Entwickeln mit Microsofts Azure Quantum

Noch sind Quantencomputer rar. Wer Erfahrungen mit ihrer Programmierung sammeln will, kann Angebote wie Microsofts Quantum Development Kit nutzen: Es bietet Quantensimulatoren und Zugang zu echter Hardware.

Von Kay Glahn

Bereits im Dezember 2017 veröffentlichte Microsoft Q# als Bestandteil seines Quantum Development Kit (QDK). Seit im Februar 2021 mit Azure Quantum die erste öffentliche Preview eines Public-Cloud-Ökosystems für Quantencomputing verfügbar war, hat Microsoft im Monatsrhythmus neue Releases nachgelegt und dadurch kontinuierlich neue Funktionen und die Unterstützung für aktuelle Entwicklungsumgebungen bereitgestellt. Das aktuelle Release setzt auf .NET 6 statt .NET Core 3.1 auf und unterstützt auch Visual Studio 2022 als Entwicklungsumgebung. Das QDK soll Entwicklern ermöglichen, Quantencomputersoftware nur einmal zu schreiben und dann mit minimalen Anpassungen auf unterschiedlichen Systemen auszuführen.

Derzeit bietet Azure Quantum Zugriff auf Ionenfallenrechner von IonQ und Quantinuum. Letzteres Unternehmen ist Ende 2021 aus dem Zusammenschluss von Honeywell Quantum Solutions und Cambridge Quantum hervorgegangen. IonQ stellt einen Quantencomputer mit 11 Qubits auf echter Hardware und einen GPU-beschleunigten Simulator mit 29 Qubits zur Verfügung (siehe Abbildung 1), von Quantinuum stehen die physischen Modelle H1-1 mit 20 und H1-2 mit 12 Qubits bereit, die beide auf Quantencomputer-Hardware von Honeywell basieren. Die Integration von weiteren Quantencomputern der Partner Pasqal, Rigetti und Quantum Circuits (QCI) ist geplant. Diese stehen bereits im Rahmen einer Private Preview zur Nutzung in Azure Quantum zur Verfügung.

Q# ist Microsofts quelloffene Programmiersprache zum Entwickeln und Ausführen von Quantenalgorithmen (siehe Artikel „Einstieg in Microsofts Quantensprache Q#“ in iX Developer 2022). Sie gehört zum QDK, das neben den Q#-Bibliotheken einen Simulator, Erweiterungen für andere Programmiersprachen sowie die API-Dokumentation enthält. Hinzu kommen Bibliotheken für quantenbasierte Berechnungen aus Chemie, Machine Learning und Numerik.

iX-TRACT

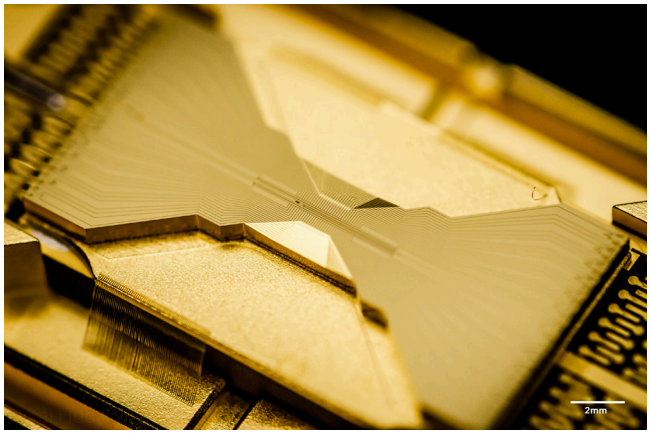
- In Azure Quantum stehen sechs Quantensimulatoren zur Verfügung, die sich in der Zahl der maximalen Qubits und der zulässigen Operationen unterscheiden.
- Auch auf echten Quantenrechnern können Entwickler in Azure Quantum Programme laufen lassen. Über die Kosten sollte man sich gründlich informieren.
- Außerdem stellt Azure Quantum Optimierungsalgorithmen fürs Annealing bereit, die keine Quantenhardware erfordern.
- Mit QIR hat Microsoft eine Darstellung für Quantenprogramme entwickelt, die unabhängig von der benutzten Programmiersprache und Plattform ist.

Quanten- und traditionelle Programmierung

Zwar ist Q# auf Quantenalgorithmen ausgelegt. Es übernimmt jedoch viele Elemente aus Python, C# sowie F# und unterstützt ein einfaches prozedurales Modell mit Schleifen, `if`-Anweisungen und den üblichen Datentypen. Mit dem QDK lassen sich bekannte Entwicklungstools und Sprachen nutzen und Programme in unterschiedlichen Umgebungen ausführen. Sie können entweder als Konsolenapplikationen im Terminal oder in einem Python- oder .NET-Hostprogramm laufen. Hierfür sind C# und F# als Hostsprache verwendbar.

