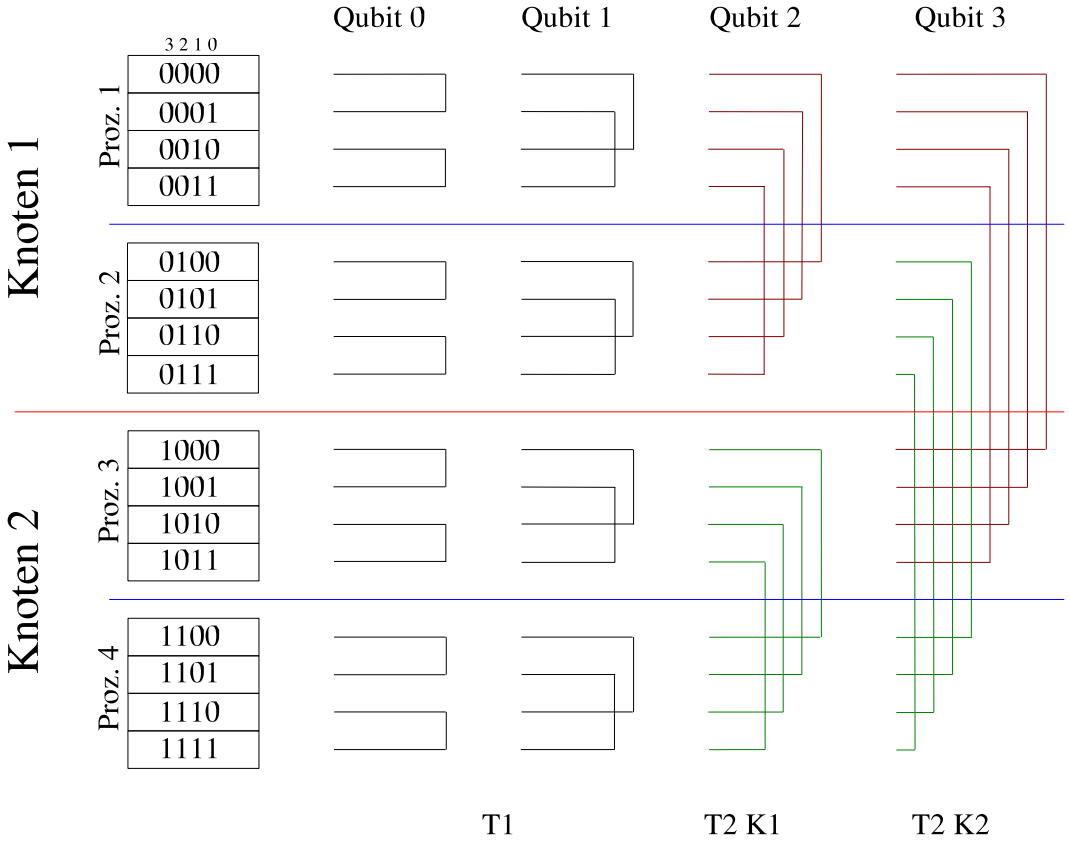

Abbildung 6.5: Beispiele von 1-Qubit Kommunikationstypen

desselben MPI-Tasks, der die Operation unabhängig von den anderen ausführen kann. Dieser Fall wird mit T1 bezeichnet. Ist `spn` > 2 · `i1v`, liegen die Einträge einer 2er- oder

4er-Gruppe auf verschiedenen Tasks. Bei einer 2-Qubit-Operation entscheidet nun der Seitenabstand, ob Position 1 und 2 sowie 3 und 4 jeweils zusammen im Speicher eines Tasks liegen. Falls gilt `spn` > `i1h`, gibt es immer zwei zusammengehörige Tasks, die jeweils Position 1 und 2 sowie Position 3 und 4 einer Vierergruppe von Einträgen besitzen. Der Fall, bei dem Datentransfer zwischen zwei Tasks nötig ist, um zusammengehörige Einträge für eine Operation zusammenzubringen, wird mit T2 abgekürzt. Ist bei 2-Qubit-Operationen auch `i1h` ≥ `spn`, befinden sich alle Positionen einer Gruppe von Einträgen auf verschieden Tasks und der Kommunikationsaufwand zum Zusammenführen der richtigen Zustände erhöht sich, da in diesem T4-Fall vier MPI-Tasks Daten untereinander austauschen müssen. Eine zusätzliche Differenzierung der Vorgänge ergibt sich, falls man

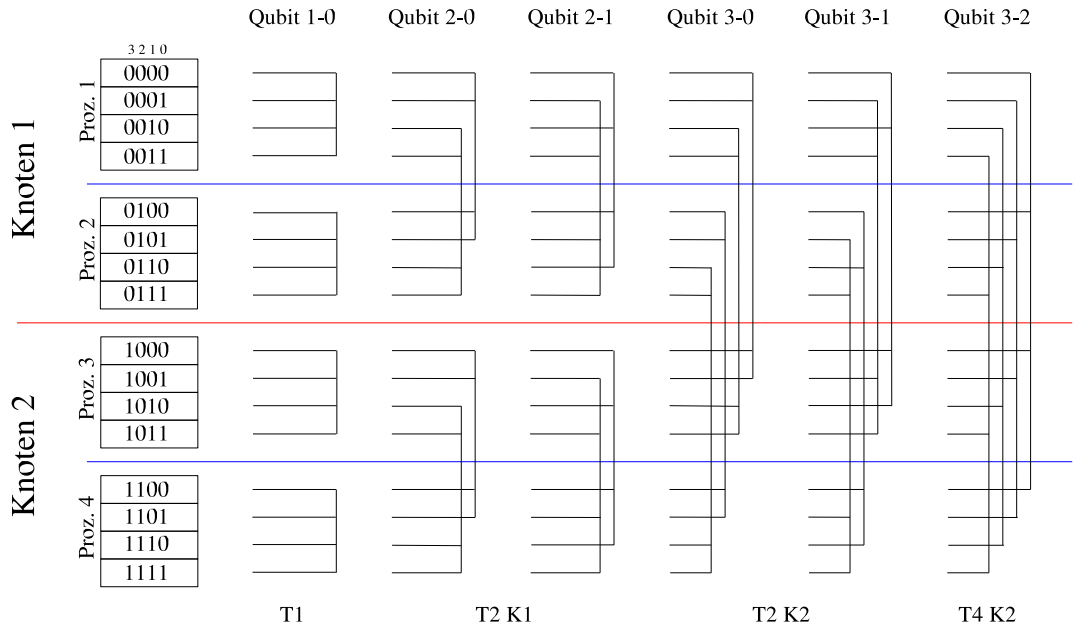

Abbildung 6.6: Beispiele von 2-Qubit Kommunikationstypen

bei den Fällen T2 und T4 unterscheidet, wieviele physikalischen Knoten beim Datentransfer beteiligt sind. Diese Unterteilung macht Sinn, da die Transfergeschwindigkeit für Daten zwischen Tasks auf einem Knoten ungleich höher ist, als zwischen Tasks auf verschiedenen Knoten, die durch das Verbindungsnetzwerk geschickt werden müssen. Diese Feinunterscheidungen werden mit T2K1, T2K2, T4K1, T4K2 und T4K4 bezeichnet. Unterschieden wird dabei durch die Bedingung, ob `i1v` bzw. `i1h` ≥ oder < 2^{36} ist.

6.7 Die Hadamard-Operation

Die Hadamard-Operation überführt den Zustand $|0\rangle$ nach $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ und den Zustand $|1\rangle$ nach $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Daher spielt sie eine zentrale Rolle bei der Herstellung eines

Superpositionszustandes, um die Eigenschaft des Quantenparallelismus ausnutzen zu können, so z.B. nach der Initialisierung des Systems in einem seiner Basiszustände.

In Matrixdarstellung sieht das Hadamard-Gatter folgendermaßen aus:

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{6.6}$$

Seine Wirkung auf den Zustand $a_0|0\rangle + a_1|1\rangle$ ist gegeben durch:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} a_0 + a_1 \\ a_0 - a_1 \end{pmatrix} \tag{6.7}$$

6.7.1 Kommunikationsschema

Alle Eintragspaare müssen bei dieser Operation jeweils miteinander verknüpft werden. Daher kommt es in keinem Fall vor, daß wie beim idealen CNOT-Gatter (Kapitel 6.8) manche MPI-Prozesse bei minimaler Kommunikation keine Rechenarbeit leisten müssen.

Beim Hadamard-Gatter kann neben dem Fall, daß keine Kommunikation nötig ist, nur der Fall T2 eintreten, d.h. der Abstand der zusammengehörigen Eintragspaare übertrifft die Anzahl der lokal abgespeicherten Einträge des Zustandsvektors. Insgesamt benötigt man dann ein Datentransferaufkommen von `spn` Einträgen, die vor und nach der Rechnung zwischen den 2 MPI-Tasks ausgetauscht werden müssen.

Wie in Abbildung 6.7 gezeigt, besitzt im T2-Fall jeder MPI-Task nur Einträge einer Position. Daher schickt Task 0 mit den Einträgen der Position 1 die 2. Hälfte seiner Einträge an Task 1. Task 1 schickt die 1. Hälfte seiner Einträge der Position 2 an Task 1. Die Paketgröße der Daten kann hierbei frei gewählt werden. Anschließend muß jeder Task `spn`/2 Matrixmultiplikationen ausführen, wonach der Kommunikationsprozess in umgekehrter Richtung wiederholt wird, um die zuvor verschickten Einträge nach der Hadamard-Operation zurück an ihren ursprünglichen Platz zu bringen.

6.7.2 Auffinden der Zustände und Rechenroutinen

Das Auffinden der richtigen Eintragspaare innerhalb eines MPI-Speichers erfolgt durch eine Doppelschleife. Die äußere erhöht in jedem Durchlauf den Wert der Variable `k` um `i1v` und setzt den Wert so auf den ersten Eintrag eines Musterblocks (Abbildung. 6.4). Die innere Schleife arbeitet nun alle Einträge der Position 1 innerhalb des Blocks ab und verknüpft diese mit um `i1h` entfernten Einträgen der Position 2.

```
c=sqrt(2)
do k=imin, imax , i1v
   do i=k,k+i1h-1
      j=i+i1h
      ...
   end do
end do
```

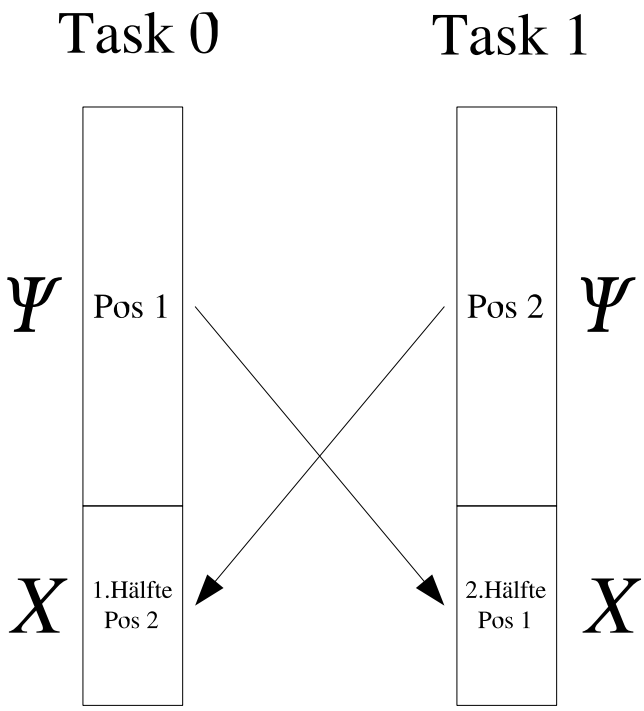

Abbildung 6.7: T2-Kommunikationsschema für 1-Qubit-Operationen

Bei dem Zugriff auf den Speicher muß dabei natürlich je nach Kommunikationsmuster unterschieden werden, ob die benötigten Zustände im regulären Speicher Ψ oder beim T2-Fall im Hilfsspeicher χ liegen. Diese Schleifen, die die eigentliche Rechnung ausführen, sind OpenMP parallelisiert und können so auf die einzelnen Prozessoren innerhalb eines MPI-Tasks aufgeteilt werden. Dies geschieht, indem die äußere Schleife gleichzeitig mit unterschiedlichen Werten für die Schleifenvariable `k` durchlaufen wird.

Diese Methode zum Auffinden der Eintragspaare hat sich als ungefähr 25% schneller gegenüber einer früheren Version erwiesen, die zwar mit einer einfachen Schleife auskommt, beim Speicherzugriff durch den alternierenden Term `i2` jedoch dazu gezwungen ist, teilweise große Sprünge im Speicher zu machen, um den nächsten Eintrag, den der Suchalgorithmus liefert, auslesen zu können.

```
c=sqrt(2)
do k=imin, imax , 2
   i2=iand(i,i1h)
   k=i-i2+i2/i1h
   l=k+i1h
   ...
end do
```

6.8 Die CNOT-Operation

Das kontrollierte NOT Gatter ist eine 2-Qubit-Operation, die den Zustand des Zielbits einer NOT-Operation unterzieht, falls das Kontrollbit sich im Zustand $|1\rangle$ befindet. In Matrix Schreibweise sieht das CNOT-Gatter folgendermaßen aus:

$$\mathbf{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{6.8}$$

Die Wirkung auf einen Zustand $a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle$ ist gegeben durch

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} a_{00} \\ a_{01} \\ a_{11} \\ a_{10} \end{pmatrix} \tag{6.9}$$

Dies bedeutet, daß die Amplituden der Basiszustände $|10\rangle$ und $|11\rangle$ bei der Operation vertauscht werden.

Für die Berechnung einer CNOT-Operation müssen dementsprechend alle Einträge des Zustandsvektors an der Position 3 (vgl. Kapitel 6.6) mit den Einträgen an Position 4 paarweise getauscht werden. Die Einträge an Position 1 und 2 bleiben unberührt. Im idealen Fall kann die CNOT-Operation also auf das paarweise Verknüpfen von jeweils 2 Einträgen des Zustandsvektors reduziert werden, anstatt alle vier zusammengehörigen Einträge einer 2-Qubit-Operation bearbeiten zu müssen. Dies spart Rechenzeit, da nur die Hälfte der 2^n Einträge bearbeitet werden müssen.

Die Kommunikation vereinfacht sich enorm durch die Tatsache, daß die Einträge an Position 1 und 2 nicht berücksichtigt werden müssen. Daher ist nur die relative Lage der Einträge an Position 3 und 4 von Bedeutung. Notwendigkeit zur Kommunikation besteht also nur im Fall eines 4-Task-Kommunikationsmusters (vgl. Kapitel 6.6), bei dem die Einträge der Positionen 3 und 4 bei verschiedenen MPI-Tasks liegen. Um zu gewährleisten, daß zusammengehörige Zustandspaare dieser Positionen im Speicher desselben MPI-Tasks vorliegen, müssen insgesamt $\frac{2^n}{4}$ Einträge des Zustandsvektors verschickt und nach der Rechnung wieder empfangen werden.

Dieser minimale Kommunikationsaufwand hat den großen Nachteil, daß im Falle eines T2- und T4-Musters die relevanten Zustände zur Berechnung der CNOT-Operation nur über die Hälfte der MPI-Tasks innerhalb eines Musterblocks verteilt ist. Die andere Hälfte der MPI-Tasks besitzt in ihrem Speicher nur Einträge der Positionen 1 und 2 und hat daher keine Möglichkeit, zur Berechnung des neuen Zustandes beizutragen. Die Rechenleistung gegenüber dem T1-Fall reduziert sich somit auf die Hälfte, da 50% der Prozessoren während der Operation nicht benutzt werden können.

Um diese offensichtlichen Leistungseinbußen zu umgehen, wurde für den T2- und T4-Fall eine Lastverteilung implementiert, die Einträge des Zustandsvektors so auf alle MPI-Tasks verteilt, daß alle Prozessoren in die Berechnung mit einbezogen werden

können. Darüber hinaus wurde in einem zweiten Schritt für den T2-Fall eine dynamische Lastverteilung programmiert. Die Lastverteilung ist spezifisch für die CNOT-Operation und verteilt die durchzuführenden Rechenoperationen auf alle MPI-Prozesse.

6.8.1 Kommunikationsschema

Nach der Übergabe der Indizes der zu verknüpfenden Quantenbits wird durch Vergleich des Seiten- und Mittelabstands des Musters entschieden, welches Kommunikationsmuster vorliegt. `i0v` und `i0h` enthalten dabei die Indizes von Kontroll- und Zielqubit, die Funktion `ibset` setzt das Bit an der Position `i0v` bzw. `i0h` auf 1 und berechnet so $2^{\texttt{i0v}}$ und $2^{\texttt{i0h}}$, also den Seiten- und Mittelabstand des Qubitmusters, der in `i1v` und `i1h` abgelegt wird:

```
i1v=ibset(0_8,i0v)
i1h=ibset(0_8,i0h)
if (spn>=2_8*i1v) then -> T1, keine Kommunikation
  else if (spn>i1h) then -> T2, 2-Task-Kommunikation
    else -> T4, 4-Task-Kommunikation
  end if
end if
```

Ist die Anzahl der Zustände pro MPI-Task (`spn`) größer oder gleich des doppelten Mittelabstands, dann befinden sich alle vier zusammengehörigen Einträge der 2-Qubit-Operation auf einem MPI-Task, der die Operation unabhängig von den anderen ausführen kann. Ist dies nicht der Fall und gleichzeitig `spn` jedoch größer als der Seitenabstand des Musters dann liegt der Fall T2 vor; ist auch der Seitenabstand größer als `spn`, liegt der T4-Fall vor.

Bei T2 liegen die zusammengehörigen Vierergruppen über 2 MPI-Tasks verteilt, bei T4 über 4 MPI-Tasks. Neben dem Wissen, um welche Art der Operation es sich handelt, muß berechnet werden, welche 2 bzw. 4 MPI-Tasks die Positionen 1-4 der zusammengehörigen Einträge beinhalten. Beim Fall T2 geschieht dies durch

```
pos=ibits(myrank*spn,i0v,1)
p1=ieor(myrank*spn,i1v)/spn
```

beim Fall T4 durch

```
pos=(2*ibits(myrank*spn,i0v,1)+ibits(myrank*spn,i0h,1))
p1=ieor(myrank*spn,i1h)/spn
p2=ieor(myrank*spn,i1v)/spn
p3=ieor(myrank*spn,i1v+i1h)/spn
```

`myrank*spn` stellt den absoluten Index des ersten Eintrags des MPI-Tasks dar. Durch die Funktion `ibits` wird das Bit an der Stelle `i0v` bzw. `i0h` extrahiert und gibt entweder 0 oder 1 zurück je nachdem, ob der Seiten- bzw. Mittelabstand des Musters gesetzt ist.

