QUANTENCOMPUTING

Von Code zu Maschinensprache

Programme für supraleitende Qubits lassen sich mithilfe spezieller Quantencompiler übersetzen.

Marius Hillenbrand

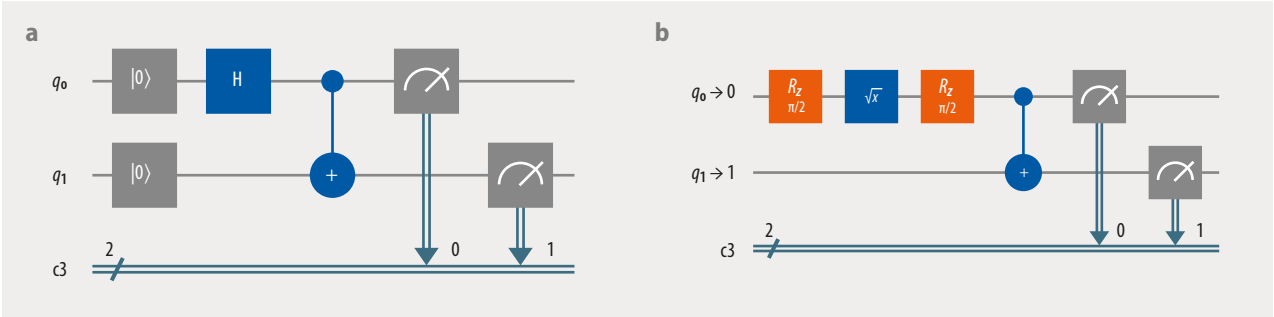


Abb. 1 Mit Qiskit SDK lässt sich ein Quantenprogramm als Abfolge von Quantengattern visualisieren (a) und anschließend mit dem Qiskit Transpiler transformieren, beispielsweise für das IBM-System Lagos (b).

Quantencomputer speichern und verarbeiten Information als Zustände quantenmechanischer Systeme. Bestimmte Probleme könnten sich mit Quantenalgorithmen deutlich effizienter lösen lassen als mit konventionellen digitalen Computern. Damit Quantencomputer als verlässliche Arbeitsgrundlage in Industrie und Forschung zum Einsatz kommen können, müssen sich diese vom Experiment zum kommerziellen Produkt weiterentwickeln. Das erfordert ein interdisziplinäres Team aus Physik, Materialwissenschaft, Elektrotechnik, Informatik und vielen weiteren Gebieten, beispielsweise um die Ansteuerung der Qubits in digitale Systeme zu integrieren. Als wesentlicher Baustein gilt ein Compiler, der einen abstrakten Quantenalgorithmus auf die komplexen Programmabläufe in der Kontrollelektronik des Quantencomputers abbildet.

Ein Compiler für konventionelle Computer übersetzt einen Quellcode aus einer Programmiersprache, die ein Mensch schreiben und lesen kann, schrittweise in binär kodierte Maschinenbefehle. Diese liest der Prozessor iterativ aus seinem Speicher und führt sie aus; der Speicher enthält gleichzeitig die zu verarbeitenden Daten und die auszuführenden Befehle. Über Jahrzehnte haben sich Techniken etabliert, um solche Compiler zu bauen. Im noch jungen Feld der Quantencomputer stellen sich neue Herausforderungen: Die Daten als Quantenzustand und der Programmspeicher liegen räumlich und organisatorisch getrennt vor. Der Quelltext und die resultierenden Maschinenbefehle unterscheiden sich semantisch viel mehr als in einem konventionellen Computer: Der Quantencomputer wird nicht direkt programmiert, sondern mittels eines digitalen Computers, der Qubits über analoge Signale ansteuert. Dieser Artikel beschreibt die Reise eines Programms durch einen Quantencompiler bis zur Ausführung auf einem System auf Basis supraleitender Qubits.

Analogie zur digitalen Welt

Ein Quantenalgorithmus beruht darauf, Information in Zuständen von Qubits zu kodieren und sie zu verarbeiten. Als übliches Modell für Operationen auf Qubits dient das Quantengatter. Angelehnt an digitale Gatter – elementare Schaltungen, die logische Zustände ineinander überführen – repräsentieren Quantengatter Operatoren, die auf Zustände von Qubits wirken. Eine Abfolge solcher Gatter als Repräsentation eines Quantenprogramms heißt auch Quantenschaltung – in Analogie zu den digitalen Schaltungen eines Computers, die sich aus einzelnen Gattern zusammensetzen. Die Messung des Zustands eines Qubits projiziert die Information wieder auf logische Bits.