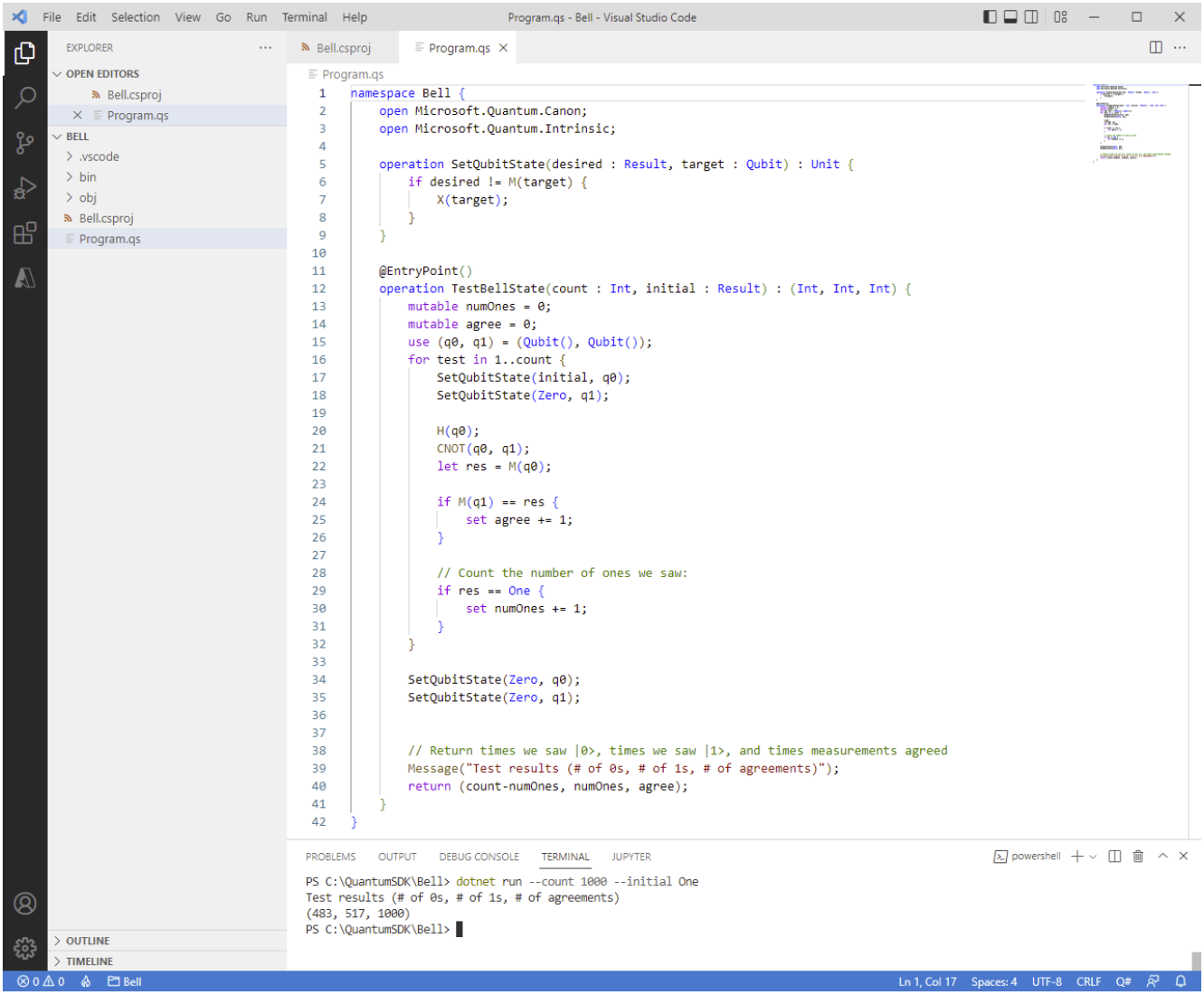
Außerdem lassen sich Q#-Programme auf Jupyter-Notebooks ausführen. Am einfachsten geht dies mit gehosteten Jupyter-Notebooks direkt im Azure-Quantum-Arbeitsbereich. Alternativ kann man wie bisher entweder in Binder auf eine Q#-fähige Jupyter-Instanz in der Cloud zurückgreifen oder die Jupyter-Umgebung lokal installieren. Hierfür steht neben einem Docker-Image des QDK auch die Installation via Conda zur Verfügung. Das QDK lässt sich mit Visual Studio und Visual Studio Code nutzen. Wer das Ganze online betreiben möchte, kann Visual Studio Codespaces verwenden – einen entsprechenden Azure-Tarif vorausgesetzt. Zum Ausprobieren neuer Funktionen oder zum schnellen Bauen eines Prototyps empfehlen sich wiederum Jupyter-Notebooks.

Quelle: IonQ



IonQ bietet als Partner von Microsoft Ionenfallensysteme mit 11 Qubits in Azure Quantum an (Abb. 1).