Die Variable `pos` beinhaltet also die Position des MPI-Tasks innerhalb des Kommunikationsmusters, im T2-Fall also 0 oder 1 und im T4- Fall 0, 1, 2 oder 3. Damit weiß jeder MPI-Task, welche Einträge er bzgl. der Positionen zusammengehöriger Einträge besitzt. `p1` bis `p3` enthalten den Rank der Prozessoren, die die übrigen Positionen der 2er- bzw. 4er-Gruppen beinhalten. Die Funktion `ieor` verknüpft dabei den absoluten Index des ersten Eintrags des lokalen Zustandsspeicher eines Tasks mit dem Seiten- und Mittelabstand bitweise durch ein exklusives Oder-Gatter. Dardurch enthalten `p1-p3` die Ränge der MPI-Tasks, dessen Indizes sich um einen Seiten- (`i1h`), einen Mittel- (`i1v`) oder um einen Seiten- und Mittelabstand (`i1v+i1h`) vom eigenen unterscheiden. Dies ermöglicht die Adressierung der MPI-Tasks innerhalb eines Musters.

Wie schon angedeutet, wird eine Lastverteilung angestrebt. Es müssen so Einträge der Positionen 3 und 4 separat zwischen den MPI-Tasks verschickt werden können. Wie in Abschnitt 6.6 gesehen, liegen die Einträge verschiedener Positionen beim T2-Fall nicht unbedingt hintereinander im Zustandsvektor, sondern in Blöcken verschiedener Größen und Abstände. Bei der Kommunikation mit MPI beeinflußt die Größe des zu verschickenden Datenpaketes die Geschwindigkeit des Kommunikationsvorgangs (s. Kapitel 5.3.2). Im Programm läßt sich daher die Größe der Pakete angeben, in die die zu verschickenden Daten unterteilt werden sollen, um die Dauer der MPI-Routinen zu minimieren. So stellt sich bei der Kommunikation im T2-Fall die Aufgabe, aus dem lokalen Zustandsvektor Einträge der Positionen 3 oder 4 zu extrahieren und diese in Paketen von festgelegter Größe zu versenden, unabhängig von der Blockgröße oder dem Blockabstand des Qubitmusters. Um dies zu erreichen, wird unterschieden, ob die Paketgröße `paksz` größer oder

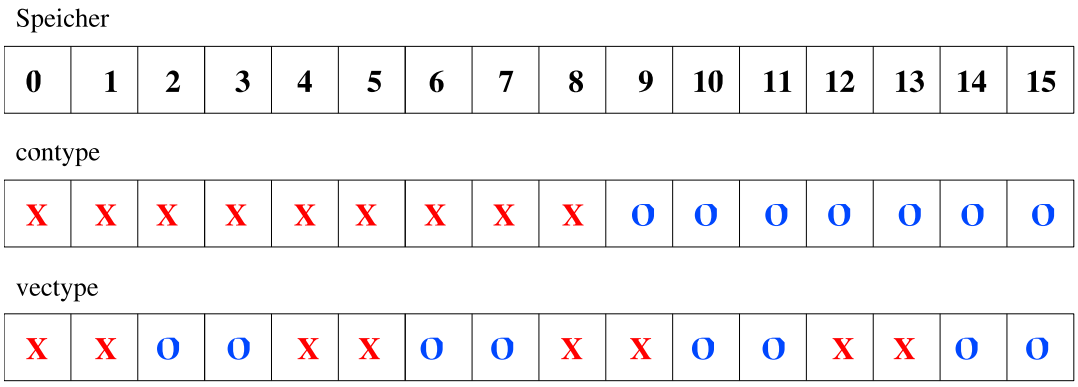

Abbildung 6.8: Die Datentypen contype und vectype nehmen an den mit X markierten Stellen Speichereinträge auf, die mit O markierten Stellen bleiben leer.

kleiner gleich der Blockgröße `blsz` ist. Ist `paksz` $\leq$ `blsz`, dann liegen alle Einträge einer Position in dem Paket hintereinander und können durch einen MPI-Sendeaufruf an einem Stück aus dem Zustandsvektor herauskopiert werden, indem ein Variablentyp `contype` definiert wird, der `blsz` Einträge zusammenfasse. Gilt `paksz` $>$ `blsz`, dann wird ein Variablentyp `vectype` definiert, der an den Stellen Löcher aufweist, an denen der Zustandsvektor unerwünschte Einträge enthält, so daß das Herauskopieren einer Va-

riablen dieses Typs aus dem Zustandsvektor und das gleichzeitige Empfangen der Daten im Typ `contype` das gewünschte Ergebnis bringt (s. Abbildung 6.8). Alle Variablen, die sich auf Blöcke beziehen, hängen nur von der Form des Qubitmusters ab. Variablen, die auf der Paketgröße basieren, enthalten Informationen für die MPI-Aufrufe.

```
blsz=i1h
blno=spn4/blsz
blds=2_8*blsz

if (paksz>blsz) then
   paksperblock=1_8
   pakno=spn4/paksz
   blocksperpak=paksz/blsz
   pakds=2_8*paksz
   call MPI_TYPE_VECTOR(blocksperpak,blsz,blds,vectype)
   call MPI_TYPE_COMMIT(vectype)
else
   paksperblock=blsz/paksz
   pakno=spn4/blsz
   pakds=2_8*blsz
   vectype=contype
end if
```

6.8.2 Statische Lastverteilung der CNOT-Operation

Das Konzept der statischen Lastverteilung sieht vor, die zu berechnenden Zustände von den MPI-Tasks mit den Einträgen der Positionen 3 und 4 auf die MPI-Tasks mit den Einträgen der Positionen 1 und 2 so zu verteilen, daß jeder Tasks den gleichen Rechenaufwand hat. Das bedeutet jeder Task, ob T2- oder T4-Fall, muß $2 \times \frac{1}{4}$`spn` Einträge des Zustandsvektors miteinander verknüpfen.

Je nach Lage der Einträge des Zustandsvektors auf den Tasks wird an Hand des Wertes der Variable `pos` entschieden, welche Aufgabe der jeweilige Task innerhalb der Lastverteilung ausführt. Im T2-Fall besitzt Task 0 die Einträge der Positionen 1 und 2 und dient als Auslagerungsprozess, Task 1 besitzt die Einträge der Positionen 3 und 4. Im T4-Fall besitzt jeder der vier zusammengehörigen Tasks nur Einträge einer Position, daraus folgt, daß Task 0 und 1 mit den Einträgen der Positionen 1 und 2 als reiner Auslagerungstask dienen, wohingegen Task 2 und 3 mit den Einträgen der Positionen 3 und 4 die Daten verteilen müssen.

Der Ablauf des Datentransfers sieht folgendermaßen aus: Beim T2-Fall schickt Task 1 die erste Hälfte seiner Einträge der Positionen 3 und 4, insgesamt `spn`/2, an Task 0, der diese im lokalen Hilfsspeicher χ zwischenlagert. Nun folgt die Rechenroutine, bei der Task 0 nur auf dem Hilfsspeicher operiert und Task 1 die verbleibende Hälfte der Positionen 3 und 4 miteinander verknüpft. Anschließend erfolgt der Datentransfer in umgekehrter Richtung. Das Kommunikationsschema für den T2-Fall ist in Abbildung

6.8.2 dargestellt. Im Fall T4 ist das Kommunikationsaufkommen höher. Um jedem Task

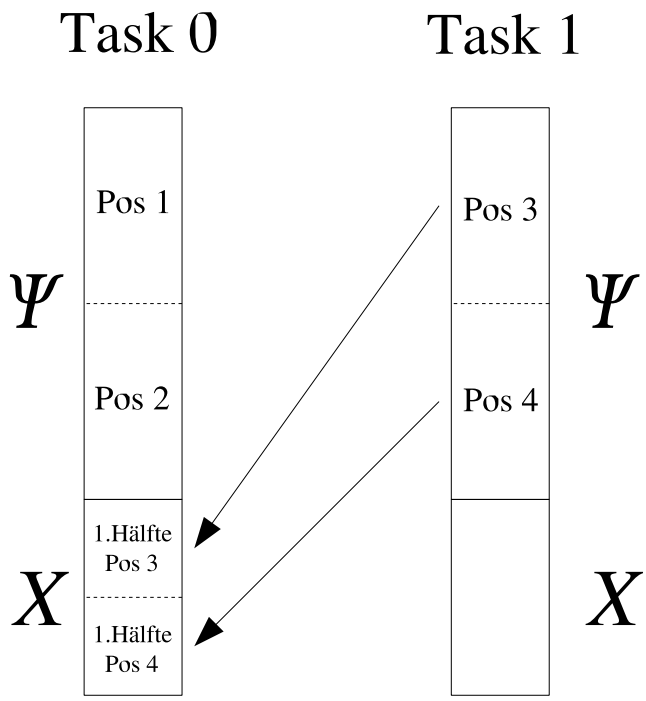

Abbildung 6.9: T2-Kommunikationsschema für 2-Qubit-Operationen

die gleiche Anzahl von Rechenoperationen zu ermöglichen, teilen Task 2 und 3 ihre Zustände in 4 gleich große Teile auf. Beide Prozesse senden nun das erste Viertel ihrer Daten an Task 0, das zweite Viertel an Task 1, die dort wieder im Hilfsspeicher χ abgelegt werden. Desweiteren erhält Task 2 von Task 3 das dritte Viertel und umgekehrt Task 3 von Task 2 das vierte Viertel seines Zustandsvektors.

Auf diese Weise hat nun jeder Task jeweils `spn`/4 Einträge der Position 3 und 4, auf die nun die Operation angewandt werden kann. Danach werden die berechneten Daten auf demselben Weg in umgekehrter Richtung zurückgeschickt. Abbildung 6.8.2 verdeutlicht diesen Vorgang. Die durch die Halbierung des pro MPI-Task vorhandenen Rechenaufwands gewonnene Zeit gleicht auch die durch die Lastverteilung erst nötig gewordene Kommunikation beim T2-Fall und den Anstieg des Kommunikationsaufkommens beim T4-Fall von `spn` auf $\frac{3}{2}$`spn` aus.

6.8.3 Dynamische Lastverteilung der CNOT-Operation

Bei der dynamischen Lastverteilung steht im Voraus nicht fest, wieviele Pakete der Größe `paksz` ausgelagert werden und wieviele Pakete ein Task selber berechnet. Implementiert wurde die dynamische Lastverteilung nur für den Fall von 2-Task-Kommunikation. Bei der 4-Task-Kommunikation war dies auf Grund der beschränkten Zeit nicht möglich. Wie im statischen Fall handelt es sich hier um zwei miteinander kommunizierende Tasks, wo-

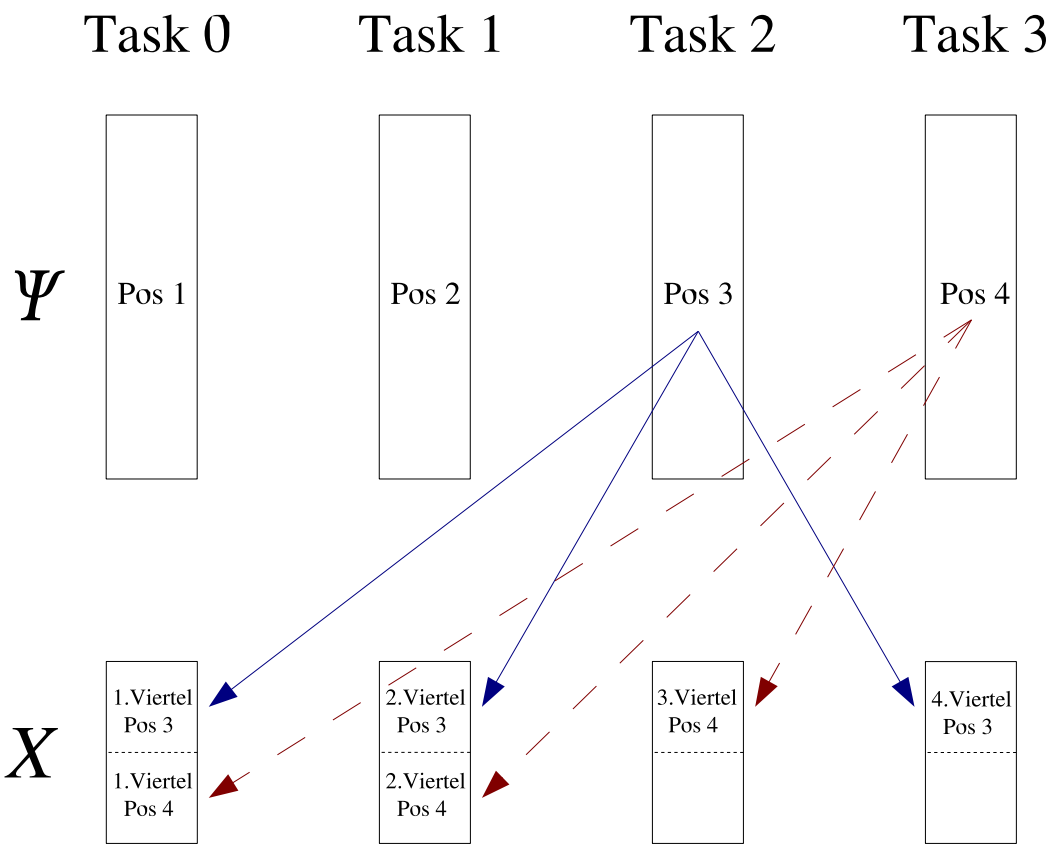

Abbildung 6.10: T4-Kommunikationsschema für 2-Qubit-Operationen

bei Task 0 den Auslagerungstask darstellt, in dessen Speicher sich nur Einträge des Zustandsvektors der Positionen 1 und 2 befinden. Task 1 besitzt alle Einträge der Positionen 3 und 4. Abbildung 6.11 verdeutlicht das Ablaufschema für Task 1 der MPI-Tasks. Zu Anfang wird an Hand der Paketgröße die Anzahl der Pakete festgelegt, die insgesamt zu berechnen sind (`pakall`) und die Anzahl auszulagernder Datenpakete (`paktbs`) festgelegt. Nun beginnt eine Schleife, die erst beendet wird, wenn alle Pakete berechnet sind. Es wird sichergestellt, daß die maximale Anzahl von gleichzeitig im Sendeprozess befindlichen Paketen (`pakmax`) durch das Senden von `paktbs` Paketen nicht überschritten wird, und `paktbs` evtl. verringert. Gibt es dann Pakete zu senden oder ausgelagerte Pakete zu empfangen (`paksndpend` > 0), so werden die entsprechenden Sende- bzw. Empfangsroutinen initialisiert und die Variablen mit der Anzahl der sich im Sende- bzw. Empfangsprozess befindlichen Pakete `paksndpend` und `pakrecvpend` aktualisiert. Ist dies geschehen, wird ein Datenpaket berechnet. Durch die anschließenden Abfragen, ob initialisierte Sende- oder Empfangsroutinen abgeschlossen sind, werden `paksndpend` und `pakrecvpend` evtl. wieder erniedrigt, um im nächsten Schleifendurchlauf wieder mit `pakmax` verglichen werden zu können. Im letzten Schritt wird nun aus dem Quotient der fertig berechneten ausgelagerten Pakete (`pakrecv`) und den selbstberechneten

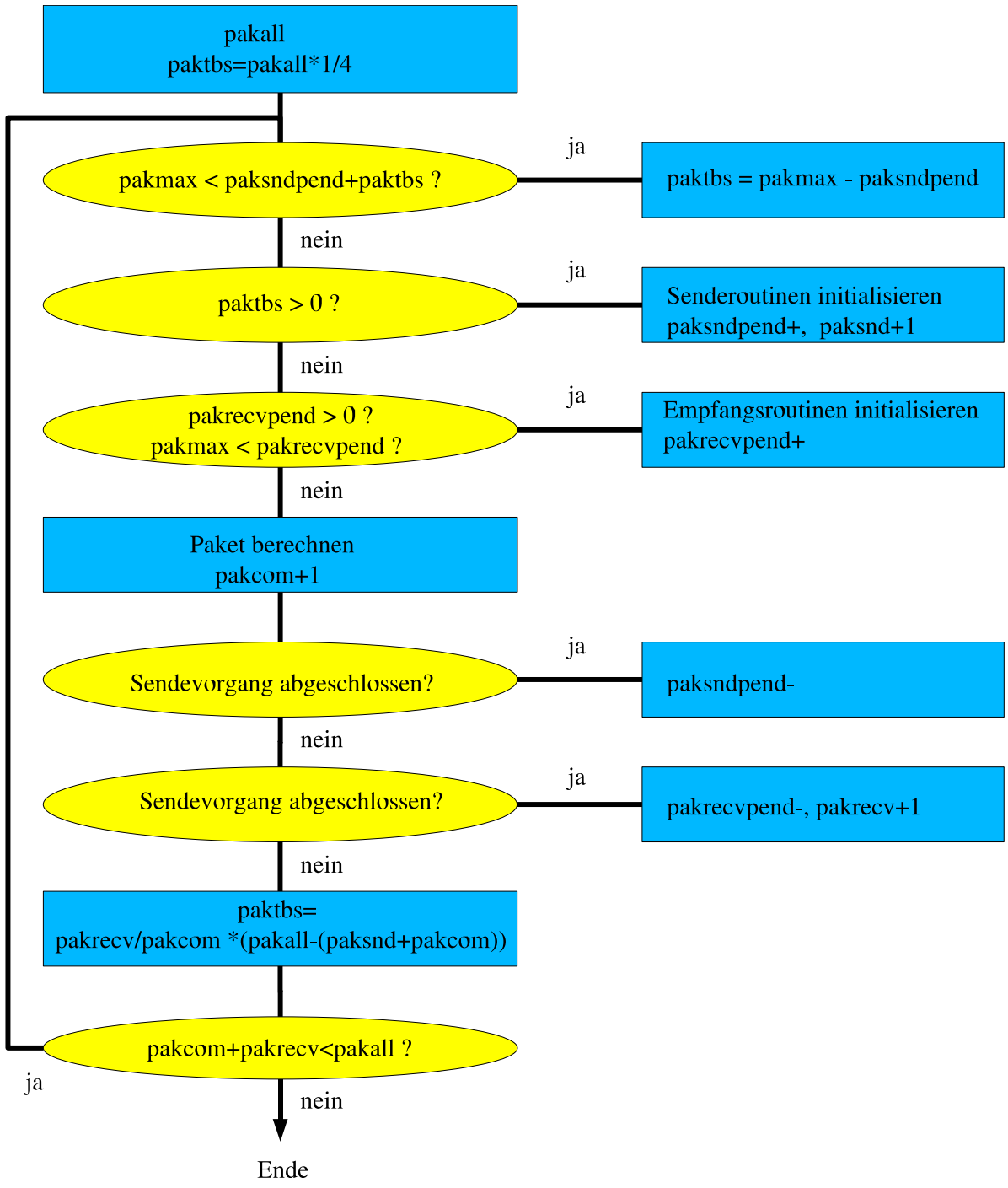

Abbildung 6.11: Schema der dynamischen Lastverteilung

Paketen (`pakcom`) der Bruchteil der verbleibenden Pakete berechnet, die zu verschicken nötig wäre, damit beide Tasks die Operation möglichst gleichzeitig beenden. Damit kann `paktbs` aktualisiert werden und solange nicht alle Pakete berechnet wurden, beginnt die

Schleife von vorne.

