

# Разработка с использованием КВАНТОВЫХ КОМПЬЮТЕРОВ

Программирование квантовых машин  
в облаке: Python, Qiskit, Quantum Assembly  
language и IBM QExperience

---

Владимир Силва



apress®

# **Practical Quantum Computing for Developers**

**Programming Quantum Rigs in the  
Cloud using Python, Quantum  
Assembly Language and IBM  
QExperience**

**Vladimir Silva**

**Apress®**

# Разработка с использованием КВАНТОВЫХ КОМПЬЮТЕРОВ

Программирование квантовых машин  
в облаке: Python, Qiskit, Quantum Assembly  
language и IBM QExperience

---

Владимир Силва



Санкт-Петербург • Москва • Екатеринбург • Воронеж  
Нижний Новгород • Ростов-на-Дону • Самара • Минск

**2020**

ББК 22.314+32.973.23  
УДК 004:530.145  
С36

## **Силва Владимир**

С36 Разработка с использованием квантовых компьютеров. — СПб.: Питер, 2020. — 352 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1429-0

Квантовые вычисления не просто меняют реальность! Совершенно новая отрасль рождается на наших глазах, чтобы создать немыслимое ранее и обесценить некоторые достижения прошлого.

В этой книге рассмотрены наиболее важные компоненты квантового компьютера: кубиты, логические вентили и квантовые схемы, а также объясняется отличие квантовой архитектуры от традиционной. Вы сможете бесплатно экспериментировать с ними как в симуляторе, так и на реальном квантовом устройстве с применением IBM Q Experience.

Вы узнаете, как выполняются квантовые вычисления с помощью QISKit (программный инструмент для обработки квантовой информации), Python SDK и других API, в частности QASM.

Наконец, вы изучите современные квантовые алгоритмы, реализующие запутанность, генерацию случайных чисел, линейный поиск, факторизацию целых чисел и др. Разберетесь с состояниями Белла, описывающими запутанность, алгоритмом Гровера для линейного поиска, алгоритмом Шора для факторизации целых чисел, алгоритмами оптимизации и многим другим.

**16+** (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 22.314+32.973.23  
УДК 004:530.145

Права на издание получены по соглашению с Apress. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1484242179 англ.  
ISBN 978-5-4461-1429-0

© 2018 by Vladimir Silva  
© Перевод на русский язык ООО Издательство «Питер», 2020  
© Издание на русском языке, оформление ООО Издательство «Питер», 2020  
© Серия «Библиотека программиста», 2020

# Краткое содержание

Об авторе .....	13
О научных редакторах .....	14
Введение .....	15
От издательства .....	21
<b>Глава 1.</b> Странный и прекрасный мир квантовой механики .....	22
<b>Глава 2.</b> Квантовые вычисления: искривление ткани самой реальности .....	46
<b>Глава 3.</b> IBM Q Experience: уникальная платформа для квантовых вычислений в облаке .....	104
<b>Глава 4.</b> QISKit — отличный SDK для квантового программирования на Python .....	167
<b>Глава 5.</b> Запускаем движки: от квантовой генерации случайных чисел до телепортации с остановкой на сверхплотном кодировании .....	217
<b>Глава 6.</b> Развлекаемся квантовыми играми .....	246
<b>Глава 7.</b> Теория игр: с квантовой механикой преимущество всегда на вашей стороне .....	300
<b>Глава 8.</b> Алгоритмы Гровера и Шора: ускоренный поиск и угроза основам асимметричного шифрования .....	327

# Оглавление

Об авторе .....	13
О научных редакторах .....	14
Введение .....	15
От издательства .....	21
 <b>Глава 1.</b> Странный и прекрасный мир квантовой механики .....	22
Двадцатое столетие — золотой век физики .....	23
Макс Планк и ультрафиолетовая катастрофа, с которой все началось .....	24
Квантовый переход Бора .....	26
Битва титанов: коты Шредингера и принцип неопределенности .....	26
Введение в универсальную волновую функцию .....	28
Вероятностная интерпретация $\psi$ : волновая функция была призвана разгромить квантовую механику, а не стать ее основой .....	30
Кот Шредингера пытается сорвать вероятностную вечеринку Борна .....	31
Принцип неопределенности .....	32
Интерференция и двухщелевой эксперимент .....	34
Эйнштейн — Бору: «Бог не играет в кости» .....	35
Бор — Эйнштейну: «Не говори Богу, что ему делать» .....	37
Запутанность и ЭПР-парадокс: мистическое дальное действие .....	38
Неравенство Белла: проверка запутанности .....	39

---

ЭПР-парадокс разгромлен: Бор смеется последним.....	41
Реальность дурачит нас: все взаимосвязано?.....	44
<b>Глава 2. Квантовые вычисления: искривление ткани самой реальности .....</b>	<b>46</b>
Транзистор вступает в противоречие с законами физики.....	47
Пятинанометровый транзистор: большая проблема .....	49
Квантовый масштаб и конец эпохи транзисторов.....	51
Туннелирование электронов .....	51
Эксперименты со щелями .....	59
Вероятное будущее транзисторов .....	60
Ричард Фейнман и квантовый компьютер .....	61
Кубит, странный и потрясающий одновременно .....	63
Суперпозиция состояний.....	64
Запутанность: наблюдение за кубитом изменяет состояние его партнера.....	65
Управление кубитами с помощью квантовых вентилях .....	66
Проектирование кубитов .....	72
Квантовые компьютеры в сравнении с традиционным аппаратным обеспечением .....	80
Сложные симуляции .....	81
Молекулярное моделирование и новые материалы .....	82
Усовершенствованное глубокое обучение.....	83
Квантовые нейронные сети и искусственный интеллект.....	85
Подводные камни квантовых компьютеров: декогеренция и интерференция.....	86
Декогеренция .....	86
Квантовая коррекция ошибок .....	88

Процессор на 50 кубитах и задача для квантового превосходства .....	91
Полемика о квантовом отжиге и минимизации энергии.....	93
Две тысячи кубитов: все не так, как кажется.....	94
Квантовый отжиг: подмножество квантовых вычислений.....	95
Универсальные квантовые вычисления и будущее.....	98
Google и квантовый искусственный интеллект .....	99
Квантовые машины в центрах обработки .....	100
Гонка становится глобальной.....	102
Будущие приложения.....	102
 <b>Глава 3. IBM Q Experience: уникальная платформа для квантовых вычислений в облаке</b> .....	104
Первое знакомство с IBM Q Experience.....	105
Квантовый Composer.....	106
Квантовые вентили.....	106
Доступное квантовое серверное ПО.....	109
Опус 1: вариации на тему состояний Белла и GHZ .....	115
Состояния Белла и мистическое дальноедействие .....	115
Еще более необычно: проверка GHZ-состояний .....	121
Супердетерминизм: уход от мистичности. Был ли Эйнштейн прав все это время?.....	126
Удаленный доступ через REST API.....	129
Аутентификация .....	130
Перечисление доступного серверного ПО .....	131
Получение информации о калибровке заданного процессора .....	133



---

Получение параметров серверного ПО .....	135
Получение статуса очереди процессора.....	138
Перечисление заданий в очереди выполнения.....	139
Получение информации о балансе аккаунта .....	140
Список экспериментов пользователя .....	142
Запуск эксперимента.....	143
Запуск задания .....	149
Получение версии API.....	152
Клиент Node JS для IBMQuantumExperience .....	153
Построение модуля Node для IBMQuantumExperience.....	154
Экспорт методов API.....	155
Аутентификация с использованием токена .....	156
Перечисление серверного ПО .....	158
Перечисление параметров калибровки .....	160
Старт эксперимента .....	161
Отладка и тестирование .....	163
Поделитесь с миром — опубликуйте свой модуль .....	165
<b>Глава 4. QISKit — отличный SDK для квантового программирования</b> <b>на Python.....</b>	<b>167</b>
Установка QISKit.....	168
Настройка в Windows.....	168
Настройка в Linux CentOS .....	169
Кубит 101: базовая алгебра.....	173
Алгебраическое представление квантового бита .....	174
Изменение состояния кубита с помощью квантовых вентилей.....	177

Универсальные квантовые вычисления позволяют получить решение быстрее, чем классические.....	185
Ваша первая квантовая программа .....	186
Внутренние компоненты SDK: компиляция схемы и QASM.....	190
Запуск на реальном квантовом устройстве.....	200
Квантовый ассемблер: мощь, скрытая за кулисами .....	211
 <b>Глава 5.</b> Запускаем движки: от квантовой генерации случайных чисел до телепортации с остановкой на сверхплотном кодировании .....	217
Квантовый генератор случайных чисел.....	217
Генератор случайных битов на основе вентиля Адамара.....	218
Тестирование результатов на случайность.....	224
Сверхплотное кодирование.....	226
Схема в Composer .....	228
Удаленный запуск с использованием Python .....	229
Результаты .....	231
Квантовая телепортация.....	234
Схема в Composer .....	236
Удаленный запуск с помощью Python .....	237
Результаты .....	242
 <b>Глава 6.</b> Развлекаемся квантовыми играми .....	246
Quantum Battleship с изюминкой .....	247
Инструкции по настройке.....	248
Инициализация.....	249
Размещение кораблей на игровом поле .....	250
Основной цикл и результаты .....	252

Cloud Battleship: модификация удаленного доступа.....	258
Упражнение 6.1. Разделение интерфейса пользователя и логики игры .....	259
Упражнение 6.2. Создание веб-интерфейса для игрового поля.....	260
Упражнение 6.3. Развертывание и устранение неполадок на сервере Apache .....	262
Решение 6.1. Программа на Python, позволяющая повторное использование.....	263
Решение 6.2. Интерфейс пользователя .....	269
Решение 6.3. Развертывание и тестирование .....	281
Устранение ошибок.....	284
Дополнительные улучшения .....	287
<b>Глава 7. Теория игр: с квантовой механикой преимущество всегда на вашей стороне .....</b>	<b>300</b>
Загадка про фальшивую монету .....	300
Квантовый способ решения .....	302
Шаг 1. Запрос к квантовым весам .....	303
Шаг 2. Создание квантовых весов.....	307
Шаг 3. Определение фальшивой монеты .....	307
Обобщенный алгоритм для любого количества фальшивых монет.....	309
Магический квадрат Мермина — Переса .....	310
Упражнение для магического квадрата Мермина — Переса .....	312
Квантовая стратегия победы.....	313
Общее запутанное состояние.....	313
Унитарные преобразования .....	315
Измерение в вычислительном базисе.....	321
Ответы для упражнения с магическим квадратом .....	325

<b>Глава 8. Алгоритмы Гровера и Шора: ускоренный поиск</b>	
и угроза основам асимметричного шифрования.....	327
Квантовый неструктурированный поиск.....	328
Фазовая инверсия.....	329
Инверсия относительно среднего значения.....	330
Практическая реализация .....	331
Обобщенная схема.....	335
Факторизация целых чисел при помощи алгоритма Шора.....	338
Квантовая факторизация бросает вызов асимметричному	
шифрованию .....	338
Нахождение периода .....	340
Алгоритм Шора с использованием ProjectQ.....	344

## Об авторе

**Владимир Сильва** окончил Государственный университет Мидл Теннесси (Middle TN State University), получив диплом магистра в области Computer Science. На протяжении пяти лет он работал в IBM инженером-исследователем (Research Engineer), где приобрел богатый опыт в распределенных и GRID-вычислениях.

У Владимира есть множество сертификатов, в том числе OCP (Oracle Certified Professional), MCSD (Microsoft Certified Solutions Developer) и MCP (Microsoft Certified Professional). Кроме того, он является автором большого количества технических статей для сайта IBM developerWorks. Он написал следующие книги: *Grid Computing for Developers* (Charles River Media), *Practical Eclipse Rich Client Platform* (Apress), *Pro Android Games* (Apress) и *Advanced Android 4 Games* (Apress).

Владимир — заядлый марафонец, участвовал в 16 забегах в штате Северная Каролина (на момент написания книги). Любит играть на классической гитаре и размышлять о таких удивительных вещах, как квантовая механика.

# О научных редакторах

## *Оригинальное издание*



**Джейсон Уайтхорн** — опытный предприниматель и разработчик программного обеспечения. Он помог многим нефтегазовым компаниям автоматизировать и усовершенствовать их технологии с помощью сбора эксплуатационных данных, SCADA (Supervisory Control and Data Acquisition — диспетчерское управление и сбор данных) и машинного обучения. Джейсон окончил Арканзасский государственный университет (Arkansas State University), получив диплом бакалавра в области Computer Science.

Свободное время Джейсон любит проводить со своей женой и четырьмя детьми. Живет в Талсе, штат Оклахома. Больше информации о Джейсоне можно найти на его сайте <https://jason.whitehorn.us>.

## *Русскоязычное издание*



**Михаил Коробко** — физик, занимается теорией и экспериментами по применению методов квантовой оптики, оптомеханики и квантовых измерений для улучшения чувствительности гравитационно-волновых детекторов. С 2012 года состоит в международной коллаборации ученых гравитационно-волнового детектора LIGO.

Михаил закончил физический факультет МГУ им. Ломоносова, в настоящий момент является аспирантом Института лазерной физики в университете Гамбурга. Свободное время он проводит с семьей, пишет научно-популярные статьи о квантовой физике и публикует посты в «Твиттере» (@hbar\_universe).

# Введение

Эта книга задумывалась как полный справочник по программированию квантового компьютера в облаке. Ее появление стало возможным благодаря IBM Research. IBM создала свою экспериментальную квантовую установку, известную как IBM Q Experience. Она доступна не только для научных исследований — ею могут пользоваться разные люди, интересующиеся данной областью вычислений.

Сегодня квантовые вычисления набирают популярность, и настало время научиться программировать квантовые машины. В ближайшие годы<sup>1</sup> должны появиться первые коммерческие квантовые компьютеры, которые обещают значительное повышение скорости вычислений по сравнению с классическими. Рассмотрим график, показывающий временную сложность для двух алгоритмов факторизации больших целых чисел: лучшего классического алгоритма — метода решета числового поля в сравнении с квантовым алгоритмом факторизации, разработанным Питером Шором (рис. В.1).