Für Optimierungsaufgaben verwendet man einen der Quantum Solver, den man entweder über ein Jupyter-Notebook im Quantum Workspace, ein lokales Jupyter-Notebook oder das vom QDK bereitgestellte Python-SDK anspricht. Bei letzterer Variante installiert man das Paket azure-quantum aus PyPI und setzt ein Jupyter-Notebook lokal auf. Der nötige QIO-Provider (Quantum-Inspired Optimization) wird beim Anlegen des Quantum Workspace in der Standardkonfiguration automatisch mit erstellt. Danach kann man Optimierungsjobs in Python an ein Target in der Cloud übermitteln.



Das Microsoft Azure Quantum Development Kit lässt sich mit Visual Studio Code nutzen (Abb. 2).

Das QDK bietet sechs verschiedene Simulatoren für Q#-Programme, die auf klassischen Computern laufen. Mit ihnen lassen sich Quantenprogramme ausführen und testen, um zu sehen, wie die Qubits auf verschiedene Operationen reagieren. Zudem lässt sich damit ein Algorithmus ausprobieren und debuggen, bevor man ihn auf echter Quantenhardware ausführt. Dabei tauscht man lediglich die darunterliegende Implementierung aus, ohne den jeweiligen Algorithmus zu ändern.

Die Simulatoren implementieren grundlegende Operationen in Klassen. Hierzu gehören primitive Quantenoperationen wie H (Hadamard-Gate), cNOT (controlled NOT-Gate) und Measure. Weitere Klassen von Quantenmaschinen sollen künftig andere Simulationen oder die Ausführung auf echter Quantenhardware erlauben.

Simulatoren mit unterschiedlichen Fähigkeiten

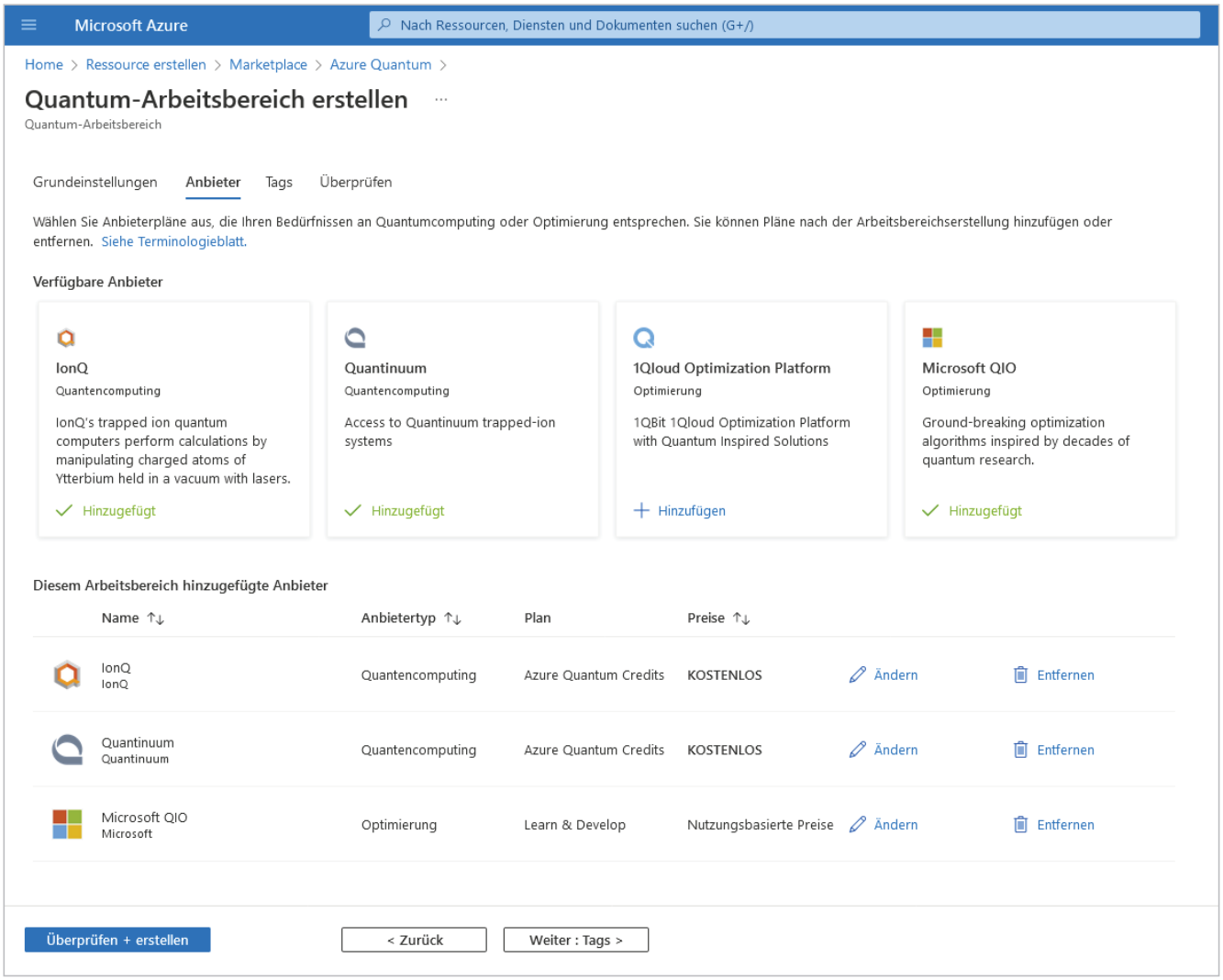
Der Full State Simulator bildet einen vollständigen Quantencomputer auf der lokalen Hardware nach. Er akzeptiert Algorithmen, die maximal 30 Qubits nutzen. Dieser Simulator ähnelt in seiner Funktionalität dem, den die LIQUi|⟩-Plattform von Microsoft Research verwendet.