Die Aufgabe von Task 0 besteht darin, ständig zu prüfen, ob Pakete von Task 1 gesendet werden. Liegen Pakete zum Empfangen vor, so werden entsprechende Empfangsroutinen gestartet, alle schon empfangenen Pakete werden berechnet und alle schon berechneten Pakete an Task 1 zurückgeschickt. Task 0 legt dabei die empfangenen Pakete der Reihenfolge nach im Hilfsspeicher `chi` ab. Sind 50% aller Pakete empfangen, ist der Speicher voll. Ein weiteres Paket kann erst wieder empfangen werden, wenn das Paket an erster Speicherposition schon fertig bearbeitet und zurückgesendet worden ist.

Es werden nicht-blockierende Sende- und Empfangsbefehle benutzt. Diese haben den Vorteil, daß nach dem Aufruf der Routinen der Vorgang initialisiert, dann jedoch weiter das Programm abgearbeitet wird, während das Empfangen bzw. Senden im Hintergrund abläuft. Dabei muß natürlich sichergestellt werden, daß die Kommunikation abgeschlossen wird, bevor auf die Daten zugegriffen werden kann.

6.8.4 Rechenroutinen

Bei den Rechenroutinen kommt OpenMP zum Tragen, das hier zum Parallelisieren der Rechenschleifen eingesetzt wird, was auch problemlos möglich ist. Allerdings ist zu beachten, daß bei den verschachtelten Schleifen die Anzahl der Durchläufe der einzelnen Schleifen wieder von dem Qubitmuster abhängt. Dies kann soweit gehen, daß eine Schleife nur ein einziges Mal aufgerufen wird und die Parallelisierung dieser Schleife, die durch Aufteilung der Schleifenvariable erfolgt, keinen Sinn macht. Daher wird vorher festgestellt, welche Schleife die meisten Durchläufe zu bewältigen hat, und diese dann auf die verschiedenen OpenMP-Prozesse verteilt.

Bei den Routinen, die die eigentliche Rechenarbeit leisten, geht es vor allem darum, eine möglichst effiziente Methode zu finden, nacheinander auf die über den Speicher verteilten Einträge der Positionen 3 und 4 zuzugreifen. Wird keine Kommunikation zur Ausführung einer Operation benötigt, dann liegen Einträge aller Positionen und somit vollständige Muster (s. Kapitel 6.6) im Speicher. In einer dreifach geschachtelten Schleife werden nun die Muster, innerhalb dieser die Blöcke und innerhalb der Blöcke wiederum die einzelnen Einträge durchgegangen und die Einträge an den Positionen `j3` und `j4` verknüpft:

```
do s1=0,mno
   x=s1*msz+i1v
   do s2=0,blno
      y=s2*blds+x
      do s3=0,blsz
        j3=y+s3
        j4=j3+i1h
           ...
      end do
   end do
end do
```

Diese Methode ist um ca. 20% schneller als eine für 2-Qubit-Routinen modifizierte Schleife, deren 1-Qubit-Form in Kapitel 6.7.2 ihre Anwendung findet:

```
do k=0,spn-1,4
   i2m=iand(k,i1v)
   i2s=iand(k,i1h)
   j3=k-i2m-i2s+ex*i2m/i1v+i2s/i1h+i1v
   j4=j3+i1h
      ...
end do
```

In den Fällen, in denen Kommunikation nötig ist, muß unterschieden werden zwischen den Auslagerungsprozessen, bei denen die Daten im Hilfsspeicher durch das Verschicken und Empfangen mit `contype` und `vectype` schon in den beiden Hilfsspeichern χ hintereinanderliegen und so mit einer einfachen Schleife der Art

```
do k=0,spn4-1 (0,paksz-1)
   j3=k
   j4=k
      ...
enddo
```

verknüpft werden können, und den Prozessen, die auf dem lokalen Zustandsspeicher `psi` operieren müssen. Im T4-Fall erübrigt sich diese Unterscheidung, da hier die Tasks 2 und 3 nur Einträge einer Position enthalten und diese so schon hintereinander liegen. Im T2-Fall liegt auf Task 1 die untere Hälfte eines Musters, das sich in diesem Fall über 2 Tasks erstreckt. Die äußere Schleife aus dem T1-Fall kann daher entfallen und die Rechenroutine sieht folgendermaßen aus:

```
do s1=0,blno-1
   x=s1*blds+off
   do s2=0,blsz-1
      j3=s2+x
      j4=j3+i1h
         ...
   end do
end do
```

Die Rechenroutine der dynamischen Lastverteilung unterscheidet sich nur wenig von der im statischen Fall. Einziger Unterschied ist hier, daß die Schleifenvariablen `k` bzw. `s1` und `s2` nur jeweils von 0 bis `paksz` laufen und die Anfangswerte im Speicher entsprechend aktualisiert werden.

7 Laufzeitmessungen

Laufzeitmessungen einzelner Gatter wurden exemplarisch an der Hadamard- und der CNOT-Operation durchgeführt. Das Hadamard-Gatter diente dabei als Vertreter der 1-Qubit-Operationen, das CNOT-Gatter stellte einen Vertreter der 2-Qubit-Operationen dar. Diese Unterscheidung in 1- und 2-Qubit-Operationen ist sinnvoll, um die unterschiedliche Implementierung in Bezug auf Speicherverwaltung und Rechenaufkommen zu berücksichtigen.

Dabei wurden für beide Gatter unterschiedliche Systemgrößen und verschiedenen MPI-/OpenMP-Konfigurationen betrachtet. Die untersuchten Systemgrößen variierten dabei zwischen 32 und 36 Qubits, was einer Größe des Zustandsvektors von 64 GB bzw. 1 TB entspricht. 1 bis 16 Knoten des JUMP-Rechners sind nötig, um ein System entsprechender Größe zu simulieren (s. Tabelle 7.1). Mehr als 16 Knoten des JUMPs zu benutzen, ist im Normalbetrieb nicht möglich, so daß 36 Qubits das größtmögliche zu simulierende System sind (s. Kapitel 6.3). Die verschiedenen Konfigurationen beziehen

Anzahl der Qubits	Größe des Zustandsvektor	Anzahl der JUMP-Knoten
32	64 GB	1
33	128 GB	2
34	256 GB	4
35	512 GB	8
36	1024 GB	16

Abbildung 7.1: Systemgrößen und benötigter Speicher

sich auf die Aufteilung der 32 Prozessoren auf einem Rechnerknoten in p MPI-Tasks mit jeweils t OpenMP-Threads. Möglich sind Konfigurationen, für die gilt $p \cdot t = 32$. Auf Grund der Rechnerarchitektur des JUMPs (s. Kapitel 5.2) sind Konfigurationen mit $t = 16$ und $t = 32$ deutlich langsamer als reines MPI ($t = 1$), da jeweils 8 Prozessoren auf einer Prozessorbank liegen und der Speicherzugriff auf den Speicher einer anderen Prozessorbank langsamer ist als der Speicherzugriff innerhalb der eigenen. Bei den 1-Qubit-Operationen betrug der Geschwindigkeitsverlust für $t = 16$ und $t = 32$ zwischen 300 und 600% gegenüber dem Fall $t = 1$. Tabelle 7.2 verdeutlicht den Zusammenhang zwischen Konfiguration und Anzahl der lokal berechenbaren Qubits, sowie der Größe des lokalen Teils des Zustandsvektors eines Tasks. Durch eine enge Kontingentierung der verfügbaren Rechenzeit auf dem JUMP für den Nutzer wurden nicht alle möglichen

MPI-Tasks p	OpenMP-Threads t	Index des höchsten lokal manipulierbaren Qubits	Größe des lokalen Zustandsvektors
32	1	26	2 GB $\widehat{=}$ 2^{27} Zust.
16	2	27	4 GB $\widehat{=}$ 2^{28} Zust.
8	4	28	8 GB $\widehat{=}$ 2^{29} Zust.
4	8	29	16 GB $\widehat{=}$ 2^{30} Zust.
2	16	30	32 GB $\widehat{=}$ 2^{31} Zust.
1	32	31	64 GB $\widehat{=}$ 2^{32} Zust.

Abbildung 7.2: Verschiedene Konfigurationen und Größe des lokalen Zustandsvektors

Konfigurationen gemessen, vor allem bei den 2-Qubit-Operationen wurden nur die Konfigurationen $t = 1 \ldots 4$ getestet, da höhere Werte für t eine längere Laufzeit erwarten ließen.

7.1 Laufzeiten der Hadamard-Operation

Zur Laufzeitbestimmung des Hadamard-Gatters wurde die Zeit für die Anwendung einer Hadamard-Operation auf alle Qubits des Systems gemessen und anschließend durch die Anzahl der Operationen geteilt. Durch Ausführen der Operation auf allen Qubits wird sichergestellt, daß jedes Kombinationsmuster von Einträgen des Zustandsvektors genau einmal benötigt wird. So wird eine Aussage über die durchschnittliche Operationsdauer bei gegebener Systemgröße möglich. Abbildung 7.3 zeigt die Dauer der einzelnen Hadamard-Operationen an einem System mit 36 Qubits bei unterschiedlichen Konfigurationen. Deutlich erkennt man den Anstieg der Laufzeiten für die Qubits mit $q > 26$. Dieser entsteht durch die benötigte Kommunikation und setzt für die unterschiedlichen Konfigurationen bei $q = 27 \ldots 31$ ein. Während bei einer Operation auf den Qubits 27-31 je nach Anzahl der Threads pro MPI-Tasks Datentransfer innerhalb eines Knotens stattfindet, ist bei Operationen auf die Qubits $q > 32$ Kommunikation zwischen den Knoten notwendig. Dies steigert die Kommunikationszeit und damit auch die Rechenzeit je nach Konfiguration um bis zu 100%, da der Datentransfer zwischen den Knoten über das Verbindungsnetzwerk laufen muß, während die Kommunikation innerhalb eines Knotens auf einen Zugriff auf den gemeinsamen Speicher abgebildet wird.

Während die Operationen auf den Qubits $q \leq 27$ bei allen Konfigurationen die gleiche Zeit benötigen, zeigt sich bei einsetzender Kommunikation, daß der Fall $t = 8$ deutlich langsamer ist, was durch die oben beschriebene architektonische Zusammenfassung von 8 Prozessoren auf einer Bank erklärbar ist.

Das Absinken der Laufzeiten bei den Operationen auf den Qubits $0 < q < 11$ kann darauf zurückgeführt werden, daß die zusammengehörigen Einträge des Zustandsvektors in diesen Fällen nahe genug beieinander liegen, um bei einem Speicherzugriff gleichzeitig in den Cache geladen zu werden, so daß ein weiterer Speicherzugriff erspart bleibt. Der Fall $q = 0$ stellt einen programmiertechnischen Sonderfall dar, bei dem der Zu-

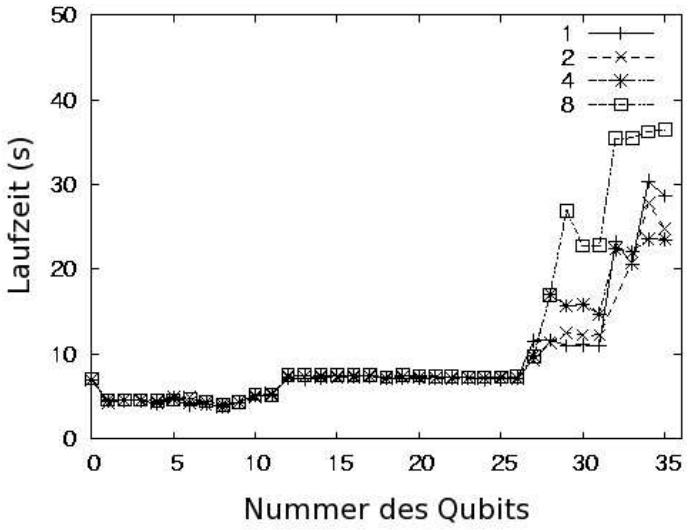

Abbildung 7.3: Laufzeiten einzelner Hadamard-Operationen auf unterschiedlichen Qubits bei einer Systemgröße von 36 Qubits in Abhängigkeit der Threadanzahl pro MPI-Task

griff auf die Vektoreinträge nicht durch sogenanntes *Even-/Odd-Splitting* erfolgen kann, sondern auf Grund des Zugriffes auf benachbahrte Speichereinträge individuell programmiert werden mußte, was eine höhere Laufzeit zur Folge hat. Das *Even-/Odd-Splitting* Verfahren benutzt zwei Speicherfelder, um Einträge des Zustandsvektors mit geraden und ungeraden Indizes getrennt abzulegen. Dies führt außer im Fall $q = 0$ zu einem Geschwindigkeitsgewinn, da die Indizes der zu verknüpfenden Vektoreinträge bei allen Operationen entweder gerade oder ungerade sind.

Abbildung 7.4 stellt die Laufzeiten der Hadamard-Operation bei verschiedenen Systemgrößen und Konfigurationen gegenüber. Es zeigt sich, daß bei gleicher Konfiguration die durchschnittliche Operationsdauer mit der Größe des Systems anwächst, da ein Vergrößern des Systems über die Zahl der innerhalb eines MPI-Tasks berechenbaren Qubits hinaus den Anteil der Operationen erhöht, für die Datenaustausch zwischen den einzelnen Tasks nötig ist und somit die durchschnittliche Operationsdauer erhöht. Deutlich zu sehen ist der starke Zeitanstieg bei den Messungen für $t = 16$, der auf Grund der Hardware-Architektur zustande kommt. Zwischen den Konfigurationen bestehend aus 8, 4, 2 und 1 Thread pro Task liegt eine weit geringere Zeitdifferenz, wobei auch hier eine Konfiguration mit maximaler Taskanzahl die geringste Laufzeit innerhalb einer Meßfehlertoleranz von 5% ergab.

Wie schon angedeutet, sollten sich bei einer kontinuierlichen Vergrößerung des Systems die Laufzeiten gleicher Konfigurationen asymptotisch einem Grenzwert nähern, der den Messungen mit Kommunikation entspricht. Dies ist jedoch nur bei Systemgrößen zu erwarten, für die $q_{max} >> 32$ gilt.

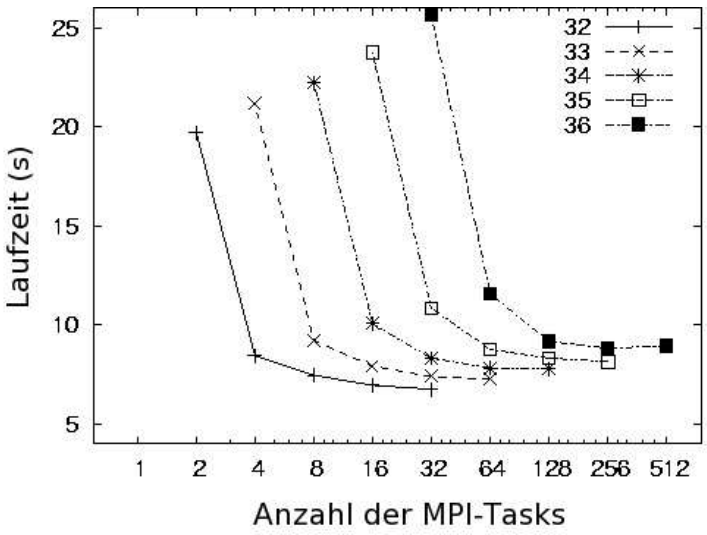

Abbildung 7.4: Laufzeiten der Hadamard-Operation bei verschiedenen Systemgrößen und Konfigurationen

7.2 Definition der Kommunikationsarten bei 2-Qubit-Operationen

Zur Diskussion der Ergebnisse des CNOT-Gatters als Vertreter der 2-Qubit-Operationen ist es sinnvoll, die verschiedenen Arten von Kommunikation zu betrachten, die sich durch die Wahl der Qubits ergeben, auf die das CNOT-Gatter angewendet werden soll.