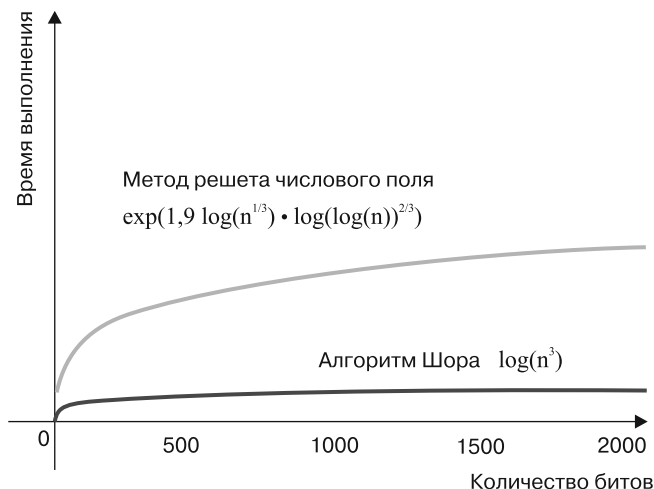
Алгоритм Шора по сравнению с методом решета числового поля обеспечивает более быстрое решение задачи, на которой основана современная криптография. Практическая реализация этого алгоритма сделает актуальное сегодня асимметричное шифрование бесполезным!

В целом при написании книги я пытался глубже вникнуть во многие темы. Если вам трудно разобраться в концепциях, изложенных в главах, то

---

<sup>1</sup> IBM создали 53-кубитный чип и обещают, что в середине ноября он будет доступен пользователям IBM Q Network. Их конкуренты Google, в свою очередь, создали 54-кубитный компьютер и 23 октября 2019 г. опубликовали в Nature статью, где заявили, что достигли квантового превосходства. С этой целью они разработали алгоритм генерации случайных чисел. По утверждению Google, эту задачу на классическом суперкомпьютере можно решить за 10 000 лет, в то время как квантовому потребовалось лишь 3 минуты 20 секунд. Однако в IBM доказывают, что задачу можно решить на классическом компьютере с помощью другого классического алгоритма за 2,5 дня. — *Примеч. пер.*

вы не одиноки. Великий физик Ричард Фейнман однажды сказал: «Если кто-то скажет вам, что он понимает квантовую механику, это значит, что он не понимает квантовую механику». Даже титаны этой странной теории изо всех сил пытались понять ее.



**Рис. В.1.** График, показывающий временную сложность для двух алгоритмов факторизации больших целых чисел

Я постарался рассмотреть квантовые вычисления в меру своих возможностей, используя реальные алгоритмы, схемы, код и результаты в виде графиков. Некоторые из алгоритмов в книге бросают вызов логике и кажутся мало похожими на вычислительное описание физической системы. Несмотря на то что мне трудно понять ошеломляющие принципы квантовой механики, я всегда был очарован этой удивительной теорией. Таким образом, когда IBM предложила единственную в своем роде платформу квантовых вычислений в облаке и открыла ее для всех нас, я воспользовался данной возможностью для изучения этой темы и написания книги.

В конечном счете это мой взгляд на квантовые вычисления в облаке, и я надеюсь, что вы получите такое же удовольствие от чтения, как я от написания. Мой скромный совет: учитесь программировать квантовые компьютеры — скоро они будут стоять в каждом центре обработки данных



и использоваться в разных целях: от поиска и симуляции до медицинских исследований и создания искусственного интеллекта. Вам останется лишь сформулировать для них задание.

Книга состоит из следующих глав.

### ○ Глава 1. Странный и прекрасный мир квантовой механики.

Все началось в 1930-х годах с Макса Планка, гения поневоле. Он придумал новую интерпретацию для распределения энергии спектра света. Сначала он, не желая того, выдвинул постулат: энергия фотона описывается не непрерывной функцией, как полагают классические физики, а крошечными порциями — *квантами*. Так началась величайшая научная революция XX столетия — стала развиваться *квантовая механика*. Эта глава является введением к основному курсу и исследует конфликт двух титанов физики — Альберта Эйнштейна и Нильса Бора. Квантовая механика в 1930-х годах была революционной теорией, и большинство ученых, в том числе колосс века Альберт Эйнштейн, неохотно приняли ее. Только что получив Нобелевскую премию, Эйнштейн не признавал вероятностного характера квантовой механики. Это привело к его конфликту с крупнейшим сторонником теории — Нильсом Бором. Два великих человека десятилетиями обсуждали ее, но так и не пришли к единому мнению. В конечном счете квантовая механика выдержала 70 лет теоретических и экспериментальных испытаний, всегда выходя из них победительницей. Прочитайте эту главу и рассмотрите теорию, эксперименты и результаты — все сквозь призму невероятной истории этих двух выдающихся личностей.

### ○ Глава 2. Квантовые вычисления: искривление ткани самой реальности.

В 1980-х годах другой великий физик, Ричард Фейнман, предложил квантовый компьютер, то есть компьютер, который может использовать преимущества принципов квантовой механики для более быстрого решения задач. Началась гонка по созданию такой машины. В этой главе в общих чертах рассматривается базовая архитектура квантового компьютера, в частности кубиты — основные элементы квантовых вычислений. Они могут показаться незначительными, но обладают почти волшебным свойством — суперпозицией. Верите или

нет, но кубит может находиться одновременно в двух состояниях: 0 и 1. Данную концепцию трудно осознать в макромасштабах, в которых мы живем. Однако в масштабе атома все возможно. Этот факт доказывался экспериментально на протяжении более чем 70 лет. Таким образом, суперпозиция позволяет квантовому компьютеру превзойти классический компьютер и выполнять большое количество вычислений с относительно небольшим числом кубитов. Другим трудно постижимым понятием является запутывание кубитов — то, что при исследовании кажется больше похожим на мистику, чем на физический принцип. Запутанные между собой кубиты передают состояния во времени или в пространстве быстрее, чем со скоростью света! Хорошенько обдумайте эти слова. В целом в главе рассматриваются все физические компоненты квантового компьютера: квантовые вентили, кубиты разных типов, такие как сверхпроводящие контуры, ионные ловушки, топологические косы и многое другое. Кроме того, мы поговорим об основных современных исследованиях этой технологии, а также рассмотрим разные типы квантовых вычислений, например квантовый отжиг (квантовая нормализация, quantum annealing).

### ○ Глава 3. IBM Q Experience: уникальная платформа для квантовых вычислений в облаке.

В этой главе вы познакомитесь с IBM Q Experience. Это первая платформа для квантовых вычислений в облаке, которая предоставляет реальные или симулированные квантовые устройства. Долгое время подразумевалось, что это квантовое устройство будет доступно только для исследовательских целей. Но теперь это уже не так благодаря ребятам из IBM, которые десятилетиями работали над платформой и любезно решили открыть ее для публичного использования.

Узнайте, как создать квантовую схему (quantum circuit, еще называют квантовым контуром) с помощью Visual Composer, или постройте ее, используя непревзойденный Python SDK. Затем реализуйте свою схему в реальных условиях, изучите результаты и сделайте первый шаг в карьере квантового программиста. Компания IBM создала первую платформу для квантовых вычислений в облаке, но ее конкуренты не отстают. Думаю, в ближайшем будущем мы увидим новые облачные платформы от других ИТ-гигантов. А сейчас — время учиться.

## ○ Глава 4. QISKit — отличный SDK для квантового программирования на Python.

QISKit расшифровывается как Quantum Information Software Kit. Это Python SDK для написания квантовых программ в облаке или на локальном симуляторе. Из этой главы вы узнаете, как настроить Python SDK на своем ПК. Далее изучите описание квантовых вентилях с использованием линейной алгебры, чтобы глубже понять, что происходит за кулисами. Это позволит вам ознакомиться с синтаксисом Python SDK, чтобы затем приступить к созданию первой квантовой программы. Наконец, вы запустите ее на реальном квантовом устройстве. Конечно, квантовые программы могут быть созданы и визуально в Composer. Хорошо разберитесь в квантовых вентилях — основных составляющих квантовой программы. Все это и многое другое рассматривается в данной главе.

## ○ Глава 5. Запускаем движки: от квантовой генерации случайных чисел до телепортации с остановкой на сверхплотном кодировании.

Эта глава представляет собой путешествие по трем замечательным функциям обработки информации в квантовых системах. Генерация квантовых случайных чисел позволяет исследовать природу квантовой механики как источника истинной случайности. Вы узнаете, как осуществить генерацию, используя очень простые логические элементы и Python SDK. Далее в этой главе рассматриваются два связанных протокола обработки информации: сверхплотное кодирование и квантовая телепортация. У них колоритные названия и почти волшебные свойства. Раскройте их секреты, постройте схемы для Composer, выполните удаленно, используя Python, и, наконец, интерпретируйте и проверьте на корректность их результаты.

## ○ Глава 6. Развлекаемся квантовыми играми.

Из этой главы вы узнаете, как реализовать несложную игру на квантовом компьютере. Для этой цели мы воспользуемся классической игрой Quantum Battleship («Морской бой»), которая поставляется вместе с руководством по QISKit в Python. Первая часть посвящена технике игры, но на этом мы не остановимся. Во второй части главы перейдем на следующий уровень, значительно изменив интерфейс игры. В этой части вы поместите игру «Морской бой» в облако,

предоставив ей пользовательский интерфейс, доступный из браузера, интерфейс Apache CGI для приема событий и отправки их на квантовый симулятор и многое другое. Произведите впечатление на своих друзей и семью, играя в квантовый «Морской бой» в собственных браузерах в облаке.

○ **Глава 7. Теория игр: с квантовой механикой преимущество всегда на вашей стороне.**

В данной главе рассматриваются две головоломки, которые демонстрируют мощь квантовых алгоритмов по сравнению с их классическими аналогами: головоломка про фальшивую монету и магический квадрат Мермина — Переса. В первой применяется квантовый алгоритм для повышения скорости решения по сравнению с классическим поиском фальшивой монеты, при котором используется шкала весов и ограничено количество взвешиваний. Магический квадрат Мермина — Переса является примером квантовой псевдотелепатии, или способности игроков практически читать мысли друг друга, достигая результатов, возможных только в том случае, если они общаются во время игры.

○ **Глава 8. Алгоритмы Гровера и Шора: ускоренный поиск и угроза основам асимметричного шифрования.**

Эта глава завершает работу с двумя алгоритмами, которые позволяют осознать возможности практических квантовых вычислений. Первый из них — алгоритм поиска Гровера — неструктурированный алгоритм квантового поиска, способный находить входные данные в среднем за  $\sqrt{N}$  шагов. Он работает намного быстрее, чем лучшее классическое решение при  $N/2$  шагах. Может показаться, что это не так уж и много, но, если говорить об очень больших базах данных, алгоритм Гровера может значительно превзойти классический алгоритм поиска. Подозреваю, что в будущем весь поиск в Интернете будет выполняться по алгоритму Гровера. Второй — алгоритм факторизации целых чисел Шора: пресловутая квантовая факторизация, которая, по мнению экспертов, может поставить на колени современное асимметричное шифрование. Этот алгоритм как нельзя лучше демонстрирует мощь квантовых вычислений, обеспечивая экспоненциальное увеличение скорости по сравнению с классическими вычислениями.

## От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу [comp@piter.com](mailto:comp@piter.com) (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства [www.piter.com](http://www.piter.com) вы найдете подробную информацию о наших книгах.

# 1

## Странный и прекрасный мир квантовой механики

История квантовой механики удивительна и невероятна. В ней есть элементы науки, философии, религии и, смею сказать, волшебства. Она перевернет ваш разум с ног на голову, а иногда даже заставит вас усомниться в существовании всемогущего Творца. Несмотря на то что мне трудно до конца понять ее концепции, я всегда был ею очарован. Некоторые из этих концепций, представленные в данной главе, трудно осмыслить, но не стоит беспокоиться. Никто не смог полностью описать, что все это значит, даже титаны физики не до конца понимают квантовую механику. Однако это не означает, что мы не можем ею восхищаться. Великий физик Ричард Фейнман однажды сказал: «Если кто-то скажет вам, что он понимает квантовую механику, это значит, что он не понимает квантовую механику». Данная глава — мое изложение этой увлекательной истории и того, как борьба двух титанов науки определила ее прошлое, настоящее и будущее.

Все началось в 1930-х годах, после того как Альберт Эйнштейн достиг мировой славы, разработав теорию относительности, которая была построена на основе ньютоновской физики с целью соединить небеса и землю. Пока Эйнштейн смотрел в небеса, ученые нового поколения во главе с такими гигантами физики, как Макс Планк, Эрнест Резер-

форд и Нильс Бор, обратили свои взгляды на мельчайшие частицы. Произошло столкновение титанов, начались одни из величайших научных дебатов в XX веке: с одной стороны, Альберт Эйнштейн, только что получивший Нобелевскую премию за новаторское открытие, касающееся природы света, с другой — Нильс Бор, чей вклад в квантовую механику принесет ему Нобелевскую премию (1922) и датский орден Слона — знак отличия, которым обычно награждали представителей королевских семей. Давайте посмотрим, как противостояние между двумя великими людьми привело к созданию научного шедевра, то есть квантовой механики.

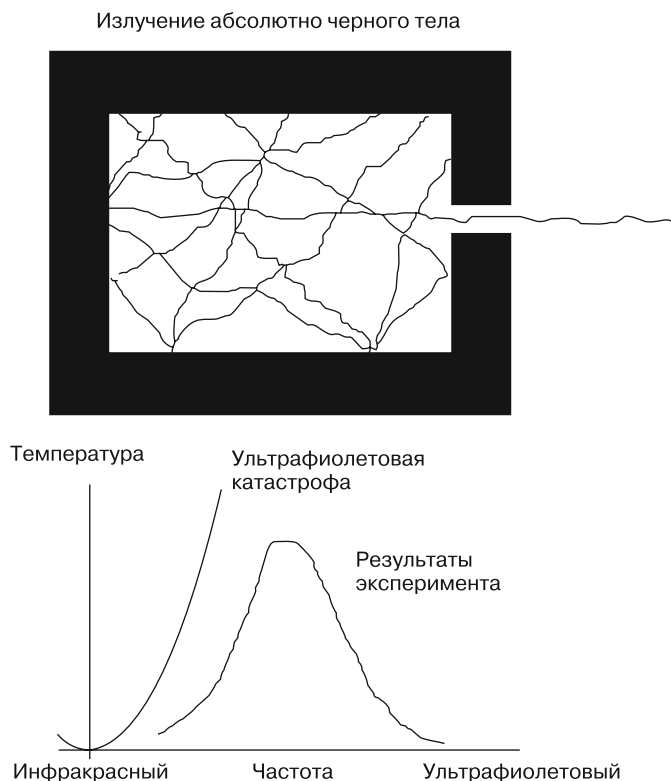
## Двадцатое столетие — золотой век физики

В начале XX века британский ученый Эрнест Резерфорд сделал потрясающее открытие — вывел строение атома. Он предположил, что атомы выглядят как миниатюрные солнечные системы, состоящие из ядра с положительным зарядом и отрицательно заряженных электронов, вращающихся вокруг него, как крошечные планеты. Это было значительным прорывом, поскольку ранее считалось, что атом представляет собой простой сферический сгусток материи с положительным и отрицательным зарядами.