Neu ist der zweite durch das QDK bereitgestellte Simulator, der Sparse Simulator, der im Gegensatz zum Full State Simulator eine dünnbesetzte Darstellung der Quantenzustandsvektoren verwendet. Das minimiert den zur Darstellung der Quantenzustände erforderlichen Speicherbedarf, was eine Simulation von Algorithmen mit bis zu 1024 Qubits ermöglicht. Standardmäßig ist die Anzahl der verfügbaren Qubits auf 64 eingestellt. Diesen Simulator zu verwenden ist sinnvoll, wenn bei einem Algorithmus viele Quantenzustände auftreten, bei denen die Mehrzahl der Amplitudenkoeffizienten in der Berechnungsbasis null ist.

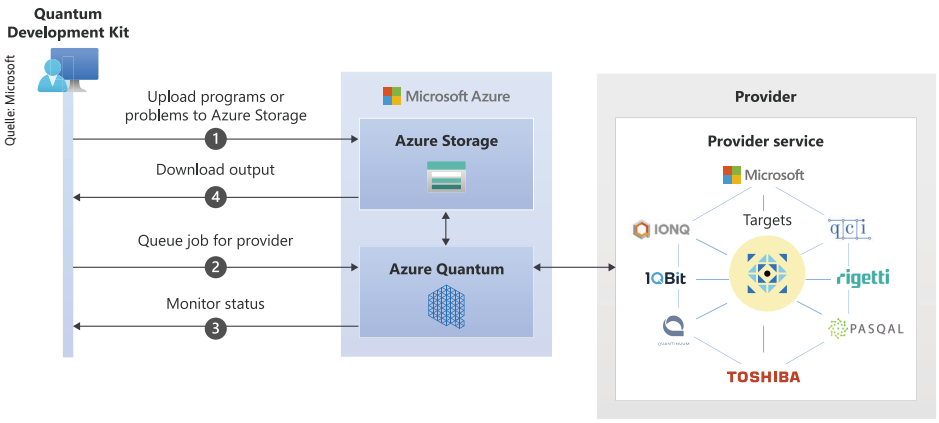
Der Resources Estimator, der dritte Simulator, führt das Programm nicht aus, sondern schätzt stattdessen die Ressourcen, die es auf einem physischen Quantencomputer benötigen würde. Dazu führt er die Operationen durch, ohne jedoch den Zustand der Quantenhardware zu simulieren. Das funktioniert anstandslos für Programme, die Tausende von Qubits verwenden.

Der Resources Estimator nutzt dafür den vierten, den Quantum Trace Simulator, der weitere Metriken und Tools für das Debuggen bereitstellt. Er eignet sich zum einen für die Fehlersuche in klassischem Code, der Teil eines Quantenprogramms ist, und zum anderen für das detailliertere Abschätzen erforderlicher Ressourcen.

Fünfter im Bunde ist der Toffoli-Simulator. Er lässt sich nur für einige Quantenprogramme verwenden, da er nur Pauli-X-, cNOT- und Multi-controlled-X-Operationen beherrscht. Ope-



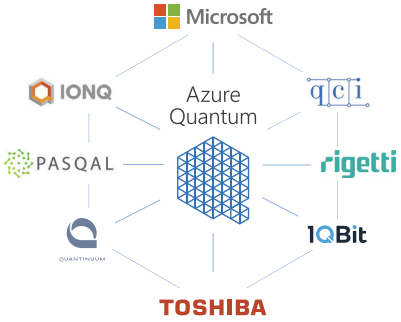
In der erweiterten Erstellung eines Azure-Quantum-Arbeitsbereichs lassen sich verschiedene Provider individuell hinzufügen (Abb. 3).



rationen wie H, die ein Qubit in eine Superposition bringen, kennt dieser Simulator nicht. Dafür ist er nicht auf 30 Qubits beschränkt: Standardmäßig alloziert er Speicher für 65 536 Qubits, kann aber auch mit Millionen davon rechnen. Er eignet sich vor allem für Orakel, die boolesche Funktionen auswerten, da sie sich mit dem beschränkten Satz an Operationen auf vielen Qubits testen lassen. Der Toffoli-Simulator arbeitet nicht mit Zufallswerten: Da er keine Superpositionen von Qubits nachbildet, ist die Wahrscheinlichkeit einer Messung in jedem Fall 0 oder 1 und kann nicht wie normalerweise bei der Messung eines Quantensystems auch dazwischen liegen.