Alle möglichen Kombinationsmöglichkeiten der Qubits lassen sich in sechs verschiedene Kommunikationsbereiche einteilen (s. Abbildung 7.5). Auf der y-Achse ist der Index des Kontrollqubits, beginnend mit 1, aufgetragen, auf der x-Achse der Index des Zielqubits, beginnend mit 0. Die Dreiecksform ergibt sich aus der Bedingung, daß der Index des Kontrollqubits größer sein muß, als der des Zielqubits ($y \leq x - 1$, s. Seite 40). Dies bedeutet keine Einschränkung der Funktionalität, da bei einem CNOT-Gatter Kontroll- und Zielqubit durch zusätzliches Anwenden von Hadamard-Operationen auf beiden Qubits vor und nach dem CNOT-Gatter vertauscht werden können (s. Abbildung 4.3). Exemplarisch dargestellt ist hier die Unterteilung in die einzelnen Bereiche für ein System mit 36 Qubits, so daß sich die Indexkombinationen von 1-0 bis 35-34 ergeben können, und einer Konfiguration mit 32 MPI-Tasks pro Knoten. Unterschiede zu anderen Systemen werden im Weiteren veranschaulicht. Die einzelnen Bereiche gliedern sich wie folgt: In Bereich I fallen alle Operationen, die zu ihrer Ausführung keine Kommunikation benötigen. Die Laufzeiten ergeben sich aus der reinen Rechenzeit, die für die Verknüpfung der entsprechenden Einträge des lokalen Zustandsvektors miteinander benötigt wird, und stellen daher auch ein nicht zu unterbietendes Minimum für kommunikationsbehaftete Operationen dar. Tabelle 7.2 gibt je nach Konfiguration den maximalen Index des Kontrollbits einer Operation an, die ohne Kommunikation

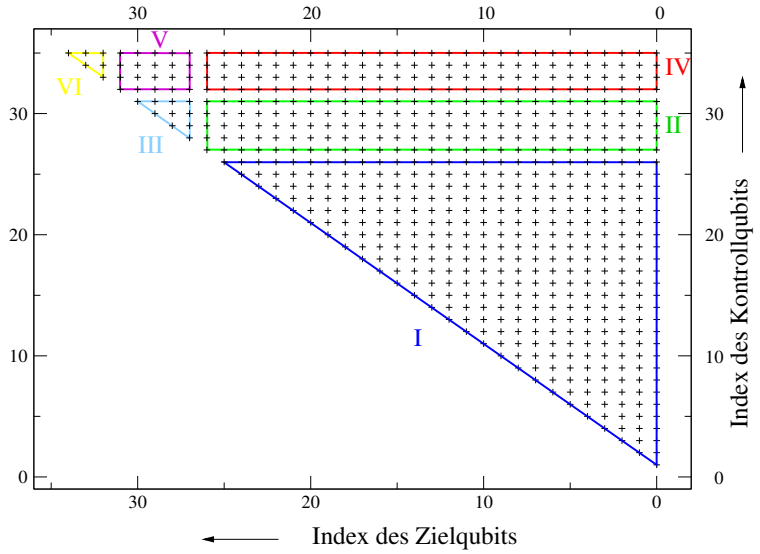

Abbildung 7.5: Muster zur Verdeutlichung der Kommunikationsarten bei einer Konfiguration mit 32 MPI-Tasks pro Knoten

durchführbar ist. Dieser maximale Index berechnet sich zu $31 - log_2 p$.

Für alle Indizes größer als $31 - log_2 p$ gilt, daß zusammengehörige Einträge des Zustandsvektors weiter auseinanderliegen, als der lokale Zustandsvektor Einträge hat. Übersteigt der Index des Kontrollbits diese Grenze, dann liegt eine 2-Task-Kommunikation (T2) vor (Bereiche II und IV), denn die Einträge der Positionen 1 und 3 liegen mehr als einen Task voneinander entfernt. Übersteigt auch der Index des Zielqubits den Wert $31 - log_2 p$, dann spricht man von einer 4-Task-Kommunikation (T4) (Bereiche III, V und VI), da nun auch der Abstand zwischen den Einträgen der Positionen 1 und 2 sowie 3 und 4 mehr als `spn` beträgt und alle Positionen auf verschiedenen Tasks liegen.

Mit einem Erhöhen der Threadanzahl pro MPI-Tasks wächst der Bereich I an, bis die physikalische Knotengrenze erreicht ist und alle Operationen bis zu den Indizes 31-30 innerhalb eines Tasks liegen. Die Bereiche II und III entsprechen daher dem T2 und T4 Kommunikationsfall innerhalb eines physikalischen Knotens und verkleinern sich so auch mit Erhöhen der MPI-Prozesse auf einem Knoten.

Die Bereiche IV, V und VI markieren die Kommunikation zwischen mehreren Knoten. Ihre Höhe in Einheiten des Kontrollqubits wird durch die Größe des Systems bestimmt. Bei nur 32 Qubits sind diese Bereiche nicht existent. Bereich IV enthält T2-Operationen zwischen 2 Knoten, Bereich V enthält T4-Operationen zwischen 2 Knoten. Ab einer Systemgröße von 34 Qubits und 4 Knoten gibt es Operationen, die dem Bereich VI zugeordnet werden. Hier überschreitet der Index des Zielqubits die physikalische Knotengrenze von 31, so daß Operationen in diesem Bereich der T4-Kommunikation entsprechen, bei der alle Positionen der Einträge auf verschiedenen Knoten liegen.

Unter T2-Kommunikation fallen die Bereiche II und IV, T4-Kommunikation beinhaltet die Bereiche III, V und VI. Zur Darstellung von Gesamtlaufzeiten wurde der

Mittelwert der Zeitdauern aller Operationen gebildet. Für einzelne Teilbereiche wurde der Mittelwert der entsprechenden Bereiche I bis VI gebildet. Daneben lassen sich die Operationen der Bereiche II und III durch Kommunikation innerhalb eines physikalischen Knotens charakterisieren, die Bereiche IV-VI durch Kommunikation zwischen verschiedenen Knoten.

Um miteinander vergleichbare Werte zu erhalten, wurden bei einer Messung alle möglichen Kombinationen der Indizes durchlaufen. Systeme gleicher Größe und somit gleicher Anzahl möglicher Qubitkombinationen unterscheiden sich nur durch ihre Konfiguration und somit durch die relative Anzahl der Operationen verschiedener Kommunikationsarten. Systeme gleicher Konfiguration unterscheiden sich durch die Anzahl der möglichen Operationen, größere Systeme haben dabei einen größeren Anteil an kommunikationsbehafteten Operationen.

Eine einzige Messung aller 630 Qubitkombinationen eines CNOT-Gatters auf einem System mit 36 Qubits unter Verwendung von 16 Knoten bzw. 512 Prozessoren dauert im *schnellsten* Fall ca. 1,25 h. Die Kosten dafür belaufen sich auf $1,25 \cdot 512 = 640$ Prozessorstunden *à* 2,10 €, also auf insgesamt ca. 1300 €. Daher konnten nur ausgewählte Messungen durchgeführt werden.

7.3 Statische Lastverteilung

Wie in Kapitel 6.8.2 beschrieben, basiert die Methode der statischen Lastverteilung auf der gleichmäßigen Aufteilung aller zu verknüpfenden Einträge des Zustandsvektors auf alle MPI-Tasks. Durch diese starre Aufteilung kommt es jedoch oft zu großen Laufzeitunterschieden zwischen den einzelnen Prozessen, die hauptsächlich dadurch bedingt sind, daß die Prozesse mit den relevanten Zuständen und die Auslagerungsprozesse stark unterschiedliche Anforderungen in Bezug auf Kommunikations- und Rechenaufkommen bewältigen müssen. Dies schlägt sich bei den Meßreihen in der relativ großen Standardabweichung nieder.

7.3.1 Verschiedene Systemkonfigurationen und Kommunikationsarten

Die Graphen in Abbildung 7.6 zeigen die durchschnittlichen Laufzeiten der verschiedenen Kommunikationsarten für verschiedene Systemgrößen und verschiedenen Systemkonfigurationen. Bei den Systemen mit 32, 33 und 34 Qubits wurden Messungen mit 1, 2 und 4 Threads pro MPI-Task gemacht, bei den größeren Systemen mit 35 und 36 Qubits wurden die Messungen auf Konfigurationen mit einem und zwei Threads pro Task beschränkt, da die Vermutung nahe lag, daß auch hier die Zeiten deutlich über denen der Fälle $t = 1$ und $t = 2$ liegen würden. Weiterhin sieht man die stark unterschiedlichen Laufzeiten für die Fälle T1 (keine Kommunikation), T2 und T4. Man erkennt bei allen Graphen deutlich, daß eine Konfiguration mit einem MPI-Task und 32 Threads pro Knoten die besten Ergebnisse liefert. Etwas schlechter schneidet eine Konfiguration mit

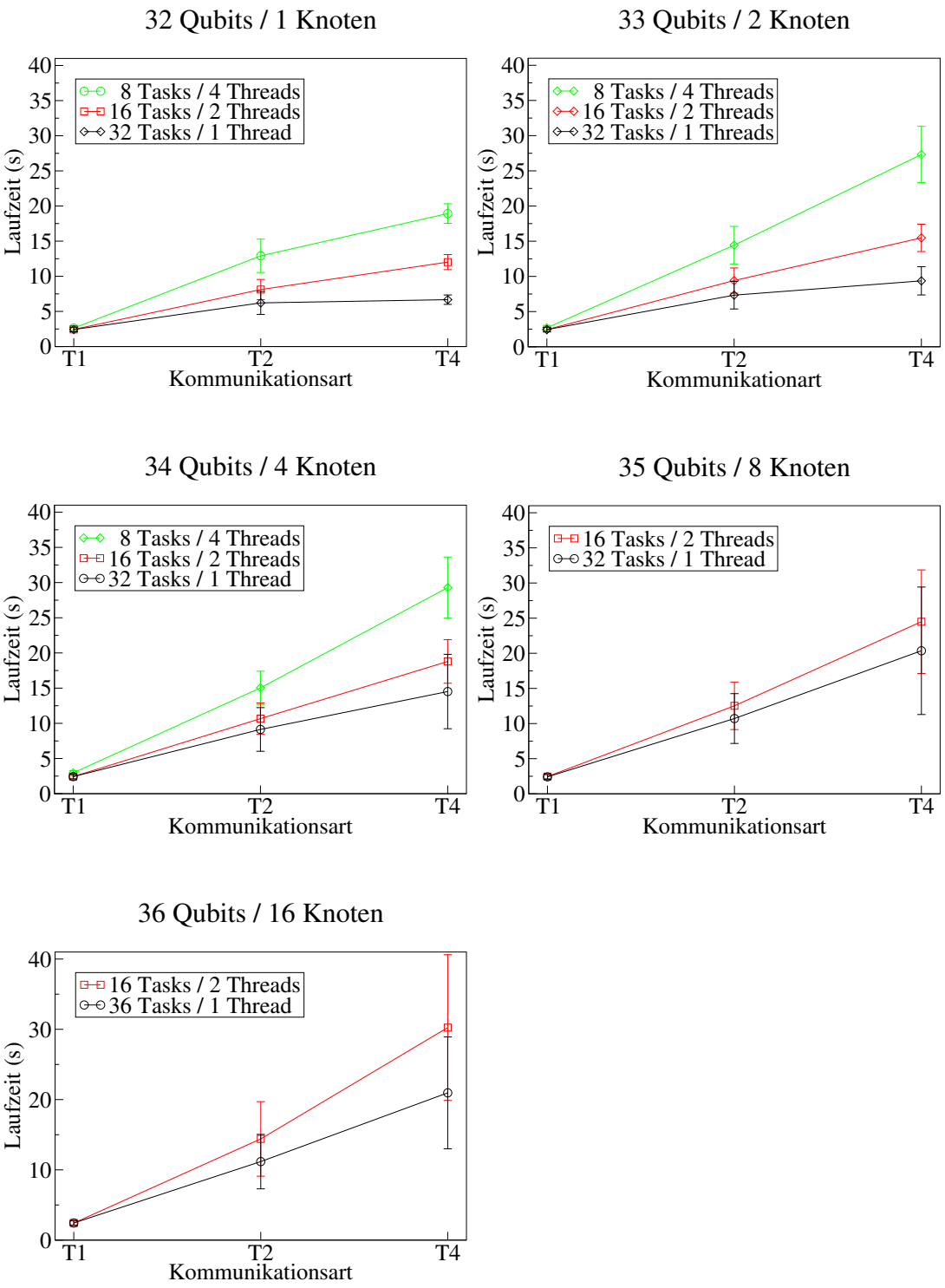

Abbildung 7.6: Laufzeiten verschiedener Kommunikationsarten in Abhängigkeit der Systemgröße und -konfiguration für das CNOT-Gatter

$t = 2$ ab, und eine Konfiguration mit 4 Threads pro MPI-Task ist soviel langsamer, daß der Abstand zur Kurve für $t = 2$ mindestens eine Standardabweichung beträgt.

Alle Messungen des T1-Falls liefern den gleichen Wert von ca. 2,5 s pro Operation, was deutlich macht, daß die Gesamtzeit maßgeblich durch die Kommunikationsdauer beeinflußt wird. Vergrößert sich die Systemgröße um ein Qubit, verdoppelt sich damit die Anzahl der Einträge des Zustandsvektors, gleichzeitig stehen aber auch doppelt so viele Prozessoren zum Berechnen zur Verfügung, so daß die eigentliche Rechnung ohne Kommunikation konstant skaliert.

Die Zeiten für die Fälle T2 und T4 steigen mit zunehmender Systemgröße an. Selbst im schnellsten Fall (1 Thread pro Task) dauert eine T2-Operation bei einer Systemgröße von 32 Qubits zweimal so lange und bei einer Systemgröße von 36 Qubits mehr als viermal so lange wie eine T1 Operation. Bei anderen Konfigurationen ist dieses Verhalten noch extremer.

Bei den T4-Operationen sieht man zum einen, daß die Zeiten unterschiedlicher Konfigurationen noch weiter auseinanderliegen als beim T2-Fall, was darin begründet ist, daß bei weniger zur Verfügung stehenden Tasks die Kommunikationszeit stark ansteigt, da hier pro Task mehr Daten verschickt werden müssen, während für die eigentliche Rechnung durch die Zunahme der Threads eine Mehrbelastung pro Prozessor vermieden wird.

Zum anderen wachsen die Laufzeiten der T4-Operationen mit zunehmender Systemgröße gegenüber den T2-Operation stärker an. Dies ist bei den größeren Systemen durch die größere Anzahl oder überhaupt erst durch das Auftreten von Operation mit Interknoten-Kommunikation begründet, wie in Kapitel 7.3.3 gezeigt wird.

7.3.2 Auswirkungen der Paketgröße

Auch bei der statischen Lastverteilung, bei der bereits von vorneherein feststeht, wieviele Daten jeder Task auslagert, spielt die Paketgröße eine Rolle. Abbildung 7.7 zeigt die Unterschiede in der Operationsdauer zwischen Paketen der Größe 2^{20} und 2^{25} Einträgen also 16 MB bzw. 512 MB. Dabei sind die Zeiten für T2- und T4-Kommunikation bei unterschiedlichen Systemgrößen angegeben. Bei der T2-Kommunikation spielt die Paketgröße nur eine eher untergeordnete Rolle, hier ist kein Laufzeitunterschied erkennbar. Bei der T4-Kommunikation ist der Datentransfer mit großen Paketen allerdings um ca. 3 s pro Operation schneller. Die Kurve zwischen den verschiedenen Paketgrößen stellt den Mittelwert aus beiden dar. In den vorherigen und in weiteren Graphen wurden die schnelleren Werte der größeren Pakete verwendet.

7.3.3 Auswirkungen der Systemgröße

Um eine differenziertere Aussage über die Zeitdauern der Operationsbestandteile zu erhalten, werden für den Fall eines Threads pro Task, der sich als am schnellsten erwiesen hat (s. Abbildung 7.6), die Laufzeiten der verschiedenen Kommunikationsarten T1, T2 und T4 über der Systemgröße betrachtet (s. Abbildung 7.8). Wie erwartet, unterscheiden sich die Zeiten im Fall ohne Kommunikation nicht. Im Fall der T2- und

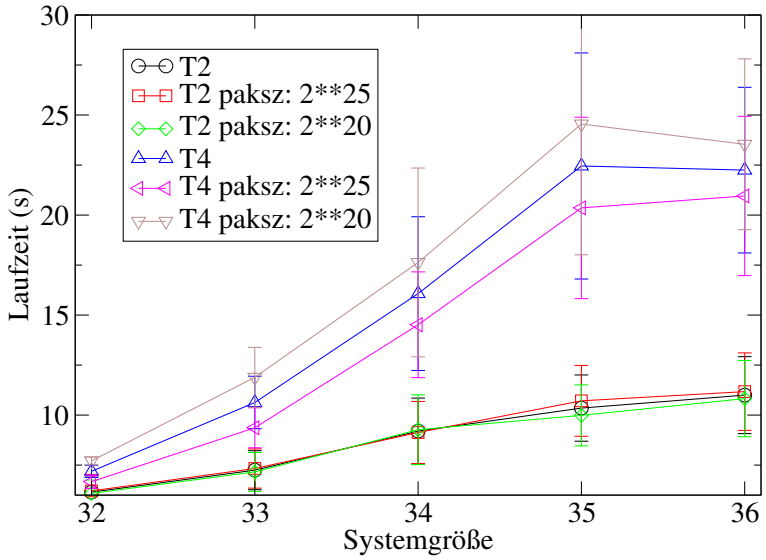

Abbildung 7.7: T2- und T4-Kommunikation in Abhängigkeit der Paketgröße

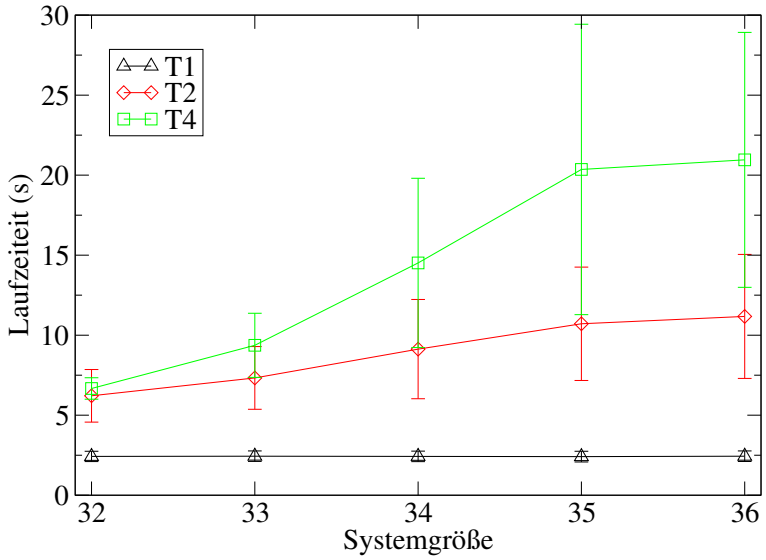

Abbildung 7.8: T1-, T2- und T4-Kommunikation in Abhängigkeit der Systemgröße

T4-Kommunikation wächst die Zeit kontinuierlich mit der Systemgröße an, wobei sich im Fall T4 ein deutlich steilerer Anstieg zeigt. Ab einer Systemgröße von 35 Qubits scheinen sich die Zeiten nur noch wenig zu verändern. Bei einem 32 Qubits umfassenden System gibt es praktisch keinen Unterschied, während bei einem 35 Qubit-System die T4-Operationen ungefähr doppelt so lange brauchen wie die T2-Operationen. Dieses Verhalten wird verständlich, schaut man sich in Abbildung 7.9 das Laufzeitverhalten differenziert nach Kommunikation innerhalb und außerhalb physikalischer Knoten

an. Charakteristisch für das Kommunikationsverhalten ist eine konstante Zeitdauer für

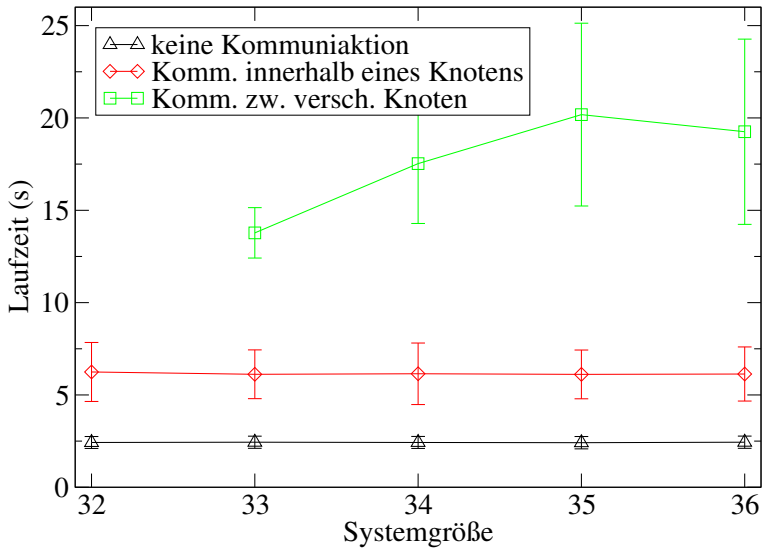

Abbildung 7.9: Rolle der Kommunikation innerhalb und zwischen physikalischen Knoten in Abhängigkeit der Systemgröße

Kommunikation innerhalb eines Knotens, egal ob es sich um eine T2- oder T4-Operation handelt. Diese liegt zwar mit ungefähr 6 s mehr als 100% über der Operationsdauer ohne Kommunikation, hängt aber nicht von der Größe des Systems ab. Nicht konstant verhält sich die benötigte Zeit, um Daten zwischen verschiedenen Knoten über das Netzwerk zu übertragen, wobei das Absinken beieinem 36-Qubit-System im Vergleich zum 35-Qubit-Fall nur durch eine statistische Schwankung erklärbar ist.