Бор прибыл в лабораторию Резерфорда в Кембридже в 1920 году. Он сразу влюбился в модель атома, в которой, однако, крылась проблема, причем большая. Если к модели Резерфорда применить классическую ньютоновскую физику, где отрицательно заряженные электроны вращаются вокруг положительно заряженного ядра, электрон в конечном итоге упадет внутрь и столкнется с ядром, создав катастрофический парадокс. Бор настолько заинтересовался этим, что отложил свою свадьбу и отменил медовый месяц, чтобы спасти модель Резерфорда. Бор опубликовал статью, где предположил, что электроны движутся по фиксированным орбитам, которые не изменяются. Это мнение шло вразрез с основами ньютоновской физики, но опиралось на новые идеи отца квантовой механики Макса Планка.

## Макс Планк и ультрафиолетовая катастрофа, с которой все началось

Планк предположил, что тепло и свет состоят из неделимых частей, которые он назвал квантами света. Эта идея возникла в результате попыток объяснить эксперименты с излучением абсолютно черного тела, когда тело, полностью поглощающее любое излучение (тепло), имеет внутри полость с отверстием, позволяющим части излучения выйти наружу (рис. 1.1). С возрастанием температуры внутри полости частота излучения достигает видимых человеческому глазу диапазонов, соответствующих разным цветам. В то время, например, производители фарфора хорошо знали, что все тела при определенных температурах окрашиваются в конкретные цвета (табл. 1.1).



**Рис. 1.1.** Результаты эксперимента по излучению абсолютно черного тела



**Таблица 1.1.** Окраска света при различных температурах

Температура, °C	Цвет
500	Темно-красный
800	Вишнево-красный
900	Оранжевый
1000	Желтый
1200	Белый

На рис. 1.1 проиллюстрирован эксперимент по излучению абсолютно черного тела совместно с результатами, соответствующими классической теории спектральных линий, полученными во время экспериментов 1890-х годов. Эксперименты физика предсказали бесконечную интенсивность ультрафиолетового спектра. Это явление, которое стало известным как ультрафиолетовая катастрофа, было порождением сомнительных теоретических споров и экспериментальных результатов. Если это правда, то получается, что опасно сидеть даже рядом с камином! Планк стремился найти решение проблемы ультрафиолетовой катастрофы.

Ученый воспользовался вторым законом термодинамики, связанным с понятием энтропии, чтобы вывести формулу для экспериментальных результатов, полученных при излучении абсолютно черного тела:

$$S = -k \log W,$$

где  $S$  — энтропия Больцмана;  $k$  — постоянная Больцмана;  $W$  — вероятность того, что для элемента, будь то твердое тело, жидкость или газ, будет достигнуто определенное расположение атомов.

Используя статистический метод Больцмана для расчета энтропии, Планк искал формулу, соответствующую результатам эксперимента с абсолютно черным телом. Разделив полную энергию  $E$  на порции пропорционально частоте  $f$ , он получил уравнение:

$$E = hf,$$

где  $E$  — порция энергии;  $h$  — постоянная Планка;  $f$  — частота. Тем не менее он столкнулся с проблемой: статистический метод Больцмана

требовал, чтобы порции со временем уменьшались до нуля. Это свело бы на нет уравнение и тем самым лишило бы его смысла. После долгих экспериментов Планк вынужден был предположить, что количество энергии должно быть конечным. И на Планка снисходит невероятное озарение: если это справедливо, то означает, что генератор не может поглощать или излучать энергию в непрерывном диапазоне. Поглощение и излучение энергии должны происходить небольшими неделимыми порциями  $E = hf$  — так называемыми квантами света. Отсюда и термин «квантовая механика».

## Квантовый переход Бора

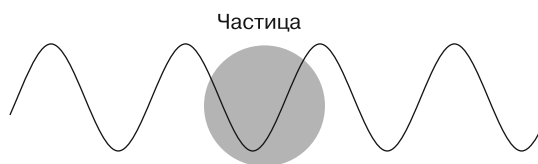
Нильс Бор применил предложенную Планком новаторскую идею квантов света к атому — наименьшей единице материи. Он дал смелое описание взаимосвязи атома и света: электрон, вращающийся вокруг ядра, будет излучать или поглощать свет, вызывая квантовый переход. Следовательно, квантовый переход был переходом между двумя состояниями, однако Бор не смог полностью описать это.

Другие ученые приняли идею скептически и называли теорию бессмыслицей, дешевым оправданием незнания или считали слишком смелой, слишком фантастической, чтобы быть правдой. Результатом стал раскол в физическом сообществе на тех, кто сплотился вокруг Бора, который верил в квантовую природу материи, и тех, кто поддерживал классическую точку зрения. В этом противостоянии Эйнштейн вскоре присоединился к борьбе на классической стороне.

## Битва титанов: коты Шредингера и принцип неопределенности

К середине 1920-х годов новая теория о квантовой природе материи оказалась в шатком положении, стоя перед реальной перспективой скорого упадка. Для того чтобы укрепить ее фундамент, потребовалось два новых революционных открытия.

Первое произошло в 1926 году, когда немецкий физик Вернер Гейзенберг попытался узаконить взгляды Бора, создав математическое описание атома для того, что сейчас известно как матричная механика. Эта идея считалась слишком сложной для восприятия даже опытным физикам. Тем не менее самым большим вкладом Гейзенберга в эту область является знаменитый принцип неопределенности, который мы рассмотрим далее. Второе открытие пришло от австрийского физика Эрвина Шредингера, который описал атом не как частицу, а как волну. Эта идея основывалась на аргументах французского физика-теоретика Луи де Бройля, который предположил, что частицы могут проявлять волновые свойства и что двойственность может быть необходима для понимания природы света (рис. 1.2).



**Рис. 1.2.** Двойственная природа фотона. Он ведет себя как частица и как волна

Де Бройль воспользовался знаменитым уравнением Эйнштейна для энергии  $E = mc^2$  и квантами света Планка  $E = hf$ , чтобы найти взаимосвязь между длиной волны  $\lambda$  и импульсом  $P$  фотона:

$$E = mc^2 = (mc)c.$$

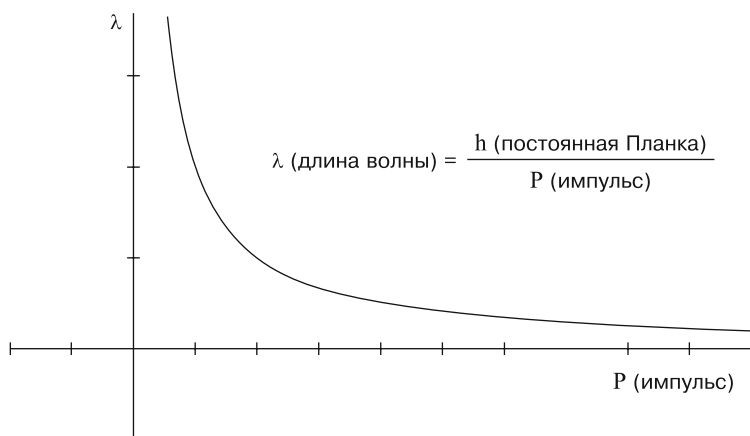
С учетом того, что  $mc$  — импульс  $P$  фотона, а  $c$  (скорость)  $= f$  (частота)  $\cdot \lambda$  (длина волны), уравнение принимает вид:

$$E = P(f\lambda).$$

Но подождите, формула Планка утверждает, что энергия  $E = hf$ . Таким образом, используя базовую алгебру, де Бройль пришел к заключению:

$$\begin{aligned} hf &= P(f\lambda); \\ h &= P\lambda; \\ \lambda &= h / P. \end{aligned}$$

Де Бройль показал, что длина волны фотона уменьшается с увеличением импульса (рис. 1.3). По аналогии он предположил, что это соотношение верно не только для фотонов, но и для всех частиц. С учетом того, что в то же время импульс электрона  $P = (\text{масса}) \cdot (\text{скорость})$  можно легко определить экспериментально, это означало, что длину волны можно рассчитать из уравнения де Бройля! В то время идея казалась нелепой, так как классические физики знали, что электрон является частицей — это открытие было сделано Дж. Дж. Томсоном задолго до того, в 1897 году.



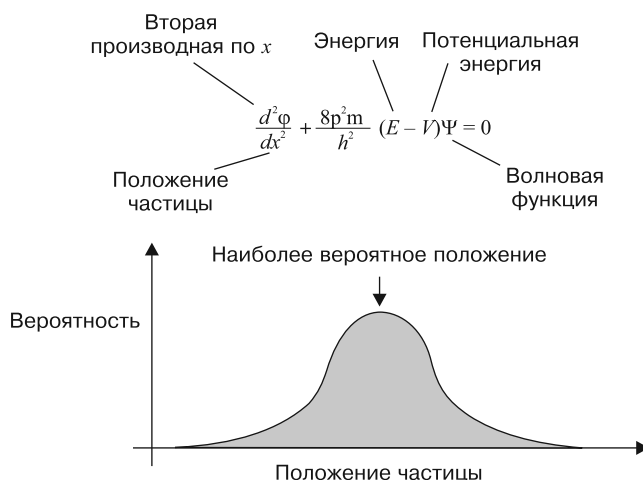
**Рис. 1.3.** Соотношение де Бройля между длиной волны и импульсом фотона

Шредингер использовал идеи де Бройля, чтобы найти подход, который был бы более приемлемым для существующего положения, ознаменовав возвращение в непрерывный, визуализируемый мир классической физики. Он был прав насчет волновой функции, но совершенно не прав насчет существующего положения вещей.

## Введение в универсальную волновую функцию

Шредингер стремился найти такую функцию, чтобы ее можно было применить к любой физической системе, для которой известна математическая форма энергии. Одним из основных уравнений квантовой механики является уравнение Шредингера, определяющее изменение состояний

квантовых систем с течением времени. Его часто называют волновым уравнением Шредингера, а его решение — волновой функцией, зависящей от времени. Она обозначается греческой буквой  $\psi$  (произносится «пси», рис. 1.4). Волновая функция Шредингера была немедленно принята как математический инструмент исключительной силы для решения задач, связанных с атомной структурой материи, и считается одним из величайших достижений XX века.



**Рис. 1.4.** Знаменитая волновая функция Шредингера стремилась описать любую физическую систему с известной энергией

Бор и Гейзенберг объединили усилия со Шредингером, чтобы использовать невероятную мощь его волновой функции, но сначала им нужно было разрешить свои разногласия. Три гиганта встретились для их обсуждения в 1926 году в недавно созданном институте в Копенгагене.

Шредингер отверг концепцию Бора и Гейзенберга о дискретных квантовых переходах в структуре атома. Он хотел использовать свое новое открытие как путь назад к непрерывным физическим процессам, не нарушаемым внезапными переходами. На самом деле он предлагал классическую теорию материи, основанную исключительно на волнах, вплоть до сомнения в существовании частиц. Шредингер предположил, что частицы на самом деле являются суперпозицией волн. Ошибочность этого утверждения

позже была обоснована Хендриком Лоренцем, который доказал, что их (частицы) все равно невозможно победить. Позже Шредингер будет сомневаться в важности волновых колебаний как источника всей физической реальности.

Бор, Гейзенберг и Шредингер спорили яростно, до изнеможения. Бор требовал абсолютной ясности во всех аргументах, пытаясь заставить Шредингера признать, что его интерпретация была неполной. Шредингер придерживался классической точки зрения, иногда оплакивая свою работу по теории атома и квантовым переходам (то, что он, вероятно, не имел в виду). Шредингер ненавидел интерпретацию строения атома, предложенную Бором. Нужен был последний элемент, чтобы эти двое смогли прийти к соглашению о цельной квантовой теории.

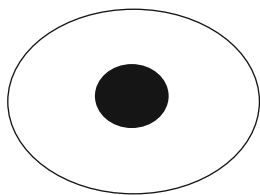
### **Вероятностная интерпретация $\psi$ : волновая функция была призвана разгромить квантовую механику, а не стать ее основой**

Великий рок-гитарист Джимми Хендрикс услышал мелодию Ней Джо, выпустил кавер и сделал ее своей, создав таким образом, пожалуй, один из величайших каверов. Точно так же поступили и отцы квантовой механики. Они осознали огромную силу волновой функции и присвоили ее. В этой истории есть небольшой любопытный нюанс: Шредингер терпеть не мог дискретную интерпретацию энергии и тепла, данную Планком. Он хотел использовать свою гладкую и непрерывную волновую функцию, чтобы победить кванты света Планка. В это трудно поверить, но в 1930-е годы открытие Планка было настолько революционным, что большинство физиков считали его сумасшедшим. Тем не менее так же, как Хендрикс — ту мелодию, основатели квантовой механики сделали своей волновую функцию.

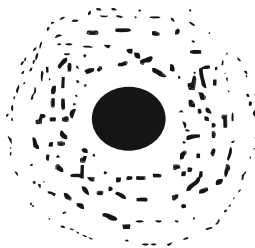
Прорыв был осуществлен немецким физиком Максом Борном, который разработал идею волновой функции как вероятности того, что электрон в данном состоянии рассеется в некотором направлении. Борн заявил, что вероятность  $P$  существования состояния задается квадратом модуля амплитуды волновой функции, то есть  $P = |\psi|^2$ . Этот подход был новатор-

ским в то время, ведь Борн не требовал более точных ответов: все, что мы получаем в атомной теории, — это вероятности. Эта принципиально новая идея привела Нильса Бора к интерпретации атома в совершенно новом ключе (рис. 1.5).

Основное состояние водорода



Нильс Бор



Макс Борн

**Рис. 1.5.** Модель Бора в сравнении с вероятностным толкованием волновой функции, данным Максом Борном

## Кот Шредингера пытается сорвать вероятностную вечеринку Борна

По мере того как идея Борна о вероятностной природе  $\psi$  набирала обороты, Шредингер предположил, что его волновая функция использовалась некорректно, и это привело к знаменитому мысленному эксперименту, который позже будет известен как «Кот Шредингера», о котором вы, вероятно, слышали. В эксперименте Шредингер стремился опровергнуть вероятностную интерпретацию  $\psi$  Борна. Это выглядит так: живого кота помещают в ящик с радиоактивным источником, который является пусковым устройством для молотка, разбивающего колбу с ядом, который, в свою очередь, убивает кота. Если предположить, что вероятность радиоактивного распада составляет 50 % в час, то через час механизм может сработать и тогда кот погибнет. Шредингер утверждал: согласно интерпретации Борна квантовая теория предсказывает, что через час кот в ящике не будет ни мертвым, ни живым, а будет комбинацией двух состояний — суперпозицией обеих волновых функций. Шредингер считал, что это нелепо и приводит к парадоксу. Однако сегодня этот так называемый

парадокс используется для обучения квантовым вероятностям и суперпозиции состояний.