Schließlich steht seit Kurzem noch der Noise Simulator als Preview Simulator bereit. Er kann das Rauschen eines offenen Quantensystems simulieren, wie es in der Praxis bei Quantencomputern auftritt, da diese nicht hundertprozentig von der Umgebung abgeschirmt werden können. Dieser Simulator ist somit in der Lage, Q#-Programme unter dem Einfluss von simuliertem Umgebungsrauschen auszuführen. Außerdem können Quantenalgorithmen hiermit die Stabilizer Representation (auch als CHP-Simulation bekannt) verwenden, was bedeutet, dass die Algorithmen ausschließlich aus CNOT-, Hadamard- und Phase-Gates bestehen. Zu beachten ist, dass die Preview dieses Simulators zwar mit Python- und C#-Hostprogrammen sowie mit Stand-alone-Q#-Notebooks, nicht jedoch mit Q#-Kommandozeilenprogrammen und QIR-basierten Executables verwendbar ist.

Für den Zugriff auf die Quantencomputing-Provider und -Targets in Azu Quantum nutzt man Microsofts QDK (Abb. 4).



Um Azure Quantum zu nutzen, fügt man dem Abonnement im Portal eine Quantum-Workspace-Ressource hinzu. Dabei erhalten neue Benutzer ein kostenloses Guthaben zur Nutzung von Quantenhardware. Für den schnellen Einstieg bietet Microsoft inzwischen die Schnellerstellung an. Ansonsten kann man über die erweiterte Erstellung seinen Quantum Workspace individuell anpassen und auch Optimierungsanbieter direkt hinzufügen. Unter ihr lassen sich die von Applikationen benötigten Assets anlegen. Obligatorisch ist hierfür ein Storage Account, der Speicherplatz in der Cloud bereitstellt. Dort lassen sich die Quantenprogramme und die Optimierungsaufgaben ablegen, damit die jeweiligen Dienste sie verarbeiten können. Außerdem landen hier die Ergebnisse der Berechnungen.

Als Nächstes benötigt man innerhalb des Quantum Workspace einen Provider, auf dem die Programme laufen sollen. Er stellt ein oder mehrere Targets bereit – entweder echte oder simulierte Hardware. Standardmäßig bekommt der Workspace den QIO-Provider, andere lassen sich jederzeit hinzufügen. Für jeden muss man einen Billing Plan auswählen; die genaue Tarifierung bleibt dem Provider überlassen.

Jobs erstellen und ausführen lassen

Wer auf Azure Quantum ein Programm ausführen oder eine Optimierungsaufgabe berechnen will, muss einen Job erstellen und zur Ausführung an ein Target übermitteln. Dabei lädt ihn Quantum auf den im Workspace konfigurierten Storage Account, reiht ihn in die Warteschlange des Providers ein und übersetzt das Programm für den Provider. Ist der Job an der Reihe, führt ihn der Provider aus und legt das Ergebnis zum Herunterladen im Storage ab. Dazu dient ein Link, der auf den Azure Storage verweist. Wie man auf diese Daten zugreift und sie weiterverarbeitet, hängt vom SDK oder dem Werkzeug ab, das den Job an das Target übermittelte.

Neben den physischen oder simulierten Quantencomputern in der Cloud können Targets auch Solver für Optimierungsaufgaben sein. Targets unterliegen unterschiedlichen Einschränkungen. Deshalb sind sie in Profile eingeteilt, die von der Art der QPU (Quantum Processing Unit) abhängen. Da die Quantenhardware noch in den Kinderschuhen steckt, kann nicht jede Hardware jedes Q#-Programm ausführen. Diese Einschränkungen sind beim Programmieren zu berücksichtigen.

Zurzeit existieren theoretisch drei QPU-Profile: Das leistungsfähigste heißt Full und könnte jedes Q#-Programm aus-

Target-Profile in Azure Quantum

Target-Profil	Beschreibung	verfügbare Targets/Provider
Full	bis auf Speicher und Qubits keine Beschränkung	Full State Simulator oder Resources Estimator des Microsoft QDK
No Control Flow	Messungen dürfen keinen Einfluss auf Programmfluss haben. Zusätzlich in Preview: keine Operationen auf Qubits, deren Zustand gemessen wurde	IonQ-QPU und -Simulator
Basic Measurement Feedback	eingeschränkte Nutzung der Messergebnisse zur Steuerung des Programmflusses	Quantinuum System Model H1-1 und H1-2 QPUs sowie System Model H1 Emulator und Syntax Checker

führen. Bei simulierter Hardware bildet lediglich der verfügbare Speicher eine Einschränkung und bei echter die Anzahl der Qubits. Bislang stellt Azure Quantum jedoch kein Target mit diesem Profil bereit. Die einzige Möglichkeit, den vollen Funktionsumfang von Q# auszunutzen, ist, ein Programm im Full State Simulator oder dem Resources Estimator des QDK lokal auszuführen. Das zweite Quantum-Profil ist No Control Flow. Ihm zugeordnete Targets können alle Q#-Programme ausführen, die für den Programmfluss nicht auf das Messen eines Qubits angewiesen sind. Sie dürfen deshalb Werte vom Typ `Result` nicht für Tests auf Gleichheit verwenden, da dies den Programmfluss beeinflussen würde.