Damit läßt sich nun das Verhalten der Kurven aus Abbildung 7.8 interpretieren. Solange keine Kommunikation zwischen mehreren Knoten notwendig ist, die erst ab einer Systemgröße von 33 Qubits einsetzt, bleibt auch die Zeit für den T2- sowie den T4-Fall auf dem Niveau der Inner-Knoten-Kommunikation. Mit wachsender Systemgröße steigt die Zahl der T2- und T4-Operationen, die Kommunikation zwischen den Knoten benötigen. Vor allem durch Zunahme der zeitintensiven T4-Operationen, deren Kommunikation sich gleichzeitig über vier Knoten erstreckt (s. Abbildung 7.10), wird die durchschnittliche Operationsdauer der T4-Operationen ab einer Systemgröße von 34 Qubits weiter erhöht. Abbildung 7.10 zeigt noch einmal, daß selbst bei der Kommunikation zwischen verschiedenen Knoten nicht alle Prozesse gleichlange dauern. Als Referenz dient hier die Inter-Knoten-Kommunikationskurve aus dem vorherigen Graphen. Diese wird ergänzt durch Kurven, die die Operationsdauern der Bereiche IV, V und VI einzeln darstellen. Man sieht hier deutlich, daß die T4-Operationen, deren zusammengehörige Einträge des Zustandsvektors über 4 Knoten verteilt liegen, deutlich länger brauchen, als der Rest der T4-Operationen. Auch die große Streuung der Werte zeugt von starken Unregelmäßigkeiten in der Geschwindigkeit des Netzwerkes.

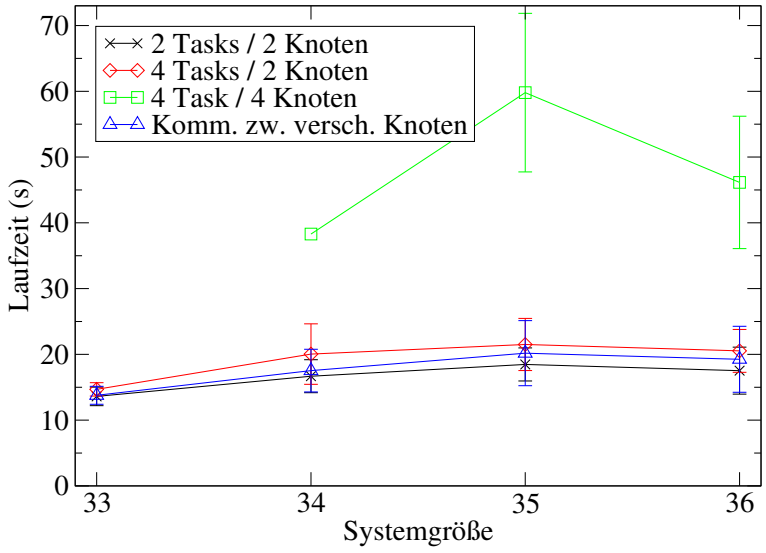

Abbildung 7.10: Laufzeiten bei Kommunikation zwischen physikalischen Knoten in Abhängigkeit der Systemgröße

7.3.4 Vergleich der Laufzeiten verschiedener Systeme

Bildet man nun den Mittelwert über alle durchgeführten Operationen einer Messung und stellt diese für verschiedene Konfigurationen und Systemgrößen, wie auch schon für das Hadamard-Gatter geschehen, in Abhängigkeit von der Anzahl der benutzten Prozessoren dar, ergibt sich Abbildung 7.11. Die Laufzeitunterschiede einer durchschnittlichen

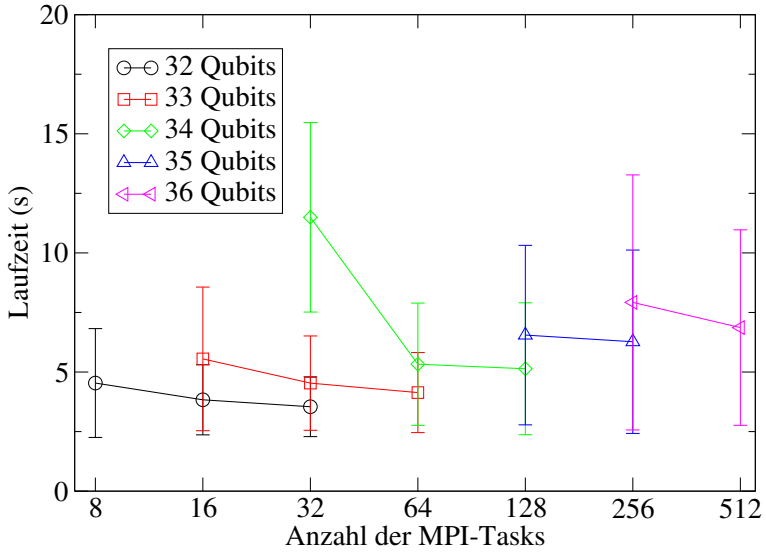

Abbildung 7.11: Laufzeiten der CNOT-Operation in Abhängigkeit von Systemgröße und Anzahl der MPI-Prozesse

CNOT-Operation zwischen 16 und 32 Threads pro Tasks sind bei allen Systemgrößen vernachlässigbar gering, nur bei dem Fall $t = 4$ erkennt man einen z.T. erheblichen Zeitanstieg bei einer Systemgröße von 34 Qubits. Dieser nur kleine Unterschied vor allem zwischen den Konfigurationen $t = 1$ und $t = 2$ resultiert aus der arithmetischen Mittelung der Laufzeiten aller Operationen, die durch die große Anzahl schneller Operationen ohne Kommunikation dominiert werden. Um eine differenziertere Aussage über das Laufzeitverhalten zu bekommen, werden daher im folgenden die Laufzeiten für die einzelnen Kommunikationsfälle separat dargestellt.

Abbildung 7.12 zeigt ein fast konstantes Verhalten der T1-Operationen unabhängig von der Systemgröße. Abweichungen zu höheren Werten bei 32-, 33- und 34-Qubit-Systemen und vier Threads pro Task deuten auf ein schlechteres Abschneiden der OpenMP-Parallelisierung gegenüber der Aufteilung in mehrere MPI-Prozesse hin. Obwohl also der Rechenaufwand pro Prozessor in allen Fällen der gleiche ist, ist die Aufteilung der Rechenschleife auf die zum Task gehörigen Prozessoren für mehr als 2 Prozessoren bis zu 20% langsamer. Deutlich ausgeprägter werden die Unterschiede zwischen den ver-

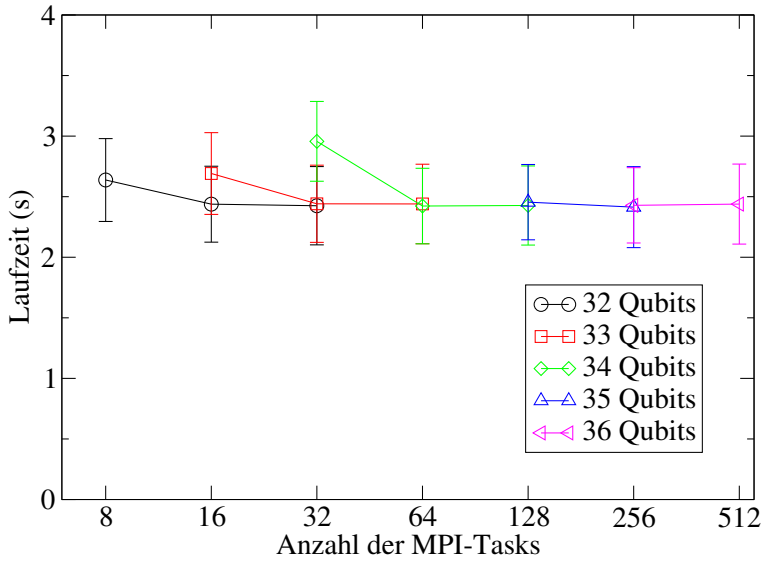

Abbildung 7.12: Laufzeiten der T1-Operationen in Abhängigkeit von Systemgröße und Anzahl der MPI-Tasks

schiedenen Konfigurationen, wenn man sich den kommunikationsbehafteten Routinen zuwendet. Bei den T2-Operationen stellt sich eine Laufzeitverlängerung zwischen 20% und 30% ein, wenn statt reinem MPI zwei Threads pro Task verwendet werden. Bei einer Konfiguration mit 4 Threads pro Task steigen die Zeiten bei den gemessenen System im Vergleich zum reinen MPI um 50% bis 100%. (s. Abbildung 7.13). Diese Unterschiede werden im Fall T4 besonders deutlich. Durch ein gesteigertes Kommunikationsaufkommen um durchschnittlich einen Faktor 1,5 pro Prozessor gegenüber dem T2-Fall ergeben sich zum einen für den reinen MPI-Fall auch ca. 1,5-fach größere Absolutwerte für die Laufzeiten, andererseits werden auch die Unterschiede zwischen verschiedenen Konfi-

gurationen größer. So lagen die Zeiten einer Konfiguration mit zwei Threads pro Task zwischen 25% und 75% höher und die Zeiten einer Konfiguration mit vier Threads pro Task sogar um 100% bis 150% höher als eine reine Steuerung der Kommunikationsprozesse mit MPI (s. Abbildung 7.14). Dieser Geschwindigkeitsgewinn resultiert daher, daß durch das Aufteilen des Zustandsvektors auf mehr MPI-Prozesse die kleinere Anzahl von Prozessoren pro MPI-Prozess auf weniger stark segmentierte Speicherbereiche zugreifen muß (s. Kapitel 6.4).

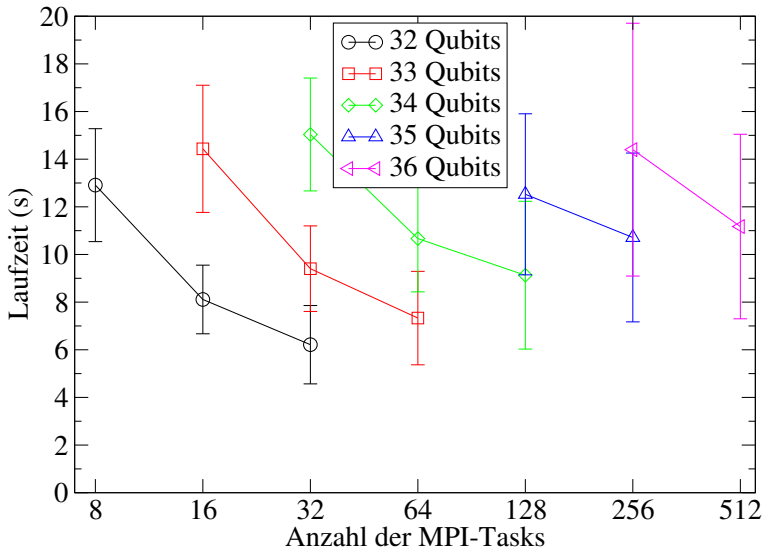

Abbildung 7.13: Laufzeiten der T2-Operationen in Abhängigkeit von Systemgröße und Anzahl der MPI-Tasks

7.4 Dynamische Lastverteilung

Das Konzept der dynamischen Lastverteilung wird in Abschnitt 6.8.3 beschrieben. Der wesentliche Unterschied zur statischen Lastverteilung besteht darin, daß neben der Paketgröße ein weiterer Parameter zur Verfügung steht, um Einfluß auf das Kommunikationsverhalten des Programmes zu nehmen, nämlich die Anzahl der sich gleichzeitig im Sendeprozess befindlichen Datenpakete.

7.4.1 Auswirkung von Paketgröße und gleichzeitig verschickten Paketen

Wie aus Abbildung 7.15 ersichtlich, spielt die Paketgröße hier eine weit größere Rolle als bei der statischen Lastverteilung. Durch geeignete Wahl der Paketgröße kann die durchschnittliche Kommunikationszeit für eine CNOT-Operation bis zu 25% gegenüber dem statischen Fall reduziert werden. Bei einer Paketgröße ab 2^{24} Einträgen pro Paket

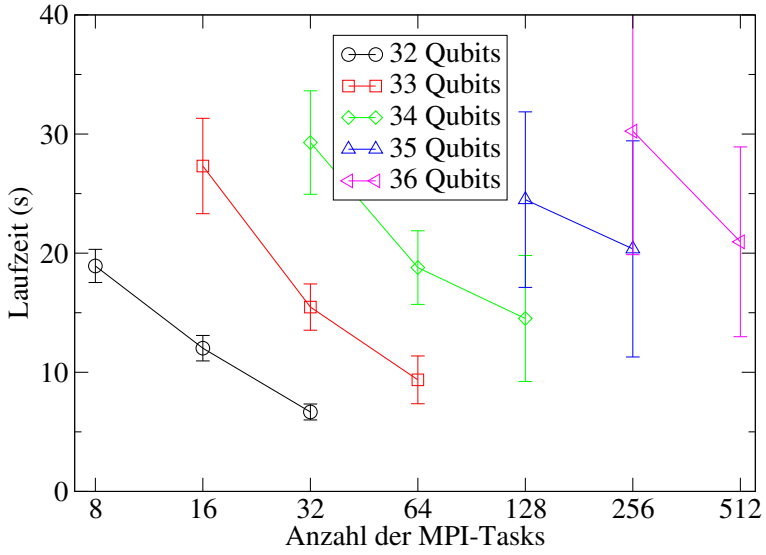

Abbildung 7.14: Laufzeiten der T4-Operationen in Abhängigkeit von Systemgröße und Anzahl der MPI-Tasks

fällt die durchschnittliche Transfergeschwindigkeit allerdings mit zunehmender Systemgröße stark ab und bleibt sogar hinter der des statischen Falls zurück. Wie weiter oben

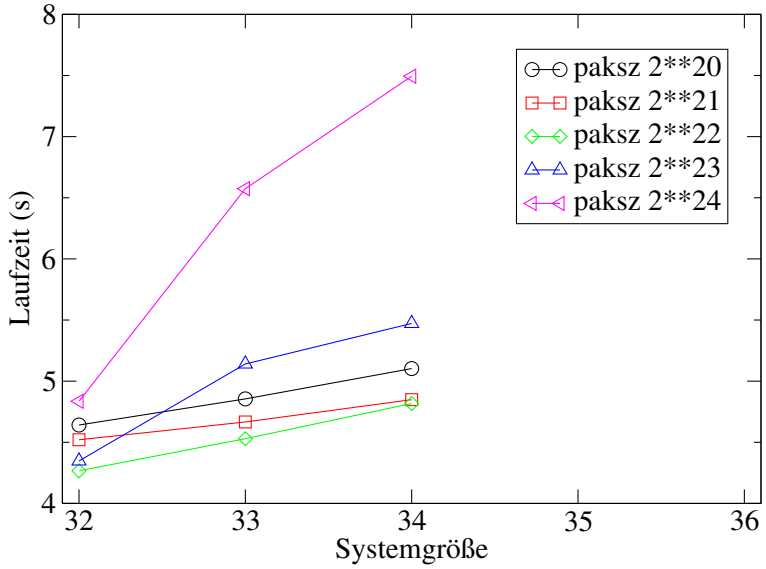

Abbildung 7.15: Einfluß der Paketgröße auf Transfergeschwindigkeit

erwähnt, hat auch die Anzahl der sich gleichzeitig im Sendeprozess befindlichen Pakete eine deutliche Auswirkung auf die Kommunikationszeit der Operation. Wird sichergestellt, daß nur eine einzige nicht blockierende Senderoutine zur gleichen Zeit aufgerufen wird, so läuft die Kommunikation am schnellsten ab. Schon bei nur 2 gleichzeitig gesen-

deten Paketen erhöht sich die Kommunikationszeit um 25%. Eine weitere Erhöhung der Zahlen gleichzeitig verschickter Pakete auf 4 bzw. 8 hat ein Anwachsen der Kommunikationsdauer von ca. 50% zur Folge. Dieses Verhalten wird aus Abbildung 7.16 deutlich. Im Rahmen dieser Arbeit wurden nur Messungen mit Systemen bis zu einer Größe von

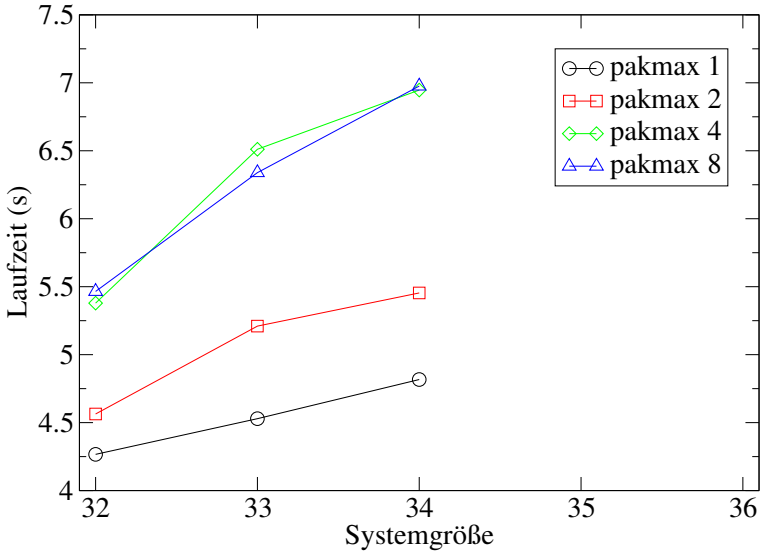

Abbildung 7.16: Einfluß der gleichzeitig geschickten Pakete auf Transfergeschwindigkeit

34 Qubits durchgeführt, da eine Ermittelung der minimalen Kommunikationszeit mit entsprechend vielen Meßpunkten für Paketgröße und gleichzeitig gesendeten Paketen für größere Systeme unverhältnismäßig viel Rechenzeit in Anspruch genommen hätte. Da alle untersuchten Systeme das gleiche tendenzielle Verhalten zeigen, kann man davon ausgehen, daß die ermittelten Minima sich auch bei größeren System nicht mehr als um eine 2er Potenz in der Paketgröße und der Anzahl gleichzeitig verschickter Pakete verschieben.