Die graphische Repräsentation einer Quantenschaltung aus zwei Qubits ähnelt einem Notenblatt (**Abb. 1a**). Jede horizontale Linie beschreibt den zeitlichen Ablauf eines Qubits. Quantengatter entsprechen Elementen auf den Linien: Sie können den Zustand eines Qubits verändern oder mehrere Qubits miteinander verknüpfen. Das Beispiel zeigt, wie sich ein Bell-Zustand, also ein verschränkter Zustand zweier Qubits, durch die Verkettung eines Hadamard-Gatters H und eines Controlled-NOT-Gatters CNOT erzeugen lässt. Das Hadamard-Gatter überführt die Basiszustände $|0\rangle$ und $|1\rangle$ in eine Superposition und entspricht der 2×2-Hadamard-Transformation; CNOT verknüpft zwei Qubits und kann diese verschränken. Dabei dient das erste als Kontroll-

© 2023 Wiley-VCH GmbH

Qubit und das zweite als Ziel-Qubit. Auf die Basiszustände angewandt invertiert CNOT das Ziel-Qubit genau dann, wenn das Kontroll-Qubit im Zustand $|1\rangle$ ist. Anschließend werden die Zustände beider Qubits gemessen und als zwei digitale Bits gespeichert.

Um Quantenprogramme zu entwickeln, dienen Programmiersprachen speziell für Quantencomputer und Software Development Kits (SDKs), die Quantengatter mittels bestehender Programmiersprachen darstellen. Industrie und Forschung haben viele Sprachen und SDKs hervorgebracht; im Folgenden dienen Qiskit SDK und die Sprache OpenQASM 3 zur Veranschaulichung. Qiskit ist ein Open-Source-Toolkit für Quantencomputing auf Basis von Python und unterstützt unter anderem die Repräsentation, Optimierung und Visualisierung von Programmen aus Quantengattern [1]. OpenQASM 3 ist eine imperative Programmiersprache, die sowohl digitale Berechnungen als auch Operationen auf Qubits beschreiben kann und als offener Standard entwickelt wird [2].

Die Quantenschaltung zu **Abb. 1a** übersetzt Qiskit in folgenden Code für OpenQASM 3:

```
OPENQASM 3;
bit[2] c3;
qubit[2] q;
reset q[0];
reset q[1];
h q[0];
cx q[0], q[1];
c3[0] = measure q[0];
c3[1] = measure q[1];
```

Das Programm legt zuerst fest, dass zwei Qubits und zwei Bits verwendet werden. Dann folgen Anweisungen für die Initialisierung der Qubits („reset“), die zwei Gatter sowie die abschließenden Messungen der Qubitzustände.

Ansteuerung supraleitender Qubits

Mikrowellenpulse steuern supraleitende Qubits wie das Transmon-Qubit [3] und lesen sie aus [4]. Um Gatter auf einzelnen Qubits – also Rotationen des Qubitzustands auf der Bloch-Kugel – zu realisieren, gilt es, die Amplitude der Einhüllenden und die Phasenlage eines Pulses passend zu Rotationsachse und -winkel zu wählen. Die Einhüllenden der Mikrowellenpulse werden auf die spezifischen Eigenschaften jedes Qubits kalibriert, um möglichst wenig von idealen Gattern abzuweichen. Um Zwei-Qubit-Gatter abzubilden, müssen zwei Qubits gezielt wechselwirken können. Hierfür verbindet eine Kopplung die Qubits; die Wechselwirkung muss durch Signale steuerbar sein. Die Ansteuerung erfolgt entweder durch Mikrowellenpulse, welche die Qubits zur Wechselwirkung anregen, oder durch gezieltes Beeinflussen der Kopplung, sodass die verbundenen Qubits nur in den gewünschten Intervallen interagieren können.

Im Gegensatz zu digitalen Computern steuern komplexe analoge Signale die Abläufe im Quantencomputer. Um Programme mit beliebigen Abfolgen von Quantengattern auszuführen, muss es möglich sein, beliebige Abfolgen solcher Steuersignale flexibel zu erzeugen. Die Signale müssen außerdem zeitlich eng zwischen vielen Kanälen synchronisierbar sein – zumindest einer pro Qubit und eventuell weitere, um die Kopplungen zu steuern. Speziell konzipierte CPUs mit integrierten Signalgeneratoren können aus Samples der komplexen Einhüllenden in einem Speicher beliebige Signale erzeugen [5, 6]. Diese Art von Signalgeneratoren heißt auch Arbiträrgenerator (Arbitrary Waveform Generator, AWG). Maschinenbefehle dieser CPUs bestimmen Startzeitpunkt, Frequenz und Phasenlage, den Kanal zur Ausgabe eines Steuersignals sowie die gewünschte Einhüllende. Liegt der Startzeitpunkt in der Zukunft, wird ein Befehl in der Warteschlange des angeforderten Kanals gehalten. Das erlaubt überlappende Abläufe, bei denen mehrere Signale zum gleichen Zeitpunkt beginnen: Eine Sequenz von Befehlen plant die Signale rechtzeitig im Voraus ein.