Удивительное свойство суперпозиции таково: как только ящик открывается, волновые функции в состоянии суперпозиции коллапсируют в одно состояние, делая кота или мертвым, или живым, — таким образом, акт наблюдения разрешает тупиковую ситуацию. Еще одно невероятное озарение посетит Гейзенберга, размышляющего о конкретной степени неопределенности местоположения частицы в структуре атома, отстаиваемой Бором.

## Принцип неопределенности

Гейзенберг рассуждал о том, почему местонахождение частицы в атоме Бора невозможно определить. После долгих размышлений в момент озарения он понял: для того чтобы узнать, где находится частица, нужно взглянуть на нее, а для этого — буквально пролить на нее свет. Но когда вы делаете это, положение частицы изменяется, таким образом, акт наблюдения частицы влияет на ее положение. Гейзенберг назвал эту идею *принципом неопределенности*.

Для изучения проблемы Гейзенберг разработал гипотетический эксперимент с использованием микроскопа, испускающего гамма-лучи с большим импульсом и низкой частотой навстречу наблюдаемому электрону. Определенная в сотрудничестве с Бором цель состояла в том, чтобы описать количественное соотношение путем оценки неопределенности при одновременном измерении импульса и координаты. Было установлено, что неопределенность координаты близка к длине волны излучения,  $\Delta X \sim \lambda$ .

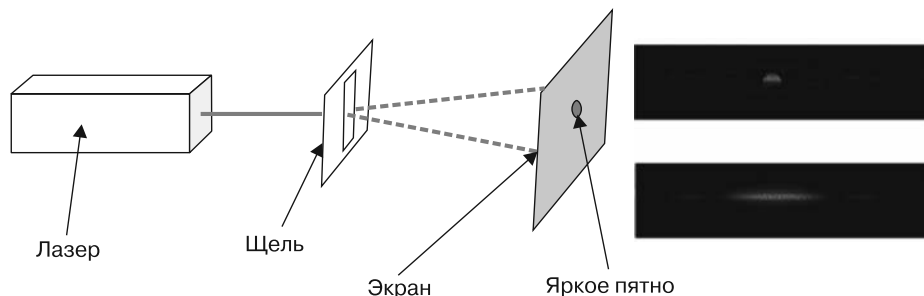
Точно так же неопределенность импульса электрона близка к импульсу фотона, используемого для освещения частицы,  $\Delta P \sim h / \lambda$ . Отметим следующее: из уравнения де Бройля известно, что импульс фотона  $P = h / \lambda$ . Гейзенберг показал, что при перемножении обеих неопределенностей произведение всегда будет больше или равно  $h$ :

$$\begin{aligned}\Delta X \cdot \Delta P &\geq \lambda \cdot h / \lambda; \\ \Delta X \cdot \Delta P &\geq h.\end{aligned}$$



Это принцип неопределенности Гейзенберга, который гласит: «Неопределенность одновременного измерения импульса и сопряженной ему координаты всегда больше некоторой постоянной величины и близка к постоянной Планка  $h$ ».

Существует простой эксперимент, который физики обычно используют, чтобы показать принцип неопределенности в действии. Он называется *экспериментом с одной щелью* и выглядит следующим образом: лазерный луч проходит через широкую вертикальную щель и падает на проекционный экран. На нем мы видим именно то, что и ожидали, — точку. Теперь, если уменьшать ширину щели, границы точки также начнут сжиматься. Однако при ширине щели около 0,254 мм (1/100 дюйма) принцип неопределенности начинает действовать и, по словам Гейзенберга, направление луча становится неопределенным. Таким образом, теперь мы наблюдаем, как свет распространяется все шире и шире! Звучит безумно: как пучок света может стать шире, если сделать щель *уже*? Это совершенно непонятно на интуитивном уровне, но так все и работает.



**Рис. 1.6.** Эксперимент с одной щелью продемонстрировал принцип неопределенности в действии

Принцип неопределенности имеет огромное значение, потому что он устраняет разрыв между теориями Шредингера и Бора, закладывая фундамент современной квантовой теории. То есть электрон — это частица, как предположил Бор, но мы не знаем точно, где он находится, как гласит принцип неопределенности (Гейзенберг). Наконец, вероятность его местонахождения определяется волновой функцией (Шредингер/

Борн). Таким образом, для электрона характерна двойственность — он и частица, и волна. На основе всего этого возникает цельное представление о квантовой механике, которое позже будет известно как копенгагенская интерпретация.

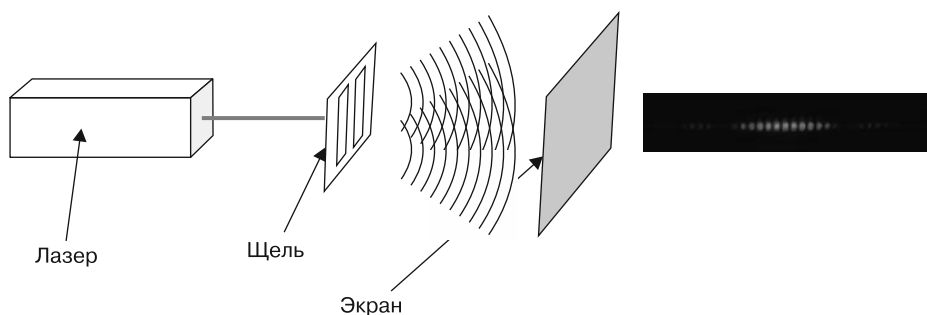
## **Интерференция и двухщелевой эксперимент**

Интерференция — еще одно невероятное свойство квантовой механики, которое заставляет задуматься о том, что происходит в мире за кулисами нашей реальности. Великий физик Ричард Фейнман однажды сказал об интерференции: основы квантовой механики можно понять из исследования интерференции и двухщелевого эксперимента.

Хорошо известно, что в начале XIX века шли дискуссии о природе света. Некоторые, как Ньютон, утверждали, что он состоит из частиц, другие предполагали, что он ведет себя как волна. В 1801 году Томас Юнг поставил опыт с двумя щелями, пытаясь свести эти теории воедино. В этом эксперименте луч света направлен на экран с двумя вертикальными щелями. После того как свет проходит через щели, полученный рисунок фиксируется на экране. Когда одна щель закрыта, видна одна линия света, соответствующая открытой щели. Здравый смысл и интуиция говорят нам, что, когда обе щели открыты, результирующий рисунок продемонстрирует две линии света, параллельные щелям. Невероятно, но это не так. На практике свет, проходящий сквозь щели и отображаемый на фотопластинке, разделяется на несколько линий различной степени яркости (рис. 1.7).

Этот странный результат озадачил физиков, которые предположили, что между волнами и частицами, проходящими через щели, происходит интерференция. Если поток фотонов достаточно слаб, чтобы гарантировать, что на пластину попадают отдельные фотоны, то можно ожидать, что будут видны две световые линии (один фотон, проходящий через первую или вторую щель и оказывающийся в одной из двух возможных световых линий). Однако это не так. Свет каким-то образом делает невозможное:

каждый фотон не только проходит через обе щели, но и одновременно проходит каждую возможную траекторию на пути к цели.



**Рис. 1.7.** Двухщелевой эксперимент, проведенный Томасом Юнгом

То, что такие события, как интерференция, которая кажется невозможной, могут происходить в атомном масштабе, сбивало с толку величайшие умы того времени. А вскоре новая теория столкнется с самой большой проблемой титана физики Альберта Эйнштейна.

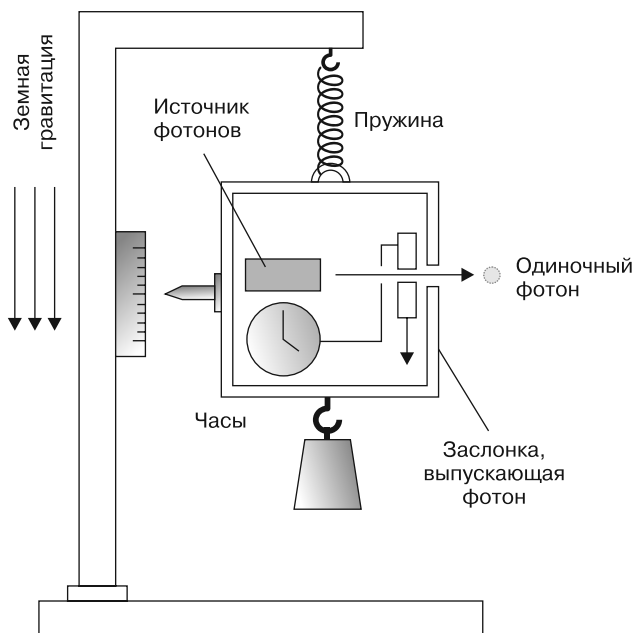
## Эйнштейн — Бору: «Бог не играет в кости»

Занимаетесь вы наукой или нет, вы, вероятно, слышали знаменитую фразу Эйнштейна «Бог не играет в кости». Эйнштейн написал это Нильсу Бору, когда обсуждал с ним природу квантовой механики. Бор считал, что понятия пространства — времени неприменимы на микроуровне. Однако Эйнштейн твердо верил в ткань пространства — времени и думал, что эту идею можно распространить на атомный масштаб. В сущности, это было корнем разногласий между ними.

Эйнштейн предположил, что свойства элементарной частицы можно измерить, не нарушая ее равновесия. Но эта идея противоречила интерпретации Бора — Гейзенберга. Два гиганта встретились на собрании

величайших физиков того времени в Брюсселе в 1927 году, где Эйнштейн пытался раз и навсегда доказать, что неопределенность не управляет реальностью.

Эйнштейн призвал Бора провести серию мысленных экспериментов, чтобы опровергнуть принцип неопределенности. В первом раунде Эйнштейн разработал коробку, которая, по его мнению, сможет зарегистрировать точный момент, когда частица света испускается из небольшого отверстия в боковой части коробки, и в то же время измерить ее вес (рис. 1.8).



**Рис. 1.8.** Экспериментальная коробка Эйнштейна, призванная опровергнуть принцип неопределенности

В мысленном эксперименте, представленном на рис. 1.8, в коробке есть источник света с часами, предназначенными для измерения точного времени испускания фотона. В то же время коробка с грузом на дне и соответствующим измерительным устройством висит на пружине. Идея была

проста: взвесить коробку до и после испускания фотона и одновременно зарегистрировать точное время с помощью часов. Уровни энергии можно легко рассчитать, используя уравнение Эйнштейна  $E = mc^2$ . В этот момент решалась судьба принципа неопределенности. Если эксперимент корректен, принцип неопределенности будет опровергнут и квантовая теория потерпит поражение.

Бор немедленно принялся за работу, пытаясь убедить Эйнштейна в том, что если его устройство сработает, то это будет означать конец физики. Бор в конце концов победил, заявив, что Эйнштейн не принял во внимание собственную теорию: часы подвержены влиянию гравитации, что приводит к неопределенности во время измерения. Он доказал верность расчета соотношения неопределенности для энергии и времени  $\Delta E \Delta t \geq h$ , используя уравнение Эйнштейна и формулу для красного смещения.

С этим результатом первый раунд остался за Бором, однако это было еще не все. Эйнштейн верил в полную картину физической реальности, и на его пути стоял принцип неопределенности. Он вернется с еще большим вызовом.

## **Бор — Эйнштейну: «Не говори Богу, что ему делать»**

«Бог не играет в кости» — это непоколебимый принцип Эйнштейна. Твердая вера в то, что реальность существует независимо от кого бы то ни было. Когда Эйнштейн написал Бору, что Бог не играет в кости, тот ответил: «Не говори Богу, что ему делать». Это подготовило почву для нового раунда противоборства между ними в попытке выяснить, что удерживает ядро от распада. К середине 1930-х годов, когда происходили эти события, общая теория относительности и квантовая теория уже были широко признаны как самые убедительные идеи, объясняющие устройство мира. Второй раунд посвящен наиболее парадоксальному аспекту квантовой теории — тому, что микрочастицы остаются связанными друг с другом даже на больших расстояниях.

## Запутанность и ЭПР-парадокс: мистическое дальноедействие

Поначалу считалось, что свет ведет себя как волна, но Эйнштейн доказал, что свет также демонстрирует поведение таких частиц, как фотоны. То же самое относится и к атомам. Они вели себя и как частицы, и как волны, в зависимости от используемого измерительного прибора. Кроме того, оба условия были необходимы для получения полной картины — Бор назвал это взаимодополняемостью.

Так как же понимать материю с учетом этих двух противоречий? Бор считал, что атом как таковой существует вне нашего восприятия. Это было больше, чем Эйнштейн мог принять с его верой в идею пространства — времени в основе всей физической реальности и желанием распространить данную концепцию на микроуровень. Бор же думал, что пространство и время бессмысленны, а реальность непознаваема и все, что у нас есть, — это явления.

Примерно в это же время Эйнштейн бросает второй и последний вызов Бору. В статье, написанной в соавторстве с коллегами Подольским и Розеном, Эйнштейн задает вопрос: дает ли квантовая механика полное описание физической реальности? Он предлагает мысленный эксперимент, в котором две частицы, испускаемые одним источником, имеют общие свойства и становятся разделенными. Тогда должна существовать возможность измерить первую частицу и получить информацию о второй, не нарушая ее равновесия. Цель эксперимента состояла в том, чтобы продемонстрировать абсурдность представления Бора о частицах, которые ведут себя по-разному из-за измерительного устройства. Согласно принципам квантовой механики измерение первой частицы будет влиять на другую во времени и пространстве.

Теперь представьте, что частицы должны быть разделены очень большими расстояниями, например находятся на разных концах Вселенной. Это создаст парадокс, нарушающий фундаментальный научный принцип причины и следствия. Существовало мнение, что все события имеют причину и следствие и информация не может передаваться быстрее скорости света, являющейся конечным пределом скорости во Вселенной. Эйнштейн

назвал это принципом локальности. Данный парадокс будет известен как парадокс Эйнштейна — Подольского — Розена, или ЭПР.

Как только Бор узнал о статье, он немедленно прекратил все работы. Задача должна была быть решена. Бор поначалу неохотно, но все же дал ответ, заявив, что обе частицы должны рассматриваться как единая система. Другими словами, эти частицы запутываются, а пространство и время в такой системе не имеют смысла. Поэтому картина микромира была непостижимой.