7.4.2 Vergleich zur statischen Lastverteilung

Daß sich der Aufwand zur Programmierung einer dynamischen Lastverteilung gelohnt hat, sieht man, wenn man sich die durchschnittlichen Laufzeiten der CNOT-Operation für den statischen und dynamischen Fall zusammen betrachtet (s. Abbildung 7.17). Deutlich erkennt man das charakteristische Auseinanderlaufen der Graphen der statischen Lastverteilung in Richtung größer werdender Systeme für die Kommunikationsfälle T1, T2 und T4. Diese Unterscheidung ermöglicht den Vergleich mit den für den T2-Fall gemessenen Zeiten der dynamischen Lastverteilung.

Man erkennt den deutlich schwächeren Anstieg der Laufzeiten einer dynamischen CNOT-Operation für zunehmende Systemgrößen, die immer eindeutig unter denen der statischen Lastverteilung liegen. Bei der kleinsten betrachteten Systemgröße von 32 Qubits bewirkt die dynamische Lastverteilung eine Laufzeitverringerung von knapp 30%,

bei einer Systemgröße von 36 Qubits vergrößert sich der Laufzeitunterschied bis auf über 50%. Weiterhin kann man an Hand der viel geringeren Standardabweichung im dynami-

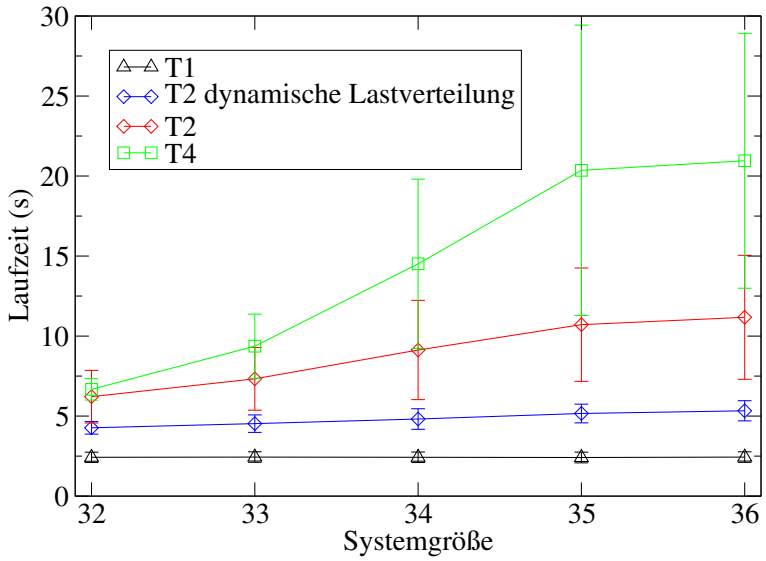

Abbildung 7.17: Statische T1, T2 und T4 Kommunikation und dynamische T2 Kommunikation in Abhängigkeit der Systemgröße

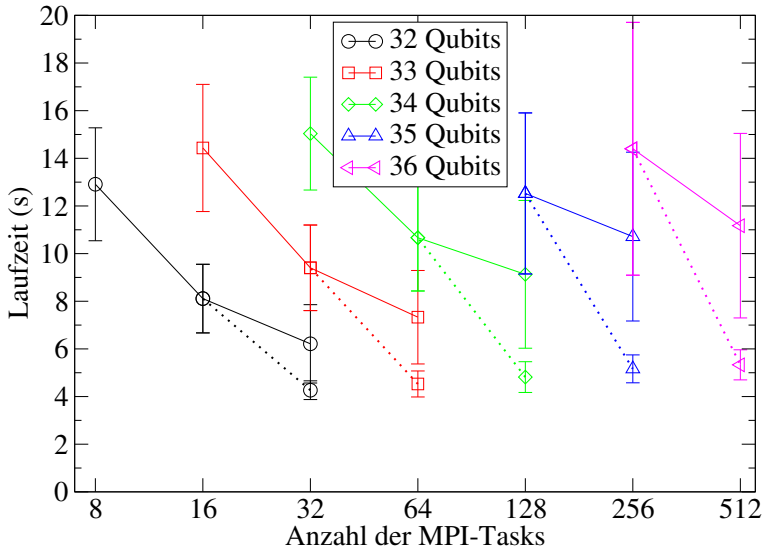

Abbildung 7.18: Skalierungsverhalten der CNOT-Operation abhängig von der Systemgröße und Anzahl der MPI-Tasks im statischen und dynamischen Fall

schen Fall die Aussage treffen, daß die Laufzeiten der einzelnen MPI-Prozesse wesentlich näher beieinander liegen als im statischen Fall, was nach dem Konzept der dynamischen Verteilung auch zu erwarten ist.

Abbildung 7.18 zeigt nochmal den in Abbildung 7.13 dargestellten Zusammenhang, nun aber ergänzt durch die dynamisch lastverteilten Laufzeiten, die durch die gestrichelten Linien angedeutet sind. Auch in dieser Darstellung wird die starke Laufzeitabnahme für eine maximale Anzahl von MPI-Tasks mit dynamischer Lastverteilung deutlich.

Durch die im T2-Kommunikationsfall erzielten Resultate bietet es sich an, eine Erweiterung der dynamischen Lastverteilung auf den T4-Fall zu implementieren, die eine entsprechende Verringerung der Lauftzeit mit sich bringen würde. Weiterhin wäre eine Untersuchung der Laufzeiten bei anderen MPI-/OpenMP-Konfigurationen durchzuführen, um auch hier den Zeitgewinn der dynamischen Lastverteilung quantifizieren zu können.

8 Auswirkungen von Gatterfehlern und Dekohärenz

Ein idealer Quantencomputer ist ein vollständig abgeschlossenes und isoliertes System. Es existieren weder Wechselwirkungen mit der Umgebung noch Ungenauigkeiten beim Ausführen des Algorithmus'. Ein reales Quantensystem ist dagegen auch im Fall einer nahezu perfekten Abschirmung in ständigem Austausch mit seiner Umgebung, sei es durch Steuerpulse zum Ausführen einer Operation oder durch Dekohärenzeffekte. Diese ungewollten Effekte, durch die dem System Informationen verloren gehen, haben großen Einfluß auf das Ergebnis einer Rechnung. So limitiert der ständige Informationsverlust z.B. die Anzahl der durchführbaren Rechenoperationen, die durch die Dauer der Dekohärenzzeit festgelegt ist. Voraussagen über das Verhalten eines Quantencomputers unter dem Einfluß von Fehlereffekten sind daher extrem wichtig zum Verständnis und zur Konstruktion eines Quantencomputers.

Untersuchungen zur Entwicklung des Zustandes eines bis zu 20 Qubits großen Systems unter dem Einfluß von fehlerbehafteten Hadamard-Operationen und Dekohärenz wurden in [27] gemacht. Simuliert wurden dabei jeweils 100 aufeinanderfolgende Hadamard-Transformationen, d.h. eine Hadamard-Operation auf alle involvierten Qubits mit verschiedenen Parametern für die Größe der Gatterfehler und die Wahrscheinlichkeit für ein Auftreten von Dekohärenzeffekten. Diese Untersuchungen wurden in dieser Arbeit auf Systemgrößen von 25 und 32 Qubits ausgedehnt. Ziel war es, die Entwicklung des Zustandsvektors in Abhängigkeit der Qubitanzahl des Systems und der Fehlergröße zu untersuchen, um auch für größere Systeme das theoretisch vorausgesagte Verhalten bestätigen zu können. Darüber hinaus wurden auf dem kleineren simulierten System mit 25 Qubits Simulationen mit bis zu 500 sukzessiven Hadamard-Transformationen durchgeführt, um das Abklingen der Amplitude über 100 Operationsschritte hinaus zu simulieren.

8.1 Implementierung fehlerbehafteter Hadamard-Gatter

Um die Einwirkung eines fehlerhaften Gatters quantitativ untersuchen zu können, muß die Größe des Fehlers parametrisiert werden. Dies geschieht beim Hadamard-Gatter

durch die Aufteilung in eine Drehung

$$\mathbf{R}(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \tag{8.1}$$

und eine Phasen-Shift-Operation

$$\mathbf{U}(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \tag{8.2}$$

Damit kann das Hadamard-Gatter in Abhängigkeit der Winkel θ und ϕ als

$$\mathbf{H}(\theta, \phi) = \mathbf{R}(\frac{\pi}{4})\mathbf{U}(\pi) \tag{8.3}$$

ausgedrückt werden ([27]) und ermöglicht die Beeinflussung der Operation durch Modifikation dieser Winkel. Eine fehlerbehaftete Operation entsteht dabei durch eine Gaussverteilte, unabhängige Abweichung dieser Winkel von den idealen Werten $\frac{\pi}{4}$ bzw. π . Die Standardabweichung σ dieser Verteilung stellt ein Maß für die Stärke des Fehlers dar, der unter Verwendung von Zufallszahlen generiert wird (s. Kapitel 8.3). Dabei werden für jede Hadamard-Operation $\mathbf{H}_q$ innerhalb einer Hadamard-Transformation

$$\mathbf{H} = \mathbf{H}_1 \otimes \ldots \otimes \mathbf{H}_q \tag{8.4}$$

zwei neue Zufallszahlen generiert.

Die Methode zur Untersuchung des fehlerhaften Hadamard-Gatters basiert auf der Tatsache, daß zwei hintereinander ausgeführte, ideale Hadamard-Transformationen das System wieder in den Ausgangszustand überführen:

$$\mathbf{H}^2 = \mathbf{I} \tag{8.5}$$

Wird nun die ideale Hadamard-Operation, wie in Kapitel 8.1 beschrieben, durch eine Drehung $\mathbf{U}(\pi)$ und eine anschließende Rotation $\mathbf{R}(\frac{\pi}{4})$ ersetzt, bei denen die Winkel von den idealen Werten abweichen, wird der Systemzustand nach 2 realen Hadamard-Transformationen vom Anfangszustand abweichen. Diese Abweichungen stellen die Auswirkungen des fehlerhaften Gatters dar. Durch die Aufteilung des Hadamard-Gatters in zwei separate Operationen steigt natürlich auch der Rechenaufwand und damit die Laufzeit des Programmes.

Um die Ergebnisse der Simulationen bewerten zu können, wurden diese mit theoretisch berechneten Werten verglichen. Analytisch ergibt sich der Wert des $|0\rangle\langle 0|$-Elements nach Anwenden von k Hadamard-Transformationen auf ein System aus n Qubits zu

$$\rho_k = \frac{1}{2}\left(\frac{1 + e^{-\frac{\sigma^2}{4}9k}}{2}\right)^n . \tag{8.6}$$

8.2 Implementierung der Dekohärenzeffekte

Bei der Simulation der Dekohärenzeffekte wird nach jeder Hadamard-Operation **RU** die Transformation $\rho \rightarrow \rho'$ (Kapitel 4.4.1) auf alle sich in dem System befindlichen Qubits angewendet, wobei die Stärke der Effekte durch die Wahrscheinlichkeit des Auftretens eines Fehlers p festgelegt wird. Dabei wird ein Zufallswert generiert und mit p verglichen. Tritt ein Dekohärenzfall auf, müssen zusätzliche X-, Y- oder Phase-Shift-Operationen ausgeführt werden.

Die Anwendung des Dekohärenz-Algorithmus' kann eine weitaus größere Beeinflussung der Amplituden der Basiszustände in einem Simulationsschritt bewirken als fehlerbehaftete Gatteroperationen, die zu einer langsamen Abnahme der Amplitude führen. Bei der Simulation einer Dichtematrix durch wiederholte Meßreihen mit dem Zustandsvektor eines Systems können so bei der Mittelwertbildung der Meßwerte extreme Schwankungen und eine große Streuung der Meßwerte auftreten. Simulationen eines Zustandsvektors unter dem Einfluß von Dekohärenz müssen daher öfter ausgeführt werden als Simulationen fehlerbehafteter Gatter, um die gleiche Standardabweichung der Werte zu erhalten.

Die Auswirkungen der Dekohärenzeffekte werden nach jeder Hadamard-Transformation für jedes Qubit einzeln berechnet. Dabei wird jedes Qubit der Transformation aus Kapitel 4.4.1 unterzogen. Auch hier wird für die Entscheidung, ob und welche Form der Dekohärenz eintritt, für jedes Qubit eine neue Zufallszahl generiert.

Anwenden des Dekohärenzmodells auf den Zustand eines n-Qubit Systems nach k Operationen führt zu einer theoretischen Vorhersage des Wertes für das $|0\rangle\langle 0|$-Element von

$$\rho_k = \left(\frac{1 + (1 - \frac{3}{4}p)^k}{2} \right)^n . \tag{8.7}$$

8.3 Durchführung der Simulationen

Das System startet im Zustand $|...000\rangle$, so daß die Amplitude dieses Basisvektors zu Beginn der Simulation gleich 1 ist. Um eine qualitative Aussage über den Abfall der Amplitude zu bekommen, wird nach jeweils zwei Hadamard-Transformationen das $|0\rangle\langle 0|$-Element der Dichtematrix berechnet und ausgegeben.

Da das Simulationsprogramm nicht in der Lage ist, die Dichtematrix eines Systems zu simulieren (s. Kapitel 6.3), werden die Ergebnisse aus mehreren hintereinander ausgeführten Simulationsdurchläufen für jeden Meßpunkt durch Mittelwertbildung zusammengefaßt. Die Dichtematrix des Systems wird daher durch eine Mittelung über mehrere Programmdurchläufe angenähert.

Man beschränkt sich hier in Folge Platzmangels auf die Entwicklung des $|0\rangle\langle 0|$-Elements der Dichtematrix, denn der gesamte Zustandsvektor hätte bei einem System mit 32 Qubits eine Größe von 64 Gigabyte, was bei der Weiterverarbeitung eine Dichtematrix der Größe $4,096 \times 10^{21}$ Byte hervorbringen würde. Dies würde zu großen Problemen bei Speicherung und Weiterverarbeitung der Daten führen.

Alle Simulationen des 25-Qubit-Systems wurden 100 Mal durchgeführt. Bei dem größeren System mit 32 Qubits konnten auf Grund der hohen Rechenzeitkosten nur jeweils 4-20 Durchläufe pro Messung gestartet werden. 100 Hadamard-Transformationen auf einem System dieser Größe dauerten selbst bei Verwendung von 8 Rechnerknoten mit 256 Prozessoren 1,75 Stunden, was ungefähr einem Zehntel der monatlich für diese Arbeit zur Verfügung stehenden Rechenzeit entspricht. Dies ist ein Kompromiß zwischen der Simulation eines möglichst großen Systems und möglichst vielen Programmdurchläufen, um eine aussagekräftige Statistik zu erhalten.

Die Anzahl der simulierten Qubits ist nun nicht mehr nur durch die Größe des Arbeitsspeichers begrenzt, sondern auch durch eine akzeptable Simulationszeit. Ein Simulationsdurchlauf mit 100 fehlerbehafteten Hadamard-Transformationen auf einem 32 Qubit-System beinhaltet $100 \times 2 \times 32 = 6400$ Operationen zuzüglich der Generierung der Zufallszahlen. Um eine auswertbare Statistik zu erhalten, sind viele Durchläufe nötig, was die Anzahl der Operationen und so die Rechenzeit um ein Vielfaches ansteigen läßt. Um Ergebnisse in möglichst kurzer Rechenzeit zu erhalten, wurde nun der Arbeitsspeicher eines Knotens nicht maximal ausgenutzt, sondern es wurde versucht, ein gutes Verhältnis von Systemgröße und Prozessoranzahl zu erreichen. So wurden die Simulationen des 32-Qubit-Systems auf 8 Rechenknoten mit insgesamt 256 Prozessoren ausgeführt, die Simulationen des 25-Qubit-Systems auf 2 bzw. 4 Rechenknoten mit insgesamt 32 bzw. 64 Prozessoren.