Mit wachsender Zahl der Qubits und höherer Komplexität eines Programms steigt die Anzahl an Steuersignalen, die in jedem Zeitintervall starten, und somit auch die Rate an auszuführenden Befehlen. Diese Rate ist in einer einzelnen CPU begrenzt – und damit auch dieser Ansatz [6]. Um die Größe eines Quantencomputers auf hunderte Qubits und mehr zu skalieren, dient stattdessen ein verteiltes System aus mehreren verbundenen Spezial-CPUs, die zeitlich synchronisiert und koordiniert den Ablauf eines Quantenprogramms steuern [7].

Aufgabe des Compilers

Ein Quantencompiler muss ein Programm als Abfolge abstrakter Quantengatter in die konkreten CPU-Instruktionen der Steuerelektronik eines Quantencomputers umsetzen, sodass diese eine Sequenz von Steuersignalen passend zu den Quantengattern erzeugt. Diese Transformation erfolgt in mehreren Schritten. Jeder davon ersetzt sukzessive die abstrakte Repräsentation durch konkrete Details des Quantencomputers, der das Programm ausführen soll. Der Compiler benötigt also detaillierte Informationen über den Aufbau und die Funktionsweise dieses Zielsystems.

Anders als ein Quantenprogramm, das beliebige Qubits über Zwei-Qubit-Gatter verknüpft, können Transmon-Qubits nur über eine Kopplung wechselwirken. Ein Compiler passt ein Programm zunächst darauf an, welche Qubits des Quantencomputers eine solche Kopplung aufweisen. Dazu bildet er die beliebig nummerierten virtuellen Qubits eines Programms auf die realen Qubits des Zielsystems ab. Verbleiben Verknüpfungen realer Qubits ohne Kopplung, so fügt der Compiler schrittweise Vertauschungen von Qubitzuständen ein, bis nur noch Gatter zwischen gekoppelten Qubits verbleiben. Die Vertauschungen bestehen ihrerseits aus Gattern auf benachbarten Qubits, was die Laufzeit und Fehlerrate des Programms erhöhen kann. Heuristische Algorithmen wie SABRE (SWAP-based BidiRectional heuristic search algorithm) passen ein Programm mit möglichst wenig Vertauschungen an die vorhandenen Kopplungen an [8].

Im nächsten Schritt muss der Compiler die Quantengatter des Programms in die verfügbaren Quantengatter des

© 2023 Wiley-VCH GmbH

Zielsystems zerlegen. Ein Quantencomputer kann nur eine endliche Menge von Gattern direkt ausführen, weil er nicht beliebig viele Steuersignale kalibrieren und vorhalten kann. Eine Menge von Quantengattern heißt universell, wenn sich daraus jedes beliebige andere Quantengatter zusammensetzen lässt [9]. Ein flexibel nutzbarer Quantencomputer braucht also eine universelle Menge von Quantengattern.

In Qiskit übernimmt der Transpiler die Abbildung auf eine konkrete Topologie und die verfügbaren Gatter eines Zielsystems (**Abb. 1b**). Ein Transpiler ist ein spezieller Compiler, dessen Ausgabe das gleiche Format wie die Eingabe besitzt. Die Operationen, welche die Qubits initialisieren entfallen, weil die Ausführung implizit im Grundzustand $|0\rangle$ beginnt. Eine äquivalente Folge dreier verfügbarer Gatter ersetzt das Hadamard-Gatter: Rotationen um jeweils 90° um die z-Achse (R_z) und die x-Achse ($\sqrt{x}$) der Bloch-Kugel. Weil die Qubits 0 und 1 gekoppelt sind, lässt sich das CNOT-Gatter direkt anwenden.

Nun sind in dem Programm nur noch Gatter vorhanden, die sich durch Steuersignale auf dem Zielsystem realisieren lassen. Doch das Programm ist nach wie vor eine Folge abstrakter Operationen. Daher identifiziert der Compiler nun für jedes der Gatter die passenden Steuersignale und gelangt somit vom Gattermodell in die Domäne realer Qubits, deren Zustand über Signale verändert wird. Welche Signale das sind, hängt von der technischen Umsetzung des Systems ab: Der Compiler benötigt eine Datenstruktur, die für jedes Gatter pro Qubit die Steuersignale, ihre Dauer, Frequenz, Phasenlage und Kanal aufführt. Mit dieser Struktur überführt der Compiler die Folge von Gattern in einen zeitlichen Ablaufplan von Steuersignalen. Mit Qiskit Pulse [10], einem Teil von Qiskit SDK, lassen sich Ablaufpläne dieser Art visualisieren und nachverfolgen.