Qubits messen oder nicht messen

Targets mit diesem Profil unterliegen in der aktuellen Preview einer weiteren Einschränkung: Operationen dürfen keine Qubits verwenden, deren Zustand bereits gemessen wurde – nicht einmal, wenn das Messergebnis den Programmfluss nicht bestimmt. Bislang stehen mit IonQ lediglich ein Provider und mit der IonQ-QPU sowie dem IonQ-Simulator zwei Targets mit No-Control-Flow-Profil zur Verfügung.

Das dritte Profil nennt sich Basic Measurement Feedback. Es erlaubt das eingeschränkte Nutzen der Messergebnisse von Qubits zum Steuern des Programmflusses. `if`-Anweisungen dürfen nur Werte des Datentyps `Result` vergleichen, und die dazugehörigen Codeblöcke dürfen keine `return`- und `set`-Anweisungen enthalten. In Quantum bieten zurzeit Quantinuums System Model H1 Emulator und der Syntax Checker sowie dessen Modelle H1-1 und H1-2 dieses Profil an (siehe Tabelle „Target-Profile in Azure Quantum“). Wer bisher die inzwischen als deprecated gekennzeichneten Targets HØ und H1 des Providers Honeywell verwendet hat, kann diese zwar vorerst noch weiterhin nutzen, sollte sie aber auf die entsprechenden Targets H1-1 oder H1-2 von Quantinuum migrieren. Hat man das passende Target bestimmt, ist das Programm nach Azure Quantum zu übertragen und der Job in die Warteschlange zu stellen. Dazu stehen das Azure CLI (Command Line Interface), das Jupyter-Notebook oder Python-Code zur Auswahl. Ist bereits ein Quantum Workspace eingerichtet, ist die Nutzung des Azure CLI oder des im Quantum Workspace integrierten Jupyter-Notebook die einfachste Variante. Mit dem CLI kann man zwar auch einen Workspace anlegen, das geht aber im Webportal wesentlich einfacher. Für das Übertragen von der Kommandozeile müssen lokal das QDK, das Azure CLI und die Azure CLI Quantum Extension installiert sein. Letztere lässt sich im Terminal neu einrichten oder ein Upgrade durchführen, falls bereits eine ältere Version installiert ist:

```
az extension add --upgrade -n quantum
```

Danach meldet man sich bei Azure mit `az login` an und legt den Standard-Workspace fest:

```
az quantum workspace set -l westus -g quantum-test -w
quantum-test-workspace
```

Hierbei gibt `-l` die Location an und `-g` die Resource Group. Die im Workspace verfügbaren Targets listet

```
az quantum target list -o table
```

Um den Job an Quantum zu übermitteln, stellt man ihn vom Programmverzeichnis aus in die Warteschlange:

```
az quantum job submit --target-id TestProvider.TestTarget
```

Die Antwort enthält Informationen zum eingestellten Job inklusive der Job-ID. Mit

```
az quantum job output -j yyyyyyyy-… -o table
```

zeigt man das Ergebnis an. Dabei gibt `-j` die gerade erfragte Job-ID an.

```
Result Frequency
-------- -------
[0,0] 0.50000000
[0,1] 0.50000000
```

Alternativ lässt sich der Status eines Jobs im Azure Portal einsehen.

Mit Quanten das Optimum finden

Mit fortschreitendem Verständnis der Quantenmechanik zeigte sich, dass Quantenalgorithmen für Optimierungsverfahren wie Simulated Annealing, Parallel Tempering Monte Carlo oder genetische Algorithmen wesentlich besser geeignet sind als klassische. Allerdings bildet man sie heute eher auf klassischen Computern nach. Sobald aber Quantencomputer mit ausreichend hoher Zahl verschränkter Qubits und geringen Fehlerraten zur Verfügung stehen, dürften dieselben Algorithmen darauf deutlich schneller zum Ziel kommen.