Auf Grund der begrenzten Rechenzeit mußten auch für die hier diskutierten Systemgrößen Einschränkungen für die Parameter σ und p getroffen werden. Daher wurden fehlerhafte Hadamard-Transformationen für σ-Werte von 10^{-3} und 10^{-4} durchgeführt. Die Wahrscheinlichkeiten für das Auftreten von Dekohärenz p lagen in der gleichen Größenordnung. Größere Werte führen zu einem extrem schnellen Abfall der Amplitude, so daß schon nach wenigen Operationen der Wert 0 erreicht wäre.

Obwohl für 32 Qubit-Systeme nur Ergebnisse aus wenigen Programmdurchläufen zur Verfügung standen, kann man trotz mangelnder statistischer Aussagekraft für 32 Qubit-Systeme dennoch ähnliche Grundtendenzen wie bei kleineren Systemen erkennen.

Da klassische Computer deterministische Rechenmaschinen sind, kann mit ihnen kein wirklicher Zufall simuliert werden. Stattdessen wird eine Pseudo-Zufallszahl beispielsweise durch einen auf der Systemzeit basierenden Algorithmus erzeugt. Es gibt mehrere Indikatoren für die Güte eines solchen Algorithmus'. So z. B. die Periodizität, d.h. die Anzahl von generierten Zufallszahlen, nach der sich die Zahlenwerte wiederholen oder die Stärke der Korrelationen der Zufallszahlen untereinander.

Bei der Simulation wurde hier der MT19937-Mersenne-Twister-Algorithmus [29] benutzt. Er hat nachweislich eine Periodizität von $2^{19937} - 1$. Weil die Anzahl der bei allen durchgeführten Simulationsrechnungen generierten Zufallszahlen weit kleiner ist, als dieser Wert, kann ein systematischer, durch den Zufallszahlengenerator verursachter Fehler ausgeschlossen werden, solange man davon ausgeht, daß die Zufallszahlen untereinander keine Korrelationen aufweisen.

Um im Fall der fehlerbehafteten Gatter Gauss-verteilte Zufallswerte zu erhalten, wird mit jeweils zwei Zufallszahlen r_1 und r_2 in Abhängigkeit der Standardabweichung

σ eine neue Zufallsvariable h nach dem in [30] vorgestellten Verfahren erzeugt.

$$h = \sqrt{2} \cdot \sigma \cdot \sqrt{-\ln(r_1)} \cdot \cos(2\pi \cdot r_2) \tag{8.8}$$

8.4 Auswirkungen fehlerhafter Hadamard-Transformationen

In den Abbildungen 8.1 und 8.2 ist die Entwicklung des $|0\rangle\langle 0|$-Matrixelements der Dichtematrix für Systeme mit 25 und 32 Qubits bei der Anwendung fehlerhafter Hadamard-Gatter dargestellt. Die Graphen unterscheiden sich durch die Größe des Fehlers, dar-

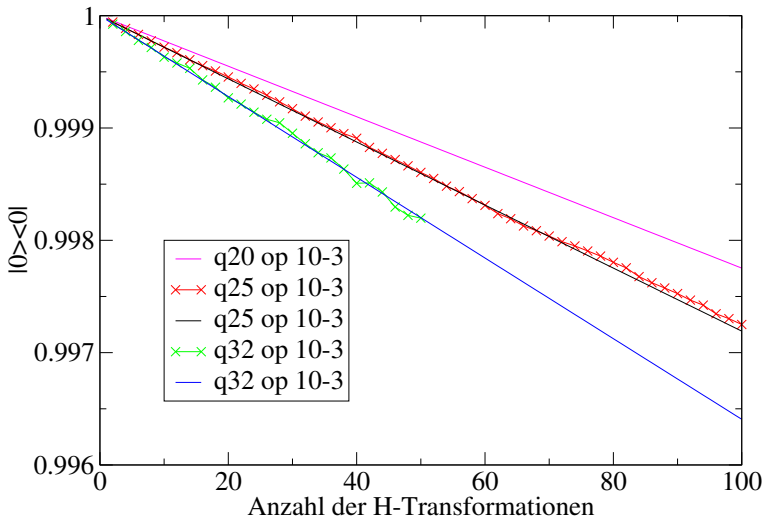

Abbildung 8.1: Entwicklung des $|0\rangle\langle 0|$-Matrixelements mit $\sigma = 10^{-3}$ für verschiedene Systemgrößen

gestellt durch den Parameter σ, der in Abbildung 8.1 den Wert 10^{-3} und in Abbildung 8.2 den Wert 10^{-4} hat. Die Größe von σ hat dabei starken Einfluß auf die Abnahme der der $|0\rangle\langle 0|$-Amplitude, was in dem um 2 Zehnerpotenzen auseinanderliegenden Maßstab der y-Achsen deutlich wird.

Die durch Kreuze markierten Meßwerte stellen den Mittelwert der simulierten Messungen dar, die durchgezogenen Linien zeigen die nach Gleichung 8.6 berechnete theoretische Entwicklung des Matrixelements an. Die theoretischen Werte sind für Systeme mit 20, 25 und 32 Qubits nach Gleichung 8.7 gegeben.

Für das 32-Qubit-System wurden im Fall $\sigma = 10^{-3}$ 10 Messungen mit jeweils 50 aufeinanderfolgenden H-Transformationen durchgeführt, im Fall $\sigma = 10^{-4}$ wurden zwar 100 H-Transformationen ausgeführt, jedoch konnten hierbei nur 4 Messungen simuliert werden. Es ist erkennbar daß schon die 2,5-fache Anzahl an Messungen eine deutliche Abnahme der Schwankungen der Simulationswerte um den theoretischen Wert bewirkt.

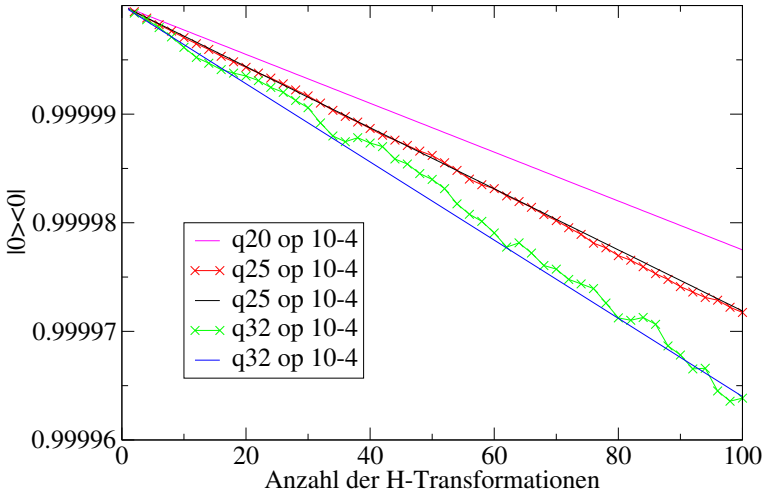

Abbildung 8.2: Entwicklung des $|0\rangle\langle 0|$-Matrixelements mit $\sigma = 10^{-4}$ für verschiedene Systemgrößen

Schon bei der Mittelung über 100 Meßreihen, wie bei dem 25-Qubit-System geschehen, fällt die relative Standardabweichung der Meßwerte unter 0,01 Prozent des Meßwertes.

20 Qubits stellen das größte in [27] simulierte System dar, für das die Übereinstimmung von Simulation und Theorie sehr gut nachgewiesen werden konnte. Für die hier simulierten Systemgrößen von 25 und 32 Qubits erkennt man ebenfalls eine sehr gute Übereinstimmung mit den theoretischen Vorhersagen. Aus dem Vergleich der Abbildungen 8.1 und 8.2 ist ersichtlich, daß ein Erhöhen der Standardabweichung für die Generierung des Fehlers um den Faktor 10 hat einen um ca. 100-fach stärkeren Abfall der Amplitude zur Folge. Auch die Größe des Systems hat sichtbaren Einfluß auf das System, so nimmt die Amplitude pro Operationsschritt um so stärker ab, je größer das System ist.

Aus den Abbildungen 8.3 und 8.4 erkennt man besonders deutlich, daß sich eine Änderung von σ in einem polynomialen Abfall der Amplitude des Matrixelements bemerkbar macht. Für das 25-Qubit-System bestätigt sich diese Vermutung in den Ergebnissen für $\sigma = 10^{-2}$. Durch die stark unterschiedlichen Wertebereiche der Kurven verschiedener σ-Parameter lassen sich die genauen Kurvenverläufe nur bedingt vergleichen, wie am Beispiel von Abbildung 8.3 deutlich wird. Hier liegt der Schwerpunkt auf der Darstellung des Abfalls der Amplitude bei verschiedenen Fehlergrößen, auf die theoretischen Kurven wurde daher verzichtet. Abbildung 8.5 beweist, daß sich dieses Verhalten auch für mehr als 100 Operationen fortsetzt. Der Abfall der Amplitude wird hier in einem 25-Qubit-System für $\sigma = 10^{-4}$ über 500 Operationsaufrufe hinweg gezeigt.

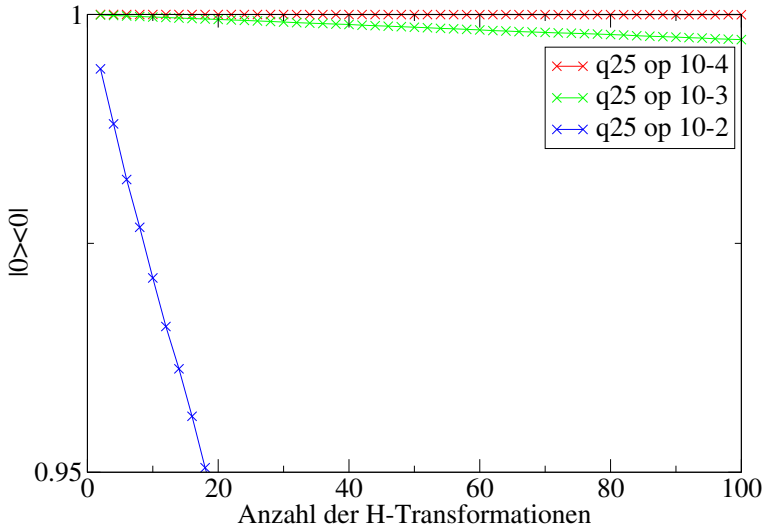

Abbildung 8.3: Entwicklung des $|0\rangle\langle 0|$-Matrixelements für ein 25-Qubit-System für verschiedene σ

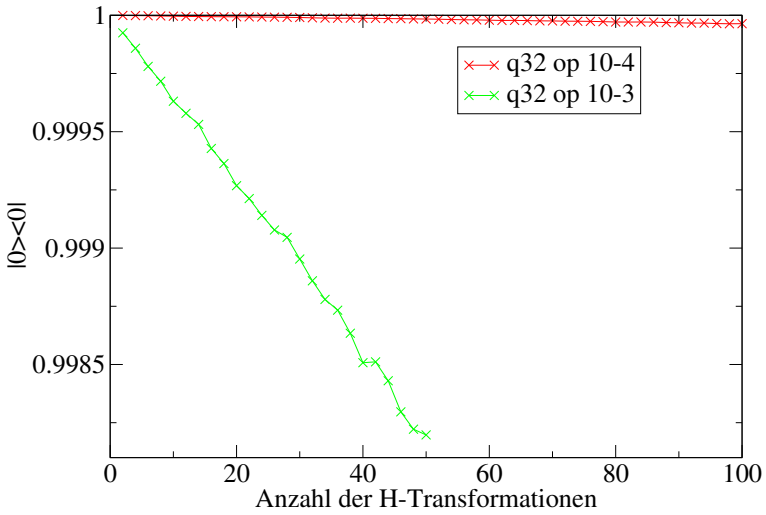

Abbildung 8.4: Entwicklung des $|0\rangle\langle 0|$-Matrixelements für ein 32-Qubit-System für verschiedene σ

8.5 Auswirkungen der Dekohärenzeffekte

Nach dem gleichen Schema wie im vorherigen Abschnitt wird nun in Abbildung 8.6 und 8.7 die Entwicklung der $|0\rangle\langle 0|$-Matrixelemente verschieden großer Systeme unter dem Einfluß von Dekohärenz gezeigt. Die Messungen wurden auch hier an Systemen mit 25 und 32 Qubits durchgeführt. Das Auftreten möglicher Dekohärenzeffekte nach jeder Hadamard-Transformation auf jedem Qubit wurde durch eine Wahrscheinlichkeit von $p = 10^{-3}$ (s. Abbildung 8.6) und $p = 10^{-4}$ (s. Abbildung 8.7) vorgegeben.

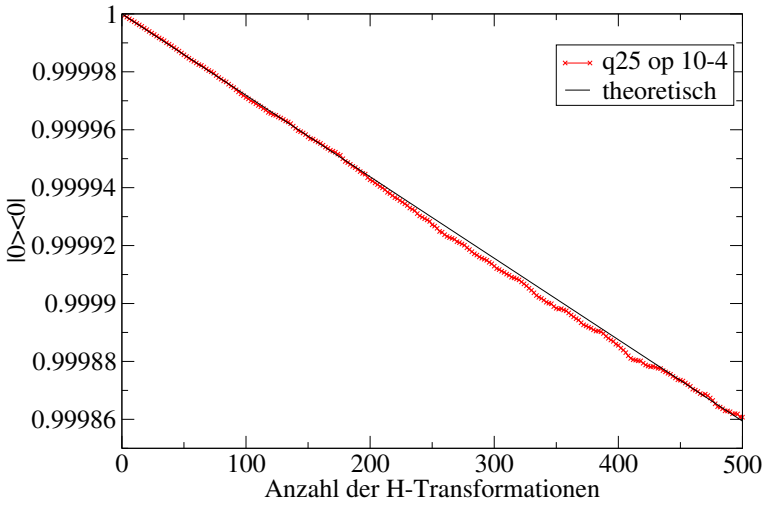

Abbildung 8.5: Entwicklung des $|0\rangle\langle 0|$-Matrixelements für ein 25-Qubit-System für $\sigma = 10^{-4}$ über 500 Hadamard-Transformationen

Zwischen den Ergebnissen aus der Simulation und den theoretischen Kurven gibt es keine zufriedenstellende Übereinstimmung. Dies ist allerdings im Hinblick auf die Art, wie Dekohärenzphänomene hier simuliert werden, erklärbar. Tritt Dekohärenz auf, dann führt dies dazu, daß die Amplitude des Basiszustandes $|...000\rangle$ und damit der entsprechende Eintrag der Dichtematrix sprungartig von 1 auf 0 gesetzt wird. Die Standardabweichung des Erwartungswertes ist daher gewaltig und selbst 100 Programmdurchläufe des 25-Qubit-System reichen nicht aus, eine statistisch aussagekräftige Graphik zu produzieren. Im 32-Qubit-System konnten auf Grund begrenzter Rechenzeit für $p = 10^{-3}$ nur insgesamt 20 und für $p = 10^{-4}$ nur insgesamt 4 Durchläufe simuliert werden. Für ein System mit 25 Qubits zeigen die Werte in beiden Abbildungen, wenn auch stark schwankend, die durch die theoretische Kurve vorgegebene Tendenz. Allerdings sind die Schwankungen zu stark, um die Werte exakt einer theoretischen Kurve zuzuordnen. Bei dem System mit 32 Qubits kann für $p = 10^{-3}$ zwar die Tendenz der theoretischen Kurve erkannt werden, jedoch sind auf Grund der wenigen Meßwerte die Standardabweichungen so groß, das keine Aussage über die Güte der Simulation gemacht werden kann.

Für $p = 10^{-4}$ macht sich die Art der Implementierung der Dekohärenz bemerkbar. In allen vier Simulationsdurchläufen trat kein einziger Fall von Dekohärenz auf, damit blieb der Wert des Matrixelements $|0\rangle\langle 0|$ unverändert bei 1. Trotzdem erkennt man in Abbildung 8.6, daß eine Vergrößerung des Systems zu einem schnelleren Abfall der Amplitude führt.

Abbildung 8.8 demonstriert deutlich, wie ein Anstieg des Wertes p um den Faktor 10 zu einer vier bis fünf Mal kleineren Amplitude des $|0\rangle\langle 0|$-Matrixelements nach 100 Operationen führt. Eine Gegenüberstellung verschiedener Parameter p für ein 32-Qubit-System macht hier auf Grund der oben beschriebenen Meßergebnisse für $p = 10^{-4}$ keinen Sinn. Wie aus Abbildung 8.8 deutlich hervorgeht, handelt es sich nicht um einen linearen Abfall der Amplitude, sondern um einen polynomialen. Diese Entwicklung wird auch

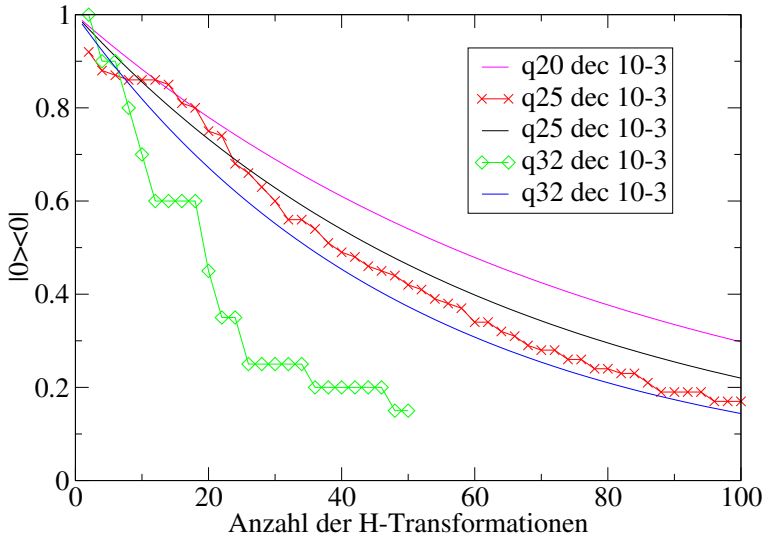

Abbildung 8.6: Entwicklung des $|0\rangle\langle 0|$-Matrixelements mit $p = 10^{-3}$ für verschiedene Systemgrößen

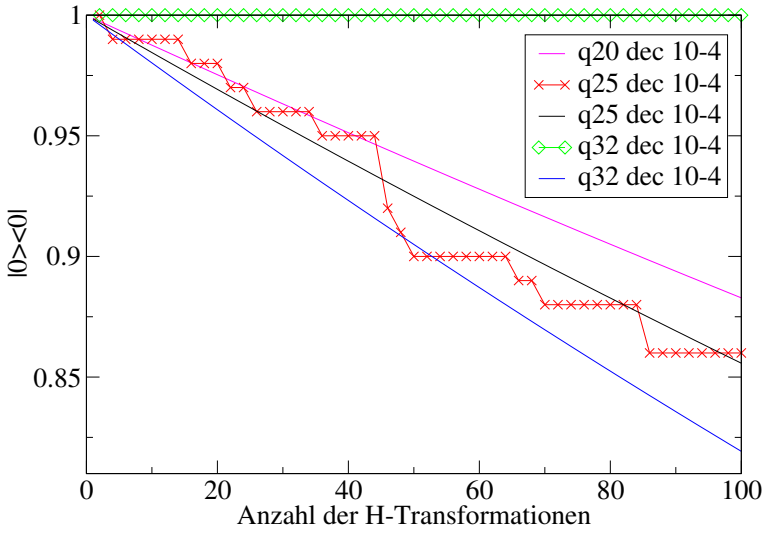

Abbildung 8.7: Entwicklung des $|0\rangle\langle 0|$-Matrixelements mit $p = 10^{-4}$ für verschiedene Systemgrößen

durch Abbildung 8.9 bestätigt, die den Abfall der Amplitude in einem 25-Qubit-System für $p = 10^{-4}$ über 500 Operationsaufrufe hinweg zeigt.