Für Systeme mit vielen Qubits ist die Kontrollelektronik über mehrere CPUs verteilt, die jeweils einen Teil der Signalgeneratoren und Kanäle ansteuern. Der Compiler teilt den Ablaufplan auf und erzeugt für jede CPU einen lokalen Ablaufplan über die Signale auf ihren angeschlossenen Kanälen. Diese lokalen Ablaufpläne wandelt der Compiler in Folgen von Maschinenbefehlen um, welche die integrierten Signalgeneratoren passend für die geplanten Signale ansteuern. Während ein klassischer Compiler zumeist ein Eingabe- in ein Ausgabeprogramm transformiert, erzeugt ein Quantencompiler ein Ausgabeprogramm pro CPU der Kontrollelektronik. Für einen korrekten Ablauf müssen alle CPUs synchron arbeiten, wozu der Compiler einen Code erzeugt, der die CPUs koordiniert [7]. Zuletzt werden die generierten Programme in die CPUs der Kontrollelektronik übergeben und ausgeführt; die Resultate der Qubitmessungen entsprechen der Ausgabe. Hier endet die beispielhafte Reise eines Programms durch den Compiler bis zur Ausführung auf einem Quantencomputer.

Weitere Facetten

Compiler bleiben aber nicht auf Transmon-Qubits beschränkt, sondern werden auch für weitere Typen von Qubits benötigt. Diese unterscheiden sich in der Art der Ansteuerung, sodass sie andere Schritte und Verfahren im Compiler erfordern. Manche Teilschritte von Quantenalgorithmen lassen sich effizienter umsetzen, wenn Qubitmessungen zur Laufzeit des Programms über Bedingungen (if/else) und Schleifen die Ausführung von Quantenoperationen beeinflussen können [11], also der Kontrollfluss im Programm und die Quantengatter zusammenwirken. Hierzu bedarf es einer geeigneten Kontrollelektronik und der Koordination der komplexen Interaktionen durch den Compiler [7]. Um leistungsfähigere Quantencomputer zu entwickeln, gilt es, die Anzahl der Qubits zu steigern und die Fehlerraten der Operationen zu verbessern. Um präzisere Resultate zu erhalten, ist es beispielsweise nötig, Störeinflüsse bei der Adaption von Quantenprogrammen gezielt zu kompensieren: Die Quantenfehlerkorrektur, welche die Information eines logischen Qubits in mehrere physikalische kodiert, kann fehlerbehaftete Qubitzustände zur Laufzeit erkennen und korrigieren [12]. Daraus ergeben sich interessante Fragestellungen für Quantencompiler, die weitere Entwicklungen in diesem Gebiet anstoßen werden.

Literatur

[1] Qiskit contributors. Qiskit: An Open-source Framework for Quantum Computing. https://qiskit.org (2023)
[2] *A. Cross* et al., ACM Transactions on Quantum Computing **3**(3), 12 (2022)
[3] *J. Koch* et al., Phys. Rev. A **76**, 042319 (2007)
[4] *P. Krantz* et al., Appl. Phys. Rev. **6**, 021318 (2019)
[5] *X. Fu* et al., Proc. 50th Annual IEEE/ACM Int. Symposium on Microarchitecture (MICRO-50 '17), 813 (2017)
[6] *A. Butko* et al., Proc. 2020 International Conference on Rebooting Computing (ICRC), 66 (2020)
[7] *G. Zettles* et al., Proc. 2022 IEEE International Solid-State Circuits Conference (ISSCC), 1 (2022)
[8] *G. Li, Y. Ding* und *Y. Xie*, Proc. Twenty-Forth Int. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), 1001 (2019)
[9] *A. Barenco* et al., Phys. Rev. A **52**, 5 (1995)
[10] *T. Alexander* et al., Quantum Sci. Technol. **5**, 044006 (2020)
[11] *A. D. Córcoles* et al., Phys. Rev. Lett. **127**, 100501 (2021)
[12] *S. Bravyi* et al., J. Appl. Phys. **132**, 160902 (2022)

Der Autor



Marius Hillenbrand hat Informatik am Karlsruher Institut für Technologie studiert und sich auf Systemsoftware und ihre Optimierung spezialisiert, beispielsweise Betriebssysteme, Virtualisierung und Compiler. Bei IBM Deutschland forscht und arbeitet er als Teil eines interdisziplinären Teams an Systemsoftware für Quantencomputer.

Marius Hillenbrand, IBM Deutschland Research & Development GmbH, Schönaicher Str. 220, 71032 Böblingen

© 2023 Wiley-VCH GmbH