Neben generischen Quantencomputern stellt Microsoft mit Azure Quantum auch Optimierungsalgorithmen bereit. Da diese QIO-Algorithmen ohne Quantenhardware auskommen, können Entwickler die Vorteile dieser Algorithmen nutzen, ohne auf ausreichend leistungsfähige Hardware warten zu müssen. Die QIO-Algorithmen laufen zurzeit auf CPUs, FPGAs, GPUs, Hardware-Annealern oder anderen Spezialchips. Sie sollen aber auch auf zukünftiger Quantenhardware arbeiten. Außerdem soll man Dienste von Partnern wie 1Qloud, Quantum Circuits (QCI) oder Toshiba SQBM+ hinzufügen können. Bislang ist dies allerdings nur für 1Qloud-Services möglich. Die Dienste von Toshiba SQBM+ sind aktuell nur für Kunden verfügbar, die ihr Azure-Abrechnungskonto in den USA, dem Vereinigten Königreich, Kanada oder Japan haben.

Ein weiterer Baustein von Microsofts Azure-Quantum-Ökosystem ist die Quantum Intermediate Representation (QIR). Diese sprach- und hardwareunabhängige Darstellung von Quantenprogrammen entspringt der freien LLVM. QIR definiert, wie LLVM Quantenkonstrukte darstellt, ohne dass sie ergänzt oder geändert werden müsste. Microsoft möchte damit ein einheitliches Beschreibungsformat als Schnittstelle zwischen verschiedenen Sprachen und Quantenplattformen schaffen und hat die bestehende Arbeit jetzt in die neu gegründete QIR Alliance eingebracht, die sich unter dem Dach der Linux Foundation befindet.

Von Sprache und Hardware abstrahieren

QIR kann jede gatterbasierte Quantenprogrammiersprache und damit auch Q# darstellen. Dabei ist es hardwareunabhängig und spezifiziert keine bestimmten Quantenbefehlssätze oder Schaltkreise, sondern überlässt dies der Zielplattform. Beispielsweise kann der Compiler Clang ein in QIR dargestelltes Programm für einen klassischen Computer kompilieren, um einen Simulator

Listing 1: Beispielprogramm Bell-Zustand in Q#

```
operation BellPair(x0 : Qubit, x1 : Qubit) : Unit
{
H(x0);
CNOT(x0, x1);
}
```

für Quantencomputer in C oder C++ zu erstellen. Eine andere Verwendungsmöglichkeit von QIR sind Programme, die Quantenalgorithmen optimieren. Das macht die Optimierung unabhängig von Programmiersprache und Hardware.

Mit einem einfachen Beispiel illustriert Microsoft, wie QIR einen kurzen Q#-Code darstellt. Es stellt durch Anwenden des Hadamard-Gates und des CNOT-Gates auf zwei Qubits den Bell-Zustand her (Listing 1). Nach der Umwandlung des Q#-Codes sieht der QIR-Code aus wie in Listing 2. Da nicht jede verfügbare Hardware den vollen Funktionsumfang eines beliebigen Quantenprogramms bereitstellt, gibt es auch bei der QIR Profile. Sie definieren abgeschlossene Mengen von Funktionen, die die Zielumgebung bieten muss.

Für Azure Quantum und seine Tools stellt Microsoft Dokumentation, Lerninhalte samt Beispielprogrammen und Kurse zur Verfügung, die Entwicklern den Einstieg in die Quantenwelt erleichtern sollen. Da sich viele Quantenphänomene nur schwer veranschaulichen lassen, beschreibt Microsoft die Zusammenhänge des Quantencomputing mit mathematischen Methoden. Da aber nicht jeder mit linearer Algebra, insbesondere mit Vektoroperationen, und imaginären Zahlen vertraut sein dürfte, führen die Dokumentationen auch in diese Themen ein.

Per Onlinekurs zum Quantenentwickler

Auf der Plattform Microsoft Learn stehen Lernpfade und Module zum Thema Quantencomputer, Q# und Azure Quantum zur Verfügung, und mit Quantum Katas existiert ein separater Kurs als Open Source auf GitHub. Er besteht aus aufeinander aufbauenden Tutorials und Programmierübungen. Man beginnt mit einer einzigen Zeile und endet bei komplexen selbst zu schreibenden Programmen. Das Ergebnis prüft ein auf Unit-Tests basierendes Framework. Beim Lösen der Aufgaben helfen Links zu Referenzmaterial, Musterlösungen und detaillierte Erklärungen zu den Aufgaben. Für die auf Azure Quantum angebotenen Dienste zahlt man nach Rechenleistung, Zeit oder per Abo. Die Kosten für den obligatorischen Storage Account richten sich meist nach belegtem Platz, und ein Q#-Programm benötigt nicht viel. Verwendet man Azure Quantum zum ersten Mal, erhält man beim Anlegen eines Quantum Workspace einmalig automatisch Azure Quantum Credits im Wert von 500 US-Dollar für jeden teilnehmenden Anbieter, die für die Nutzung von Hardware verwendet werden können. Erstellt man weitere Quantum Workspaces, dann teilen sich diese das vorhandene Kontingent an Quantum Credits. Ist dieses aufgebraucht, muss man einen separaten Tarif für die jeweilige Hardware abschließen. Alternativ kann man auch versuchen, einen Antrag bei dem Microsoft-Azure-Quantum-Credits-Programm zu stellen, bei dem Microsoft unter Umständen bis zu 10 000 US-Dollar an zusätzlichen Quantum Credits gewährt, wenn man begründen kann, wofür man diese benötigt.