Um genauere Aussagen über das Verhalten der hier untersuchten Systeme unter dem Einfluß von Dekohärenz treffen zu können, wäre eine Erhöhung der Anzahl der Simulationsdurchläufe nötig. Dies ist, wie Simulationen von Systemen mit mehr als 32 Qubits, eine Frage der verfügbaren Rechenzeit, da der Rechenaufwand hierbei enorm ist.

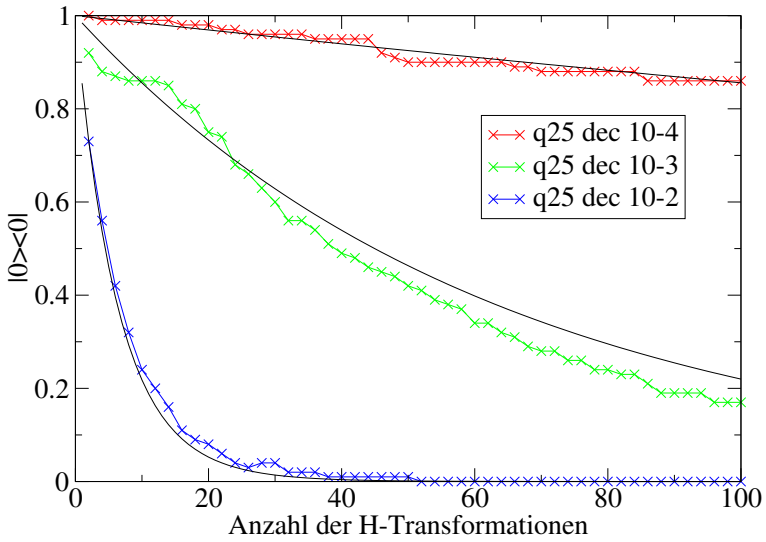

Abbildung 8.8: Entwicklung des $|0\rangle\langle 0|$-Matrixelements eines 25-Qubit-Systems für verschiedene p

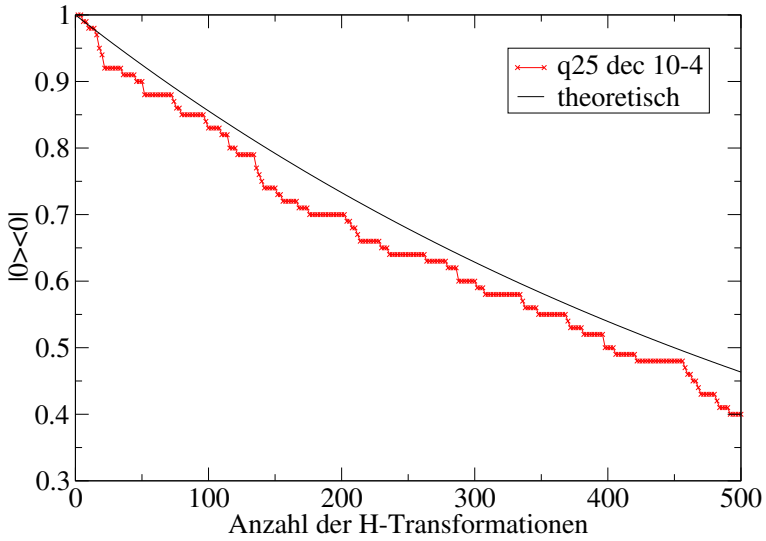

Abbildung 8.9: Entwicklung des $|0\rangle\langle 0|$-Matrixelements für ein 25-Qubit-System für $p = 10^{-4}$ über 500 Hadamard-Transformationen

9 Zusammenfassung

Ein klassischer Computer muß zur Simulation eines Quantencomputers mit n Qubits die Entwicklung der Amplituden des Zustandsvektors des Systems in dem 2^n dimensionalen Hilbertraum berechnen. Da der benötigte Arbeitsspeicher sowie der Rechenaufwand exponentiell mit der Anzahl der simulierten Qubits ansteigen, stellen Simulationen von einigen wenigen Zehn Qubits zwar hohe Anforderungen an heutige Multi-Prozessor-Rechner, können jedoch extrem wichtige Erkenntnisse für das Verständnis der Vorgänge in Quantensystemen liefern. So lassen sich z.B. Voraussagen über die maximale Anzahl durchführbarer Operationen machen, die durch Auswirkungen von ungewollten Effekten auf das System in Form von Informationsverlust limitiert sind. Auch werden Simulationen zur Entwicklung und zum Testen von Fehlerkorrekturmechanismen und neuen Quantenalgorithmen eingesetzt.

Der während dieser Arbeit entstandene Quantencomputersimulator stellt einen universellen Gattersatz, bestehend aus idealen 1- und 2-Qubit-Gattern, zur Verfügung, aus dem in modularer Weise komplexere Algorithmen aufgebaut werden können. Das Programm ist in der Lage, die Wirkung des Algorithmus' auf den Zustandsvektor eines quantenmechanischen Systems zu berechnen. Die Größe des Systems ist dabei durch die Größe des Arbeitsspeichers des klassischen Computers begrenzt. Das größte in dieser Arbeit simulierte System umfaßte 36 Qubits, welches einem Zustandsvektor der Größe ein Terabyte entspricht.

Die Simulationen wurden auf einem IBM Regatta p690 System mit insgesamt 1312 Prozessoren und 5,2 Terabyte Arbeitsspeicher durchgeführt, einem parallelen Höchstleistungsrechner mit einer Mischung aus Shared und Distributed Memory-Architektur. Die Kommunikation und der Datentransfer zwischen den Prozessoren erfolgen durch MPI- und OpenMP-Routinen, die den Einsatz des Programmes auf Clustercomputern mit beliebiger Speicherstruktur möglich macht.

In diesem Zusammenhang war auffallend, daß eine Ansteuerung der Prozessoren durch reines MPI schneller ist als ein Hybridansatz mit MPI und OpenMP, was der intuitiven Annahme widerspricht, daß OpenMP auf einem *Shared-Memory*-System gegenüber MPI schnelleren Zugriff auf die Daten im gemeinsamen Speicher ermöglicht. Im Fall des hier benutzten Computers kann durch das Erhöhen der MPI-Tasks bei gleicher Prozessoranzahl vermieden werden, daß die Threads innerhalb eines Tasks beim parallelen Bearbeiten der Speicherdaten auf stark segmentierte Bereiche zugreifen müssen. Dies kostet mehr Zeit als der zusätzlich benötigte Datentransfer durch Erhöhen der Anzahl der MPI-Tasks.

Auf Grund der jeweilig unterschiedlichen Operationsmatrizen kann es vorkommen,

daß nicht alle Prozessoren in die Rechenoperation involviert werden können. Daher wurde ein Lastausgleich implementiert, der eine Gleichverteilung des Rechenaufwands für alle Prozessoren gewährleistet. Im Fall des CNOT-Gatters wurde der statische Lastausgleich durch eine dynamische Variante entschieden verbessert, die die Verteilung der Daten unter Minimierung der Laufzeit durchführt. Dies führte zu mittleren Laufzeitverkürzungen von bis zu 50% gegenüber dem statischen Fall.

Aus der Analyse der Laufzeiten der 1- und 2-Qubit-Operationen wird ein linearer Anstieg der mittleren Operationsdauern bei zunehmender Systemgröße und einem damit verbundenen Anstieg der Anzahl der MPI-Tasks ersichtlich. Es ist anzunehmen, daß dieser Anstieg für sehr viel größere Systeme als mit 36 Qubits bei gleicher Hardwarestruktur eine Sättigung erreicht, da in diesem Fall die Positionen der Qubits, auf denen eine Operation ausgeführt wird, im Gegensatz zu kleinen Systemen nur minimalen Einfluß auf die Art der Kommunikation haben und so das Kommunikationsaufkommen pro MPI-Task konstant bleibt.

Neben der Simulation idealer Gatter wurde der Einfluß von Dekohärenzeffekten untersucht. Diese wurden in Form eines *depolarizing channel*-Modells implementiert, der einen mit einer festlegbaren Wahrscheinlichkeit eintretenden Bit- und/oder Phase-Flip der Qubits bewirkt. Darüber hinaus wurde am Beispiel der Hadamard-Operation die Auswirkung eines fehlerbehafteten Gatters auf den Zustand des Systems untersucht. Die Hadamard-Operation wurde daher in eine Rotation und eine Phase-Shift-Operation zerlegt, die in Abhängigkeit eines Parameters in ihrer Wirkung modifiziert werden können.

Zur qualitativen Analyse diente die Entwicklung des $|0\rangle\langle 0|$-Elementes der Dichtematrix des Systems unter der Anwendung von bis zu 500 Hadamard-Operationen auf jedem Qubit des Quantencomputers. Die Simulationen wurden auf Systemen mit 25 und 32 Qubits durchgeführt, da ein Kompromiß zwischen möglichst großen Systemen und der begrenzten, zur Verfügung stehenden Simulationszeit gefunden werde mußte. Die Dichtematrix des Systems wurde dabei durch zeitliche Mittelung über die Ergebnisse wiederholter Programmaufrufe berechnet.

Die Auswirkungen der Dekohärenzeffekte unterschiedlicher Stärke für das System mit 25 Qubits entsprechen sehr gut den theoretischen Voraussagen, die Ergebnisse des 32-Qubit-Systems konnten nur der Tendenz nach den theoretischen Werten zugeordnet werden. Dies lag an der sehr begrenzten Anzahl der durchführbaren Messungen bei einen System dieser Größe.

Sehr gute Ergebnisse konnten ebenfalls für das fehlerbehaftete Hadamard-Gatter erzielt werden. Auf Grund der unterschiedlichen Fehlerimplementation haben die Fehlereinwirkungen nicht wie im Fall der Dekohärenz extreme Auswirkungen auf den Zustandsvektor, sondern verursachen eine stetige Abnahme der Amplituden. Bei beiden Systemgrößen zeigt sich zum einen die gute Übereinstimmungen mit der theoretischen Erwartung, zum anderen werden die Unterschiede zwischen verschiedenen Parameterwerten deutlich: Je größer das System ist, desto schneller fällt die Amplitude des Systems mit der Anzahl der durchgeführten Operationen ab. Besonders starken Einfluß auf das Absinken der Amplitude und somit im Endeffekt auf die Anzahl der durchführbaren Operationen hat die Größe der Abweichung vom Idealfall. Hier stellt eine kleine Verringerung der Fehler eine immense Verlangsamung des Informationsverlusts dar.

Das Simulationsprogramm wurde so konzipiert, daß es als Basis genutzt werden kann, um weitere physikalische Effekte einzubeziehen und programmtechnisch zu entwickeln. So bietet es sich an, in einem nächsten Schritt, Fehlerkorrekturschemata zu untersuchen oder die Zeitentwicklung und Steuerung einem realistischen Modell eines Quantencomputers anzupassen.

Literaturverzeichnis

[1] G.E. Moore, 'Cramming more components onto integrated circuits', Electronics 38, pp. 114-117 (1965).

[2] D. Deutsch, 'Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer', Proc. Roy. Soc. Lond. A 400, pp. 97-117 (1985)

[3] R.P. Feynman, 'Simulating physics with computers', Int. J. Theor. Phys. 21, pp. 467-488 (1982).

[4] R.P. Feynman, 'Quantum mechanical computers', Found. Phys. 16, pp. 507-531 (1986).

[5] P. W. Shor, 'Polomial-time algorithms for prime factorization and discrete logarithms on a quantum computer', Proc. 35th Annual Symposium on Foundations of Computer Science, IEEE Press (1994).

[6] L. K. Grover, 'A fast quantum mechanical algorithm for database search', Proc. 28th Annual ACM Symposium on the Theory of Computation, 212-219, ACM Press (1996)

[7] L. K. Grover, 'Quantum mechanics helps in searching the needle in a haystick', Phys. Rev. Lett. 79 , p. 325 (1997)

[8] I. Cirac, P. Zoller, 'Quantum computations with cold trapped ions', Phys. Rev. Lett. 74, p. 4091 (1995).

[9] F. Schmidt-Kaler, H. Häffner, M. Riebe, S. Gulde, G. P. T. Lancaster, T. Deuschle, C. Becher, C. F. Roos, J. Eschner, R. Blatt, 'Realization of the Cirac-Zoller controlled-NOT quantum gate', Nature 422, 408-411 (2003).

[10] D. G. Cory et al, 'Ensemble quantum computing by NMR spectroscopy', Proc. Natl. Acad. Sci. USA 94, pp. 1634-1639 (1997).

[11] N. A. Gershenfeld. I. L. Chuang, 'Bulk spin resonance quantum computation', Science 275, p. 350 (1997).

[12] L. M. K. Vandersypen et al, 'Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance', Nature 414, (2001).

[13] C. H. Bennett, 'Logical reversibility of computation',
IBM J. Res. Develop. 17, pp. 525-532 (1973).

[14] T. Toffoli, 'Reversible Computing',
MIT Laboratory for Computer Science, TM-151 (1980).

[15] A. Church, 'An unsolvable problem of elementary number theory',
Am. J. Math. 58, pp. 345-363 (1936)

[16] A.M. Turing, 'On computable numbers, with an application to the Entscheidungsproblem', Proc. Lond. Math. Soc. Ser. 42, p. 230 (1936)

[17] Y. Ozhigov, 'Quantum Computers cannot speed up iterated applications of a black box', quant-ph/9712051

[18] D. P. Vincenzo, 'The Physical Implementation Of Quantum Computation',
quant-ph/0002077

[19] E. Knill, 'Quantum Computing with Very Noisy Devices',
quant-ph/0410199, (2004)

[20] M. A. Nielson, I. L. Chuang, 'Quantum Computation and Quantum Information',
Cambridge University Press (2000), ISBN 0-521-63503-9

[21] J. Stolze, D. Suter, 'Quantum Computing - A Short Course from Theory to Experiment', Wiley-Vch (2004), ISBN 3-527-40438-4

[22] Los Alamos National Laboratory, 'Los Alamos Science - Information, Science, and Technology in a Quantum World', Nr.27, (2002)

[23] F. Laloë, 'Do we really understand quantum mechanics? Strange correlations, paradoxes and theorems.', quant-ph/0209123, (2004)

[24] `www.top500.org`, (06/2002)

[25] H. De Raedt, K. Michielsen, A. Hans, S. Miyashita, K. Saito, 'Quantum spin dynamics as a model for quantum computer operation', (2002)

[26] N. Pomplun, 'NMR-Quantum Computer Simulation on a parallel Supercomputer', 'Beiträge zum wissenschaftlichen Rechnen - Ergebnisse des Gaststudentenprogramms 2004 des John von Neumann-Instituts für Computing', Esser, R. (Hrsg.), (2004)

[27] J. Niwa, K. Matsumoto, H. Imai, 'General-Purpose Parallel Simulator for Quantum Computing', quant-ph/0201042, (2002)

[28] H. De Raedt, K. Michielsen, 'Computational Methods for Simulating Quantum Computers', (2004)

[29] M. Matsumoto, T. Nishimura, 'Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator', ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, Jan. 1998, pp. 3-30

[30] G. Bhanot, 'The Metropolis Algorithm',
Rep. Prog. Phys. 51, pp. 429-457, (1988)

Danksagung

Nachdem die Arbeit nun hinter mir liegt, möchte ich allen Menschen, die mich bei ihrer Entstehung unterstützt haben, herzlich danken.

Herrn Professor Dr. K.-E. Hellwig gilt mein besonderer Dank für die Stellung des Themas sowie für sein wohlwollendes und förderndes Interesse an dieser Arbeit. In gleicher Weise danke ich Herrn Professor Dr. Dr. Lippert, sowie Herrn Dr. R. Esser für die großzügige Unterstützung dieser Arbeit in Jülich.

Dr. Guido Arnold und Dr. Marcus Richter danke ich für die fachliche Betreuung der Arbeit und die fruchtbaren Diskussionen über die Physik des Quantencomputings.

Ivo Kabadshow gebührt der Dank für alle gelösten Probleme, die mir der Computer und das Betriebssystem in den Weg gestellt haben. Ihm und Michael Hofmann bin ich auch sehr dankbar für die vielen Diskussionen über programmiertechnische Finessen und das wertvolle Wissen auf dem Gebiet der Informatik.

Besonderes bedanken möchte ich mich bei meiner Freundin Stephanie Radecki für die mentale Unterstützung und das Verständnis für die der Arbeit geopferte Zeit, sowie bei meinen Eltern Rosa-Elisabeth und Ekkehard Pomplun, die mich bei meiner Ausbildung mit allen Mitteln unterstützt und auch immer zu Neuem ermuntert haben.

Am Schluß möchte ich noch all denen danken, die die Mittagsrunde sowie die Kaffeepausen und die alltägliche Fahrt von und zur Arbeit zu einer heiteren Angelegenheit haben werden lassen.

Die selbstständige und eigenhändige Anfertigung versichere ich an Eides statt.

Berlin, den 8.11.2005

Nikolas Pomplun