Эйнштейн назвал эффект запутанных частиц, отстоящих друг от друга на большие расстояния, мистическим дальнодействием<sup>1</sup>. Разногласия между исследователями никак не разрешались. Но в 1965 году физик Джон Белл совершил открытие, которое всех примирило.

## Неравенство Белла: проверка запутанности

Белл предложил ряд неравенств, чтобы обеспечить экспериментальное доказательство существования локальных скрытых переменных. Формально теорема Белла о неравенстве гласит: *никакая физическая теория локальных скрытых переменных никогда не сможет воспроизвести все предсказания квантовой механики*. Существует простой способ понять эту очень важную теорему, используя простые средние. Рассмотрим поляризацию фотонов (колебания света в определенной плоскости) под углами  $A = 0^\circ$ ,  $B = 120^\circ$  и  $C = 240^\circ$  (рис. 1.9).

Согласно теореме Белла если реальность не зависит от наблюдения, то фотон имеет определенные одновременные значения для этих трех настроек поляризации и они должны соответствовать восьми случаям, показанным в табл. 1.2.

---

<sup>1</sup> В немецком тексте Эйнштейна оно названо *spukhafte Fernwirkung*, в английском переводе Макса Борна — *spooky action at a distance*. Иногда используется обозначение «жуткое дальнодействие» или «призрачное дальнодействие». Мы будем придерживаться варианта «мистическое дальнодействие». — *Примеч. науч. ред.*

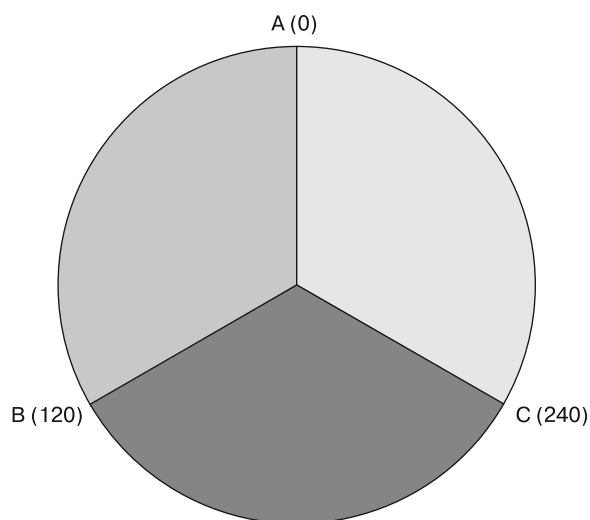


Рис. 1.9. Поляризация света под тремя углами

Таблица 1.2. Перестановки для поляризации фотонов под тремя углами

Номер	$A (0^\circ)$	$B (120^\circ)$	$C (240^\circ)$	$[AB]$	$[BC]$	$[AC]$	Сумма	Среднее
1	$A+$	$B+$	$C+$	1 (++)	1 (++)	1 (++)	3	1
2	$A+$	$B+$	$C-$	1 (++)	0	0	1	1/3
3	$A+$	$B-$	$C+$	0	0	1 (++)	1	1/3
4	$A+$	$B-$	$C-$	0	1 (--)	0	1	1/3
5	$A-$	$B+$	$C+$	0	1 (++)	0	1	1/3
6	$A-$	$B+$	$C-$	0	0	1 (--)	1	1/3
7	$A-$	$B-$	$C+$	1 (--)	0	0	1	1/3
8	$A-$	$B-$	$C-$	1 (--)	1 (--)	1 (--)	3	1

Теперь задайте простой вопрос: если мы измерим поляризацию под любым углом, какова вероятность того, что поляризация у любого соседа будет такой же, как у первого? Также рассчитайте суммарное и среднее значения поляризации. В табл. 1.2 поляризация соседей представлена столбцами  $AB$ ,  $BC$  и  $AC$ . Знаки «+» и «-» в столбцах  $A$ ,  $B$  и  $C$  указывают на положительную или отрицательную поляризацию под заданными



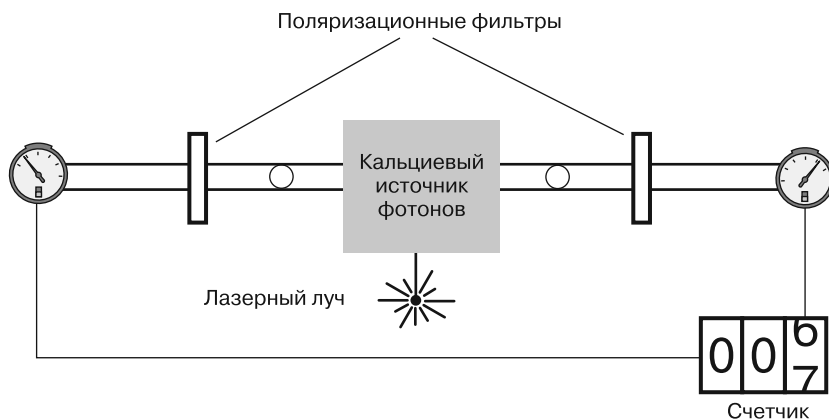
углами. Обратите внимание на то, что существует восемь возможных перестановок, обозначенных номерами строк. Таким образом, если мы находим одинаковую поляризацию (одинаковый знак) для двух соседей, то записываем 1, а также знак в столбцах *AB*, *BC* или *AC*. Это необходимо для расчета суммарного и среднего значений для соответствующей строки в таблице перестановок.

Теперь, если, как утверждает Эйнштейн, существует поляризация, независимая от измерения (локальности), тогда вероятность этой поляризации должна быть  $\geq 1/3$ . С другой стороны, если Бор прав, а реальность определяется актом наблюдения, то вероятность поляризации будет  $< 1/3$ . Это лежит в основе неравенства Белла. Белл не принимает чью-либо сторону: это не говорит о корректности одного или другого, но дает средства для поиска истины экспериментальным путем. В 1982 году французский физик Ален Аспе провел эксперимент, который раз и навсегда доказал, что Бор был прав с самого начала.

## **ЭПР-парадокс разгромлен: Бор смеется последним**

В эксперименте Аспе лазером облучается кальциевый источник, в результате чего создается пара фотонов, одновременно движущихся в противоположных направлениях. Фотоны проходят через поляризационный фильтр, который пропускает только фотон, поляризованный в одной плоскости. При прохождении фотона результат записывается измерительными устройствами, расположенными с обеих сторон. Измерительные приборы подключены к счетчику, который регистрирует результаты множества взаимодействий (рис. 1.10).

Когда оба поляризационных фильтра были откалиброваны в одном направлении, Аспе обнаружил корреляцию между парами фотонов. Они либо пройдут, либо будут заблокированы одновременно. Эта корреляция согласуется с мнением Эйнштейна о том, что фотон обладает свойством поляризации, предопределенным в момент излучения источником, а не в момент измерения, как предсказывала квантовая механика.



**Рис. 1.10.** Эксперимент Алена Аспе по проверке неравенства Белла, первый этап

В то же время если настройки поляризации фильтров различаются, то следует ожидать, что определенный минимальный процент фотонов либо пройдет, либо будет заблокирован. Здесь в игру вступает неравенство Белла, как показано в табл. 1.2.

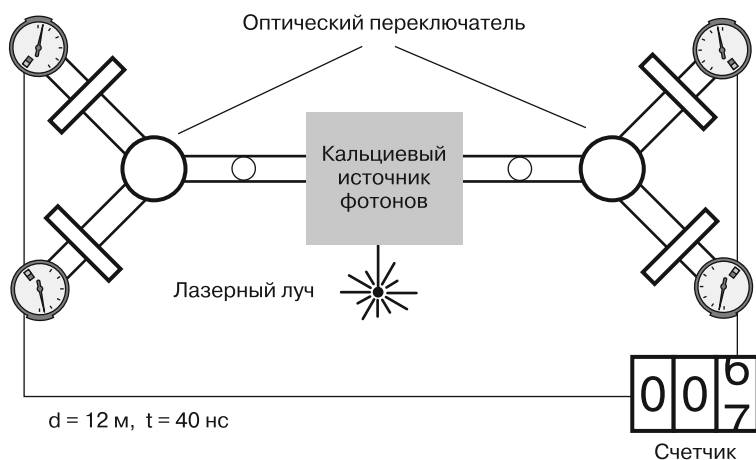
- Если процент проходящих или блокируемых фотонов больше ожидаемого минимума или равен ему, неравенство Белла сохраняется и поляризация фотонов определяется в момент излучения (победа достается Эйнштейну, а квантовая механика терпит поражение).
- Если процент меньше ожидаемого минимума, неравенство Белла нарушается и квантовая физика верна. Поляризация определяется в момент измерения (побеждает Бор, и квантовая механика спасена).

Аспе выполнил измерения для множества пар фотонов при разных настройках поляризации. Результаты были поразительными: измерения нарушали неравенство Белла, таким образом, невозможно было заранее определить поляризацию в момент излучения. Квантовая механика корректна! Оказалось, фотоны выбирали общую поляризацию в момент измерения. Может ли существовать какой-то неизвестный сигнал между фотонами, говорящий им, что нужно выбрать общее значение в момент измерения?

Теория относительности Эйнштейна гласит, что ни один сигнал не может распространяться быстрее скорости света — конечного предела скорости во Вселенной. Он назвал этот мнимый одновременный сигнал мистическим дальнодействием. Ален Аспе хотел проверить это утверждение на втором этапе своего эксперимента.

Физик использовал два оптических переключателя, которые разветвляются на два отдельных поляризационных фильтра, присоединенных к измерительному устройству каждый (рис. 1.11). Как и прежде, все измерительные приборы подключены к счетчику для сбора результатов.

- Оптический переключатель предназначен для передачи фотона в одном из двух направлений с чрезвычайно высокой частотой — 2 нс.
- Расстояние между концами экспериментальной установки было равно 12 м. Для скорости света ( $3 \cdot 10^8$  м/с) переход от одного конца к другому составляет 40 нс.



**Рис. 1.11.** Эксперимент Алена Аспе для проверки мистического дальнодействия

Теперь, если никакой сигнал не может распространяться быстрее скорости света, как гласит теория относительности, одному фотону потребуется свыше 40 нс, чтобы сообщить другому, какое значение поляризации

выбрать. Поскольку оптический переключатель изменяется с большей скоростью (2 нс), корреляция между фотонами не должна сохраняться. То есть фотоны не должны быть в состоянии выбрать одинаковую поляризацию в момент измерения (никаких мистических дальнодействий). В то же время, если корреляция сохраняется, все становится очень странным, так как какой-то сигнал передается на оба фотона быстрее скорости света.

Невероятно, но корреляция находилась в полном согласии с квантовой механикой, раз и навсегда доказав, что значение поляризации было выбрано одновременно обоими фотонами в момент измерения быстрее скорости света. Это означало, что расстояние между фотонами могло быть бесконечно большим, например от одного конца Вселенной до другого, и, что звучит еще более пугающе, такую корреляцию можно рассматривать как движение во времени: из настоящего в прошлое и наоборот!

## **Реальность дурачит нас: все взаимосвязано?**

Эксперимент Аспе доказывает, что существуют квантовые запутывания. И если мы хотим объяснить их, а не просто принять, то должны признать: корреляции<sup>1</sup> происходят быстрее скорости света. Раз некоторым это трудно осмыслить, все становится еще более странным. В телевизионном интервью для BBC физик Джон Белл сказал: «Мы никак не можем это использовать, например, мы не можем отправлять сообщения или информацию быстрее скорости света. И этот факт предсказывает квантовая механика. Кажется, что природа подшучивает над нами: за кулисами происходят необычные вещи, которые мы не можем использовать».

---

<sup>1</sup> Автор иногда пишет, что фотоны «обмениваются» или «взаимодействуют» быстрее скорости света. Но это не совсем так: между ними не происходит никакого взаимодействия и нет нарушения теории относительности. Квантовая механика фундаментально нелокальна — волновая функция может быть любых размеров, и на нее не действуют обычные правила распространения взаимодействий или информации. По сути, просто нельзя говорить о двух разных фотонах в запутанной паре, которые «взаимодействуют», — это некий новый объект, который описывается «размазанной» в пространстве волновой функцией. — *Примеч. науч. ред.*

В конце концов, Бор и Эйнштейн так и не разрешили свои разногласия. Они оба ушли в мир иной, но их наследие продолжает существовать. Пролистывая страницы их увлекательных биографий, нельзя не задаться вопросом: как бы чувствовал себя Бор, глядя на результаты эксперимента Алена Аспе, доказывающие, что он был прав с самого начала? Был бы он счастлив от своего триумфа над Эйнштейном? Было ли все это противостоянием двух эгоцентричных гениев, пытающихся доказать, кто лучше? Как вы думаете? Я предпочитаю верить, что это была борьба на благо науки. В общем, победителем в столкновении двух титанов стало все человечество.

# 2

## **Квантовые вычисления: искривление ткани самой реальности**

Полупроводники прошли долгий путь со времен вакуумной трубки. Трудно поверить, что сегодня размер транзистора составляет около 14 нм, то есть близок к размеру молекулы. В этой главе вы узнаете о происхождении квантовых вычислений, начиная с транзистора. Складывается впечатление, что технология производства полупроводниковых приборов и транзисторов противоречит законам физики. Далее углубленно рассмотрим базовый компонент квантового компьютера — кубит, включая странные эффекты суперпозиции, запутывания и управления кубитом с помощью логических элементов. Разработка кубитов — важная тема, и в этой главе описываются ведущие прототипы от крупных ИТ-компаний с учетом плюсов и минусов каждого из них.

Вы также узнаете о том, как квантовые компьютеры выдерживают конкуренцию с традиционными компьютерами. В настоящее время у квантовых дела идут не слишком гладко, но в ближайшие несколько лет ситуация изменится. Тем не менее у них есть некоторые недостатки: они нестабильны и склонны к ошибкам. Нам предстоит выяснить почему. В этой главе также обсуждаются весьма интересные поиски так называемого квантового превосходства (quantum supremacy). Между ИТ-гигантами идет жесткая борьба, победителя в которой не видно.

Еще одной темой обсуждений стал спорный алгоритм квантового отжига, а также его отличие от стандартного подхода на основе квантовых вентилей, используемого на протяжении всей этой книги.

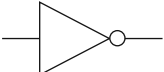
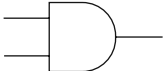


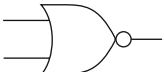

Закончим главу рассмотрением истории появления универсальных квантовых компьютеров, включая достижения всех основных поставщиков: в ближайшей перспективе квантовые компьютеры появятся во многих центрах обработки данных. В долгосрочной перспективе будущее видится блестящим, поскольку значительные ресурсы вкладываются в такие области, как аэрокосмическая промышленность, медицина, искусственный интеллект и т. д. Гонка становится глобальной. Итак, начнем.