Wer auf den Quantencomputern von IonQ rechnen will, zahlt für die verwendeten Quantengatter, und zwar pro Durchlauf und Gate, also pro Gate Shot. Da man auf Quantencomputern für ein statistisch relevantes Ergebnis viele Durchläufe benötigt, summieren sich hier die Kosten, bleiben aber überschaubar. Pro Shot kosten Ein-Quanten-Gates 0,00002 Euro und Zwei-Quanten-Gates 0,00025 Euro. Für Programme mit mehr als zwei Qubits rechnet IonQ 6 × (n-2) Zwei-Qubit-Gates pro Shot ab, wobei n der Qubitzahl der Gates entspricht. Ein NOT-Gate mit drei Control-Qubits entspräche beispielsweise 12 Zwei-Qubit-Gates. Pro Programmausführung fällt zudem mindestens ein US-Dollar an. Für die IonQ-Hardware schätzt Microsoft Kosten von 3000 US-Dollar pro Stunde. In der Regel laufen einfache Quantenprogramme wesentlich kürzer.

Listing 2: Aus Q# erzeugter QIR-Code

```
define void @BellPair__body(%Qubit* %x0, %Qubit* %x1) {
entry:
call void @__quantum__qis__h(%Qubit* %x0)
call void @__quantum__qis__cnot(%Qubit* %x0, %Qubit* %x1)
ret void
}
```

Quantinuum bietet für seine Targets eine Abrechnung nach H1 Quantum Credit (HQC) für die Model-H1-Quantencomputer und Quantinuum Emulator Quantum Credits (eHQC) für den System Model H1 Emulator an, deren geschätzten Verbrauch eine Formel berechnet. Für die 500 US-Dollar Azure Quantum Credits erhält man 40 HQCs und 400 eHQCs. Hat man diese aufgebraucht, wird es deutlich teurer. Der Standardtarif gewährt 10 000 HQCs und 40 000 eHQCs im Monat für 125 000 US-Dollar, und der Premiumtarif 17 000 HQCs und 100 000 eHQCs im Monat für 175 000 US-Dollar.

Während man bei den Quantencomputer-Providern in den Einstiegstarifen nur für die beanspruchte Rechenleistung per Azure Quantum Credits zahlt, entstehen für andere Quantum-Dienste zeitabhängige Kosten. Die Nutzung von 1QCloud schlägt mit rund 6235 Euro monatlich zu Buche. Microsofts QUI-Dienst ist für eine Stunde pro Monat gratis, danach kostet jede Sekunde. Für bis zu 100 parallele Jobs und längere Nutzungszeiten bietet Microsofts Tarif „Performance at Scale“ ein gestaffeltes Preismodell mit Mengenrabatt auf die gebuchten Stunden an. Kosten, die bei Partnern im Azure-Quantum-Ökosystem anfallen, sind nicht durch ein Azure-Guthaben abgedeckt. Man kann deshalb die Gelder für den Azure-Mindestverbrauch oder Gutschriften nicht für diese Dienste einsetzen.

Fazit

Microsoft bietet mit dem Azure-Quantum-Ökosystem ein umfangreiches Paket für den Einstieg in die Quantenprogrammierung. Allerdings ist Quantenhardware noch immer nicht leistungsfähig genug für den produktiven Betrieb und das Bearbeiten sinnvoller Aufgaben. Dem stehen die geringe Zahl der Qubits und diverse Einschränkungen entgegen. Simulatoren bieten zwar mehr Qubits an, sind aber weniger leistungsfähig als echte Quantencomputer. Außerdem lassen sie sich ab einer bestimmten Anzahl von Qubits nicht weiter skalieren.

Dadurch ist das Angebot bislang nur für Entwickler interessant, die den Einstieg in die Quantenwelt finden wollen und ein Gespür für die Ausführung von Anwendungen auf Quantencomputern bekommen möchten. Allerdings können bei Azure Quantum schnell Kosten von über 100 000 Euro im Monat anfallen, was den finanziellen Rahmen der meisten Hobbyprogrammierer sprengen dürfte. Wer Optimierungsaufgaben im produktiven Einsatz bearbeiten will, kann als Übergangslösung die QIO-Dienste und mit Partnern wie 1QCloud weitere Algorithmen nutzen, die sich an den Quantencomputeransatz anlehnen. Quantenhardware steht dafür aber noch nicht zur Verfügung. (sun@ix.de)

Quellen

Weitere Informationen zum QDK finden sich unter ix.de/zapr

Kay Glahn

ist freiberuflich als strategischer und technischer Berater im Bereich neuer und disruptiver Technologien für internationale Kunden tätig.