## Транзистор вступает в противоречие с законами физики

Вы когда-нибудь заглядывали в свой домашний компьютер просто из любопытства, чтобы увидеть, из чего он состоит? По сути, это материнская плата со всевозможными электронными устройствами, а в центре находится большой черный квадрат — процессор. В зависимости от того, какой у вас ПК, может быть несколько процессоров, графических процессоров (GPU), аудио- и сетевых карт и всевозможных модульных компонентов. Все они состоят из миллионов транзисторов, которые являются основным строительным элементом многих электронных устройств. Транзистор — это, по сути, крошечный переключатель с положениями «включено/выключено», позволяющий электронам либо проходить, либо нет. Это свойство используется для кодирования 0 или 1 — основы двоичной системы счисления, применяемой во всей электронике.

Транзисторы объединяются для создания логических элементов (табл. 2.1). Они выполняют основные арифметические действия: сложение, вычитание, умножение и деление. Эти простые операции обеспечивают всю мощь, которая нам необходима для запуска высокоэффективных научных симуляций, игр, шифрования данных, просмотра веб-страниц, отправки электронных писем друзьям и т. д.

**Таблица 2.1.** Основные логические элементы

Тип	Символ	Описание	Таблица истинности		
НЕ		Инвертирует ввод (логическое отрицание)	A	$\sim A$	
			0	1	
			1	0	
И		Логическое произведение (конъюнкция)	A	B	A И B
			0	0	0
			0	1	0
			1	0	0
			1	1	1
ИЛИ		Логическое дополнение (дизъюнкция)	A	B	A ИЛИ B
			0	0	0
			0	1	1
			1	0	1
			1	1	1
И-НЕ		Инвертирует логическое произведение (отрицание конъюнкции)	A	B	A И-НЕ B
			0	0	1
			0	1	1
			1	0	1
			1	1	0
ИЛИ-НЕ		Инвертирует логическое дополнение (отрицание дизъюнкции)	A	B	A ИЛИ-НЕ B
			0	0	1
			0	1	0
			1	0	0
			1	1	0
ИЛИ-ИЛИ		Исключающее ИЛИ: выход ИЛИ-ИЛИ для двух входных элементов равен 1, только когда оба входных значения различны, и 0, если они одинаковы	A	B	A ИЛИ-ИЛИ B
			0	0	0
			0	1	1
			1	0	1
			1	1	0

Появление транзисторов стало причиной огромного технологического прогресса в нашем обществе. Они повсюду: в компьютерах, устройствах связи, медицинском оборудовании, аэрокосмической технике и т. д. О каком бы



устройстве вы ни подумали, вполне возможно, что она сделана на основе транзисторов. Но технология их развития вот-вот столкнется с непреодолимым препятствием — законами физики, в частности квантовой механики.

## Пятинанометровый транзистор: большая проблема

С 1960-х годов вычислительная мощность традиционных компьютеров возрастала экспоненциально, в то время как сами они становились все меньше и меньше. Сегодня компьютеры состоят из миллионов транзисторов, но, как только транзистор начинает приближаться по размеру к атому, вступает в действие странный мир квантовой механики и прежние правила теряют силу.

Рассмотрим рис. 2.1 и 2.2, на которых показаны размеры полупроводников, выпускаемых в период с 1970-х по 2020-е годы. С приблизительно 10 мкм в 1970-х годах размеры постоянно уменьшаются и составляют примерно 1 мкм к концу 1980-х годов (табл. 2.2, 2.3). Более того, с 1990-х годов наблюдается огромный спад кривой в нанометровом масштабе (1 нм =  $10^{-9}$  м) (см. рис. 2.2). Мы говорим о транзисторах, приближающихся по величине к молекулам. К 2019 году размер транзистора составляет около 5 нм. При таком масштабе странные свойства квантовой механики могут начать сеять хаос в классическом компьютере.

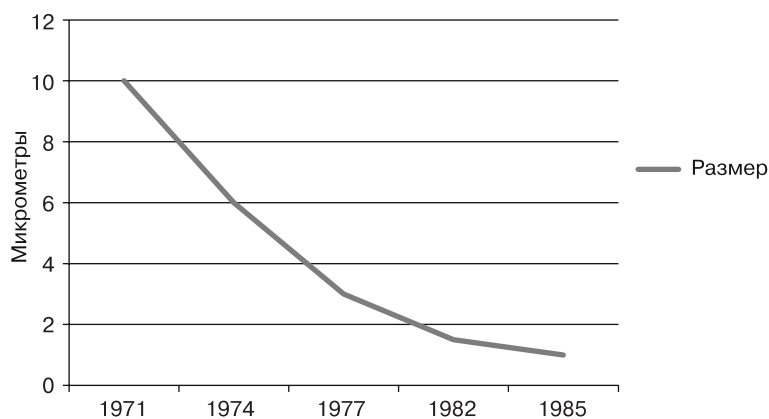
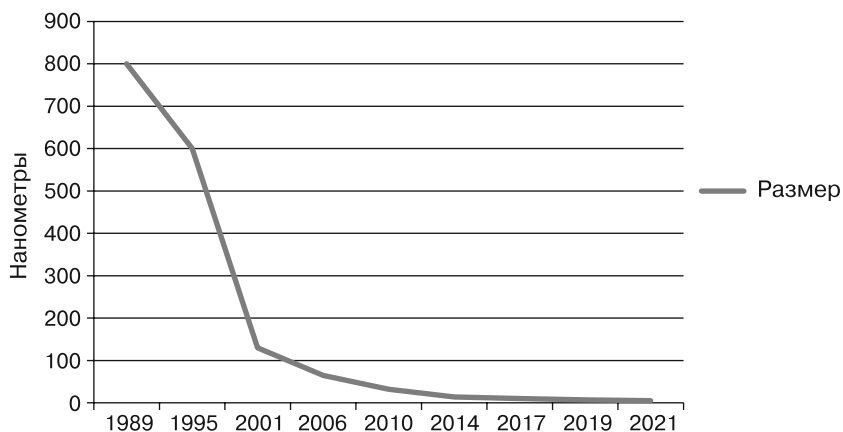


Рис. 2.1. Размеры полупроводников с 1970-х по 1980-е годы

**Таблица 2.2.** Данные о размерах полупроводников для рис. 2.1

Год	Размер, мкм
1971	10,0
1974	6,0
1977	3,0
1982	1,5
1985	1,0

**Рис. 2.2.** Размеры полупроводников с 1990-х годов**Таблица 2.3.** Данные о размерах полупроводников для рис. 2.2

Год	Размер, нм
1995	600
2001	130
2010	32
2014	14
2019	7
2021	5

На рис. 2.3 показан размер транзистора в 2020 году (около 5 нм) в сравнении с молекулой воды (0,275 нм). К сожалению, транзисторы не могут уменьшаться вечно. Существует порог, который называется квантовым масштабом. Если размер полупроводника будет меньше значения этого порога, классические компьютеры окажутся бесполезными.

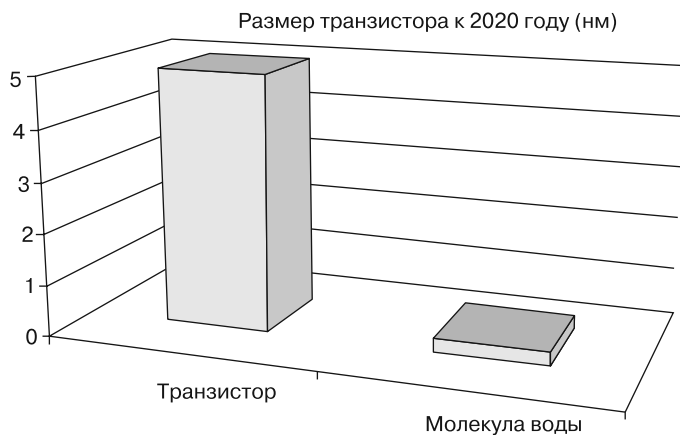


Рис. 2.3. Размер транзистора в сравнении с размером молекулы воды

## Квантовый масштаб и конец эпохи транзисторов

Возможно, конец эпохи транзисторов — это преувеличение. Тем не менее квантовый масштаб и его влияние таковыми не являются. В физике квантовый масштаб — это расстояние, на котором квантово-механические эффекты проявляются в изолированной системе. Эта странная граница существует при 100 нм и менее или при очень низкой температуре. Формально квантовый масштаб — это расстояние, на котором квантуется действие или момент импульса.

Квантовые эффекты проявляются в микроскопических масштабах, приводя к проблемам в современной микроэлектронике. Наиболее типичные эффекты — туннелирование электронов и интерференция, как показано в двухщелевом эксперименте.

### Туннелирование электронов

Электронное туннелирование, также известное как квантовое туннелирование, представляет собой явление, когда частица проходит через барьер, который невозможно преодолеть в классическом масштабе. Это создает проблемы для транзистора, и вот почему.

Предположим, у нас есть частица с энергией  $E$ , пытающаяся преодолеть барьер высотой  $V$ . Согласно классическому закону сохранения энергии частице нужно обладать энергией  $E > V$ , чтобы пройти через него, то есть кинетическая энергия частицы должна быть больше, чем потенциальная энергия  $V$  (рис. 2.4).

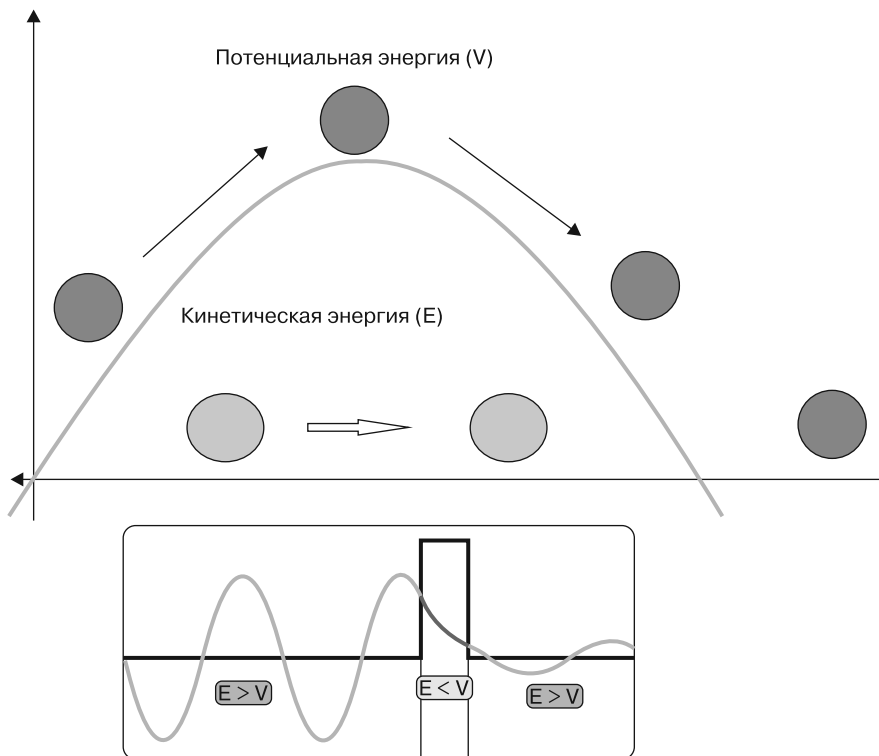


Рис. 2.4. Квантовое туннелирование в действии

#### ПРИМЕЧАНИЕ

Электронное туннелирование может предвещать конец эпохи транзисторов, но там, где один теряет, другой находит. Это важное свойство привело к разработке сканирующего туннельного микроскопа, что значительно повлияло на химические, биологические и материаловедческие исследования.

На рис. 2.4 показаны эффекты классической механики, а также квантового туннелирования. Согласно принципам квантовой механики существует вероятность того, что электрон пройдет через барьер, даже если его кинетическая энергия меньше, чем потенциальная энергия барьера ( $E < V$ ). Это связано с принципом неопределенности Гейзенберга. Из предыдущей главы вы узнали о двойственном поведении фотонов и других частиц, демонстрирующих свойства как волн, так и частиц. Для волн действует волновая функция Шредингера, для частиц Нильс Бор описал изменения в состоянии атома, когда тот получает или теряет энергию (квантовые переходы). Принцип неопределенности устраняет разрыв, вводя вероятность положения и импульса частицы в данный момент времени.

Когда частица, такая как электрон или фотон, приближается к барьеру, такому как транзистор, существует вероятность того, что она пройдет прямо через него. Это связано с тем, что ее волновая функция уменьшается от синусоидальной до экспоненциальной формы, и ее решением становится уравнение:

$$\psi = Ne^{-\beta x}; \quad (2.1)$$

$$P = \exp\left(-\frac{4a\pi}{h}\sqrt{2m(V-E)}\right), \quad (2.2)$$

где:

- $\psi$  — убывающая волновая функция Шредингера;
- $N$  — константа нормализации;
- $\beta = \sqrt{2m(V-E)}/\hbar$ ;
- $m$  — масса частицы;
- $V$  — потенциальная энергия,  $E$  — кинетическая энергия;
- $\hbar$  — постоянная Планка =  $6,626 \cdot 10^{-34}$  м<sup>2</sup> · кг/с;
- $a$  — толщина барьера.

Согласно Энгелю<sup>1</sup>, вероятность  $P$  того, что частица пройдет через барьер, может быть рассчитана по формуле (2.2). Кроме того, чтобы

---

<sup>1</sup> Engel T. Quantum Chemistry and Spectroscopy. Upper Saddle River, N.J. Pearson, 2006.

произошло квантовое туннелирование, должны быть выполнены следующие условия:

- барьер тонкий и имеет конечную высоту;
- потенциальная энергия барьера превышает его кинетическую энергию ( $E < V$ );
- частица обладает волновыми свойствами, это говорит о том, что квантовое туннелирование применимо только к наноразмерным объектам, таким как электроны, фотоны и т. д.

Давайте немного развлечемся, рассчитав вероятности квантового туннелирования для различных размеров барьеров полупроводников, выпускаемых на данный момент. Далее приводятся упражнения, которые помогут вам лучше разобраться в этом процессе.

### Упражнение 2.1

Воспользовавшись любимым инструментом, например электронной таблицей Excel, рассчитайте вероятность квантового туннелирования для электрона по формуле (2.2), учитывая следующие значения:

- кинетическая энергия электрона  $E = 4,5$  эВ;
- квадратный барьер с потенциальной энергией  $V = 5$  эВ (помните: для того чтобы произошло квантовое туннелирование, должно соблюдаться условие  $E < V$ );
- постоянная Планка  $h = 6,626 \cdot 10^{-34}$  м<sup>2</sup>·кг/с;
- масса электрона  $m = 9,1 \cdot 10^{-31}$  кг.

Размер барьера, обеспечиваемого техпроцессом производства полупроводников, приведен в табл. 2.2 и 2.3 (менее 100 нм для 2000 года и позже).

---

#### ПРИМЕЧАНИЕ

Электрон-вольт (эВ) — основная единица энергии в квантовой механике;  $1 \text{ эВ} = 1,6 \cdot 10^{-19}$  Дж. Это значение потребуется для преобразования единиц при расчете вероятности.

---

## Решение 2.1

Я использовал электронную таблицу Excel, чтобы легко вычислить значения из таблицы. Итак, выберите ячейку в Excel и введите формулу (2.2). Помните, что часть  $(V - E)$  нужно уменьшить, умножив на  $1,6 \cdot 10^{-19}$  Дж/эВ. Таким образом, формула (2.2) в Excel принимает вид:

$$\text{EXP}((( -4 * D5 * 3.14 ) / ( 6.626E-34 ) ) * \text{SQRT}( 2 * ( 9.1E-31 ) * ( 5 - 4.5 ) * ( 1.6E-19 ) ) )).$$

В приведенной формуле ячейка D5 содержит размер барьера, а остальное — константы  $\pi = 3,14$ ,  $h = 6,626 \cdot 10^{-34}$ ,  $m = 9,1 \cdot 10^{-31}$  и  $1 \text{ эВ} = 1,6 \cdot 10^{-19}$  Дж. Добавив формулу, создайте новую таблицу с годом выпуска и размером барьера в нанометрах (см. табл. 2.2 и 2.3). Наконец, скопируйте логику формулы для данных из ячеек для всех лет и размеров барьеров (табл. 2.4).

**Таблица 2.4.** Вероятности туннелирования электронов для технологии производства полупроводников

Год	Размер барьера, м	Вероятность
1989	8E-07	0
2001	1,3E-07	0
2006	6,5E-08	6,5829E-205
2010	3,2E-08	3,0188E-101
2014	1,4E-08	1,053E-44
2017	1,00E-08	3,86767E-32
2019	7,00E-09	1,02616E-22
2021	5,00E-09	1,96664E-16
После	5E-10	0,026876484

Какие выводы можно сделать на основании этих данных?

- Вероятность оказалась низкой даже для техпроцесса при размере барьера 5 нм, ожидаемого в 2021 году ( $2E-16$ ). Помните, что для получения процентов данное значение нужно умножить на 100.

- При размере барьера около 500 пм начинаются некоторые странности. Вероятность составляет 0,0269, следовательно, существует 2,69%-ная вероятность того, что электрон пройдет через барьер. Это значит, например, что если вы отправляете закодированное сообщение, то 2,69 % бит будет потеряно! Ничего хорошего.

---

### ПРИМЕЧАНИЕ

Пикометр (пм) — это 1/1000 нм, или  $10^{-12}$  м.

---

## Упражнение 2.2

Напишите на любимом языке программирования крошечную программу для расчета вероятности, как показано в упражнении 2.1. Убедитесь, что результаты совпадают. Распечатайте результаты в виде стандартного вывода, как показано далее.

Quantum tunnelling probabilities for current semiconductor processes.

2001	1.30e-07	0.000e+00
2010	3.20e-08	2.684e-101
2014	1.40e-08	1.000e-44
2019	7.00e-09	1.000e-22
2021	5.00e-09	1.931e-16
Beyond	5.00e-10	2.683e-02

## Решение 2.2

В листинге 2.1 показана маленькая программа на Java, в которой рассчитывается вероятность для разных годов и размеров для существующей технологии производства полупроводников, как это сделано в упражнении 2.1.

**Листинг 2.1.** Программа на Java для расчета вероятности квантового туннелирования для технологии производства полупроводников с 2000 года

```
public class Quantum Tunnelling {  
  
    /** Постоянная Планка */  
    static final double K_PLANK = 6.626e-34;
```



```
/** Масса электрона (кг) */
static final double K_ELECTRON_MASS = 9.1e-31;

/** Электрон-вольт */
static final double K_EV = 1.6e-19;

/**
 * Энгелева вероятность квантового туннелирования *
 * @param size
 *      Размер барьера в метрах.
 * @param E
 *      Кинетическая энергия в электрон-вольтах (эВ).
 * @param V
 *      Потенциальная энергия в эВ.
 * @return Вероятность квантового туннелирования
 */
static double EngelProbability(double size, double E, double V) {
    if (E > V) {
        throw new IllegalArgumentException
            ("Potential energy (V) must be > Kinetic Energy (E)");
    }
    double delta = V - E;
    double p1 = ((-4 * size * Math.PI) / K_PLANK);
    double p2 = Math.sqrt(2 * K_ELECTRON_MASS * delta * K_EV);
    return Math.exp(p1 * p2);
}

/** Простой тест для существующей технологии производства
    полупроводников */
public static void main(String[] args) {
    try {
        /** Размеры барьеров для текущей технологии производства
            полупроводников */
        final double[] SIZES = { 130e-9, 32e-9, 14e-9, 7e-9, 5e-9,
            500e-12 };

        // Даты для вывода на экран
        final String[] DATES = { "2001", "2010", "2014", "2019",
            "2021", "Beyond" };

        final double E = 4.5; // Кинетическая энергия электрона (эВ)
        final double V = 5.0; // Потенциальная энергия (эВ)

        // Отображение...
        for (int i = 0; i < DATES.length; i++) {
            double p = EngelProbability(SIZES[i], E, V);
```

```

        System.out.println(String.format("%s\t%2.2e\t%2.3e",
            DATES[i], SIZES[i], p));
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Листинг 2.1 определяет функцию `EngelProbability`, которая принимает три аргумента: размер барьера ( $m$ ), кинетическую энергию частицы  $E$  (эВ) и потенциальную энергию  $V$  (эВ). В ней вычисляется формула (2.1), а затем возвращается вероятность. После этого основная программа просто проходит в цикле по массиву для разных годов выпуска полупроводников `String[] DATES = { "2001", "2010", "2014", "2019", "2021", "Beyond" }` и соответствующих размеров: `double[] SIZES = { 130e-9, 32e-9, 14e-9, 7e-9, 5e-9, 500e-12 }`. Данные отформатированы в виде таблицы для стандартного вывода.

### Упражнение 2.3

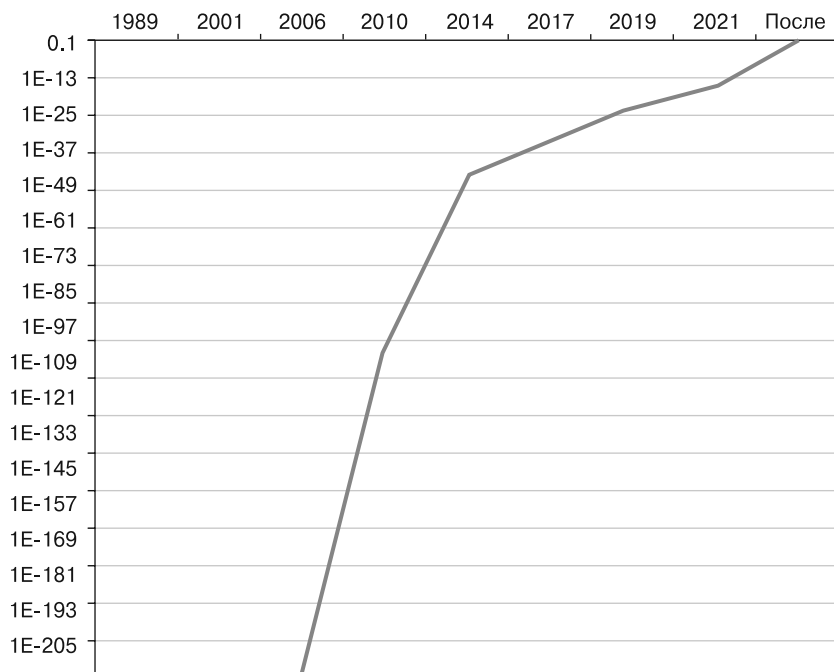
Отобразите данные, полученные в упражнениях 2.1 или 2.2, на графике, чтобы лучше представить ситуацию. И смело предсказывайте год кончины транзистора при существующей технологии производства полупроводников!

### Решение 2.3

Электронные таблицы — замечательный инструмент для обработки статистических значений. На основе приведенных ранее данных можно построить превосходную линейную диаграмму (рис. 2.5).

Итак, конец эпохи транзисторов должен наступить году примерно в... Я скептически отношусь к возможности предсказания точного года. Что я узнал о квантовой механике, так это то, что все управляется неопределенностью. Если предположить, что 1%-ная вероятность квантового туннелирования неприемлема для существующей технологии производства, то приведенные ранее данные покажут: примерно к 2025 году при размере барьера от 1 нм до 500 пм транзисторы и, следовательно, все компьютеры могут стать непригодными для использования. Хотя

я предполагаю, что транзисторы превратятся в нечто другое, возможно, во что-то органическое или еще более странное. Тем не менее пришло время на всякий случай начать учиться программировать квантовые компьютеры.



**Рис. 2.5.** Вероятности квантового туннелирования для технологии производства полупроводников

Теперь рассмотрим следующий квантовый эффект, вызывающий проблемы для транзистора: неопределенность положения или импульса, продемонстрированную в базовых экспериментах со щелями.

## Эксперименты со щелями

Эти эксперименты были проведены много десятилетий назад и предназначались для демонстрации странного мира квантовой механики. Они бывают разных видов: с одной щелью, с двумя щелями и др. В эксперименте

с одной щелью луч лазера проходит через вертикальную прорезь шириной несколько сантиметров и проецируется на поверхность. Ширину щели можно уменьшить. Как и ожидалось, на поверхности мы видим точку. Теперь, если ширина щели уменьшается, спроецированная точка становится *уже* и *уже* — это тоже ожидаемый результат. Но подождите, когда ширина щели достигает примерно 0,254 мм (1/100 дюйма), начинается нечто странное. Точка не становится *уже*, а расплывается в широкую горизонтальную линию. И это чрезвычайно нелогично.

---

#### ПРИМЕЧАНИЕ

Более подробное иллюстрированное описание эксперимента приведено в главе 1.

---

Когда речь идет о транзисторах, эксперименты со щелями важны, потому что они показывают странные эффекты квантовой механики в крошечных масштабах. В общем, ньютоновские и релятивистские законы времени и пространства не имеют смысла при таких масштабах и создадут проблемы для транзистора.

## Вероятное будущее транзисторов

Возможно, я слишком рано заговорил о конце эпохи транзисторов. На самом деле наука уже рассматривает возможные альтернативы и помимо квантовых вычислений.

Есть несколько интригующих проектов для решения этой проблемы.

- *Молекулярная электроника* — очень перспективная область. Она обещает выйти за рамки малых интегральных микросхем, используя молекулярные строительные блоки для изготовления электронных компонентов. Это междисциплинарная область, которая охватывает физику, химию и материаловедение.

- *Органическая электроника* — термин, который звучит одновременно захватывающе и фантастически. Это область материаловедения, рассматривающая разработку и применение органических молекул или полимеров, которые демонстрируют желаемые электронные свойства, такие как проводимость. Представьте себе транзисторы из органических материалов, таких как углерод. Не совсем живые машины, но уже близко.

## Ричард Фейнман и квантовый компьютер

Идею вычислительной системы, основанной на квантовых свойствах, высказал лауреат Нобелевской премии по физике Ричард Фейнман. В 1982 году он предложил квантовый компьютер, способный обратить эффекты квантовой механики в свою пользу<sup>1</sup>. С тех пор интерес к квантовым вычислениям носил в основном теоретический характер, но все должно было измениться. В 1995 году Питер Шор в своей общеизвестной работе «Алгоритмы простой факторизации и дискретного логарифмирования на квантовом компьютере за полиномиальное время»<sup>2</sup> описал алгоритм факторизации больших чисел для работы на квантовом компьютере. Это положило начало гонке за созданием реального квантового компьютера, для которого математически доказано, что временная сложность ( $O$ , или время выполнения) его алгоритма значительно меньше, чем у нынешнего лидера классических вычислений — метода решета числового поля.

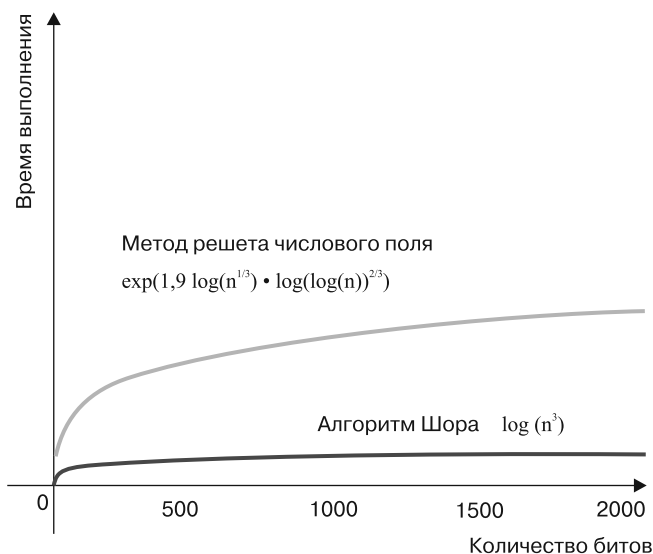
Алгоритм Шора сам по себе имеет глубокий смысл. Рассмотрите рис. 2.6, демонстрирующий временные сложности метода решета числового поля и алгоритма Шора в сравнении.

Математически было подсчитано, что алгоритм Шора может вычислить 232-значное целое число (RSA-232), одно из самых больших целых чисел,

<sup>1</sup> Quantum computation. David Deutsch. Physics World, 1/6/92. A comprehensive and inspiring guide to quantum computing.

<sup>2</sup> Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer // <https://arxiv.org/abs/quant-ph/9508027>.

за считанные секунды. Таким образом, действующий квантовый компьютер, который сможет реализовать алгоритм Шора, сделает бесполезным асимметричное шифрование. Имейте в виду, что асимметричное шифрование задействовано во всех областях современной жизни: например, в банках для шифрования личных данных и счетов клиентов, в Интернете для защиты аккаунтов пользователей и т. д.



**Рис. 2.6.** Сравнение временных сложностей метода решета числового поля и алгоритма Шора

Но пока еще рано бежать в банк, забирать все свои деньги и прятать их под матрасом.

Практическая реализация этого алгоритма отстоит на десятилетия от настоящего момента. Более подробно обсудим этот увлекательный алгоритм в главе 3.

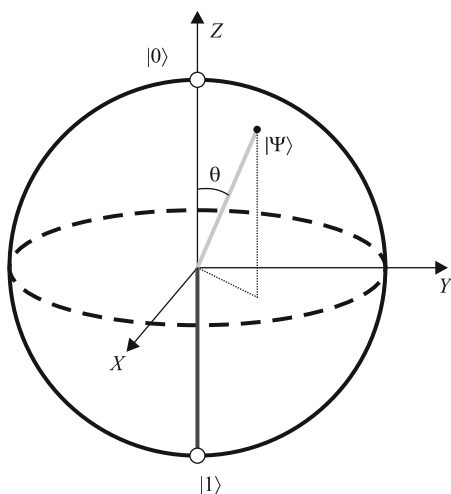
Теперь вернемся к Фейнману и его квантовому компьютеру. В классическом компьютере основной единицей является бит (0 или 1). В компьютере Фейнмана основной единицей является кубит, или квантовый бит, — единица столь же причудливая, как и основанная на ней теория.

## Кубит, странный и потрясающий одновременно

Как и его классический собрат, кубит может принимать значение либо 0, либо 1. Физически кубиты могут быть представлены как любая двухуровневая квантовая система наподобие:

- спина частицы в магнитном поле, где верх означает 0, а низ — 1;
- поляризации единичного фотона, где горизонтальная поляризация соответствует 1, а вертикальная — 0. Квантовый компьютер можно сделать из света. Разве это не странно?

В обоих случаях 0 и 1 — единственно возможные состояния. Геометрически кубит можно представить с помощью поверхности, носящей название сферы Блоха — инструмента, названного в честь шведского физика Феликса Блоха (рис. 2.7).



**Рис. 2.7.** Геометрическое представление квантового состояния с помощью сферы Блоха

Строго говоря, сфера Блоха — это геометрическое представление в трехмерном гильбертовом пространстве чистого состояния двухуровневой

квантовой системы или кубита. Ее северный и южный полюсы представляют собой стандартные базисные векторы  $|0\rangle$  и  $|1\rangle$  соответственно, которые, в свою очередь, соответствуют спине электрона, направленному вверх и вниз. Помимо основных векторов, сфера может иметь промежуточное значение, называемое *суперпозицией* и являющееся, по сути, вероятностью для 0 или 1. Хитрость в том, что мы не можем предсказать, какой она будет, за исключением момента наблюдения, когда вероятность коллапсирует в окончательное состояние.

## Суперпозиция состояний

Представьте себе, что можно было бы подбросить монету, которая могла бы упасть не только орлом или решкой, но и обеими сторонами одновременно. Такая монета открывала бы для нас большие возможности. Тем не менее здесь есть подвох: в тот момент, когда вы наблюдаете квантовую монету, ей приходится выбирать свое состояние: орел или решка, и никогда не известно, в каком положении она находилась раньше. Это одна из причин, по которой нужно быть осторожными при измерении кубитов, потому что они изменяются в момент наблюдения. В общем, *суперпозиция* меняет правила игры. Давайте посмотрим почему.

- Однобитовый классический компьютер в каждый отдельно взятый момент времени может находиться в одном из двух состояний (или хранить его), 0 или 1. Однокубитный квантовый компьютер может находиться (или хранить) в двух состояниях одновременно. Это  $2^1 = 2$ .
- Двухбитовый классический компьютер может хранить только одну из  $2^2 = 4$  возможных комбинаций. Двухкубитный квантовый компьютер может хранить  $2^2 = 4$  возможных значения одновременно.

С учетом того, что байт (8 бит) является базовой единицей, используемой для хранения информации в любой системе, число значений, которые могут быть сохранены одновременно в квантовом компьютере, будет равно  $2^n$ , где  $n$  — число кубитов. Сравните это с емкостью запоминающего устройства классического компьютера (табл. 2.5), и вы поймете, почему кубиты действительно мощны.



**Таблица 2.5.** Емкость запоминающего устройства (ЗУ) для кубитов

Биты/кубиты	Классическое ЗУ, байт	Квантовое ЗУ, бит	Квантовое ЗУ, байт
4	1	16	2
8	1	256	32
32	4	4 294 967 296	536 870 912
68	8	1,84467E+19	2,30584E+18

Таким образом, объем данных, которые могут храниться одновременно в квантовом компьютере, поражает настолько, что появился новый термин — *квантовое превосходство*. Это та точка, в которой квантовый компьютер сможет решить задачи, которые не по силам классическому компьютеру. Это понятие будет подробно обсуждаться в следующем разделе данной главы. А пока рассмотрим другое странное свойство кубита — запутанность.

## Запутанность: наблюдение за кубитом изменяет состояние его партнера

Много лет назад Альберт Эйнштейн назвал квантовую запутанность *мистическим дальним действием*. Хотите верить, хотите нет, но запутанность была доказана экспериментально французским физиком Аленом Аспе в 1982 году. Он продемонстрировал, как влияние на одну из двух коррелированных частиц передается со скоростью большей, чем скорость света!

---

### ПРИМЕЧАНИЕ

Ирония здесь в том, что люди не могут использовать запутанность, чтобы отсылать сообщения со скоростью больше скорости света, поскольку информация не может так быстро перемещаться. Это противоречие, а также эксперимент Алена Аспе более подробно описаны в главе 1.

---

Если набор кубитов запутан, то каждый из них мгновенно отреагирует на изменение другого, независимо от того, как далеко они находятся друг от друга (на противоположных сторонах Галактики, например, что звучит действительно невероятно). Это полезно в том смысле, что если мы измеряем свойства одного кубита, то можем сделать вывод о свойствах его партнера, не видя последнего.

Кроме того, запутанность можно измерить с помощью квантовой томографии. Квантовая томография рассчитывает вероятность измерения каждого возможного состояния системы.

---

#### ПРИМЕЧАНИЕ

Многокубитное запутывание представляет собой шаг вперед в реализации крупномасштабных квантовых вычислений. Это область активных исследований. В настоящее время физики в Китае экспериментально продемонстрировали квантовую запутанность с десятью кубитами в сверхпроводящей цепи<sup>1</sup>.

---

Запутывание — один из аспектов управления кубитом. Еще одна умопомрачительная особенность — управление им с помощью квантовых вентиляей.

## Управление кубитами с помощью квантовых вентиляей

Вентили являются базовыми строительными блоками квантового компьютера. Как и их классические аналоги, они оперируют набором входов для создания набора выходов. Но, в отличие от своих собратьев, они работают одновременно во всех возможных состояниях кубита, что делает их

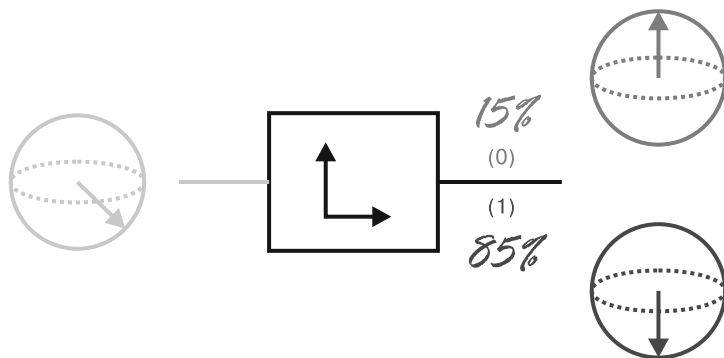
---

<sup>1</sup> *Chao Song et al.* 10-Qubit Entanglement and Parallel Logic Operations with a Superconducting Circuit // Physical Review Letters. DOI: 10.1103/PhysRevLett.119.180511.

действительно крутыми и странными одновременно. Далее рассмотрим основные вентили квантового компьютера.

## Измерительный элемент

Известно, что акт измерения или наблюдения кубита изменяет его состояние. Этот процесс также рассматривается как измерительный элемент (measurement gate). Измерительный элемент принимает кубит в суперпозиции состояний в качестве входных данных и выдает либо 0, либо 1. Вероятность того, что значением вывода будет 0 или 1, зависит от состояния, в котором изначально находится кубит (рис. 2.8).



**Рис. 2.8.** Измерительный элемент и вероятность его выходного значения

Обратите внимание на то, что измерение должно быть последним действием в квантовой цепи, поскольку согласно принципам квантовой механики наблюдение кубита при промежуточных вычислениях коллапсирует его волновую функцию и разрушит параллелизм, достигнутый суперпозицией состояний.

## Вентиль SWAP

Вентиль SWAP обмена состояниями принимает два кубита и меняет их состояния (рис. 2.9).

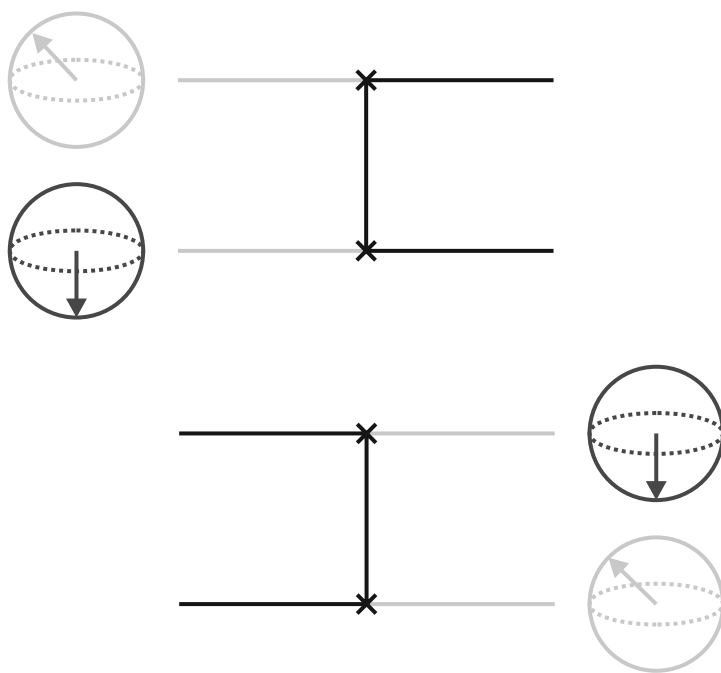


Рис. 2.9. Вентиль SWAP в действии

### Вентиль Паули X

Вентиль Паули X (рис. 2.10) — это квантовый аналог классического оператора НЕ. Строго говоря, он поворачивает кубит на  $180^\circ$  вокруг оси X. Обратите внимание на то, что ось X указывает за пределы экрана, как показано на сфере Блоха на рис. 2.7.

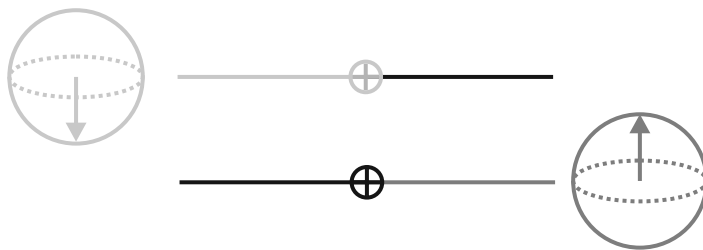


Рис. 2.10. Вентиль Паули X

### ПРИМЕЧАНИЕ

Вентиль Паули назван в честь одного из отцов квантовой физики австрийца по происхождению Вольфганга Эрнста Паули. В 1945 году он получил Нобелевскую премию по физике за разработку принципа исключения, или принципа Паули, в котором, по сути, говорится, что два электрона не могут существовать в одном и том же квантовом состоянии<sup>1</sup>. Работу Паули высоко оценивал Альберт Эйнштейн, кроме того, австриец был близким другом колоссов квантовой механики Нильса Бора и Бернарда Гейзенберга.

## Вентили Паули Y и Z

Вентили, осуществляющие поворот вокруг осей Y и Z, известны как вентили Паули Y и Z соответственно.

- Вентиль Паули Y воздействует на одиночный кубит. Он поворачивает его вокруг оси Y сферы Блоха на  $\pi$  радиан ( $180^\circ$ ) и преобразует  $|0\rangle$  в  $i|1\rangle$  и  $|1\rangle$  в  $-i|0\rangle$ .
- Вентиль Паули Z воздействует на одиночный кубит. Он поворачивает его вокруг оси Z сферы Блоха на  $\pi$  радиан ( $180^\circ$ ), оставляет базисное состояние  $|0\rangle$  неизменным и преобразует  $|1\rangle$  в  $-|1\rangle$ .

## Вентиль Адамара (H)

Вентиль Адамара воздействует на одиночный кубит. Он является комбинацией двух поворотов:

- на  $\pi$  радиан вокруг оси X;
- на  $\pi/2$  радиан вокруг оси Y.

---

<sup>1</sup> Nobel Lecture: Exclusion Principle and Quantum Mechanics Pauli's own account of the development of the Exclusion Principle // [www.nobelprize.org/nobel\\_prizes/physics/laureates/1945/pauli-lecture.html](http://www.nobelprize.org/nobel_prizes/physics/laureates/1945/pauli-lecture.html).

Вентиль Адамара является квантовым аналогом матрицы Адамара — квадратной матрицы, элементы которой принимают значения либо +1, либо −1, а строки являются взаимно перпендикулярными:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

ПРИМЕЧАНИЕ

Преобразование Адамара можно использовать при шифровании данных, а также во многих алгоритмах обработки сигналов и сжатия данных.

Управляемые вентили (сX сY сZ)


Управляемые вентили действуют на два и более кубита, где один или более из них являются управляющими для какой-либо операции. Например, управляемый вентиль НЕ (CNOT или сX) действует на два кубита и выполняет операцию НЕ на втором кубите только тогда, когда первый кубит имеет состояние  $|1\rangle$ , и оставляет его неизменным в противном случае.


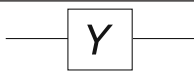
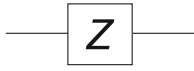
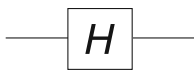
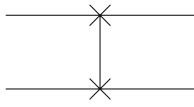
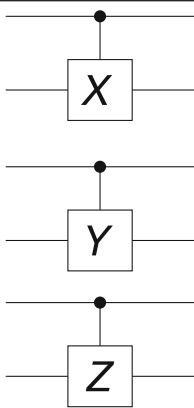
Вентиль Тоффоли (CCNOT)

Это управляемый вентиль, который оперирует тремя кубитами. Если первые два кубита находятся в состоянии  $|1\rangle$ , он применяет оператор Паули X (или НЕ) к третьему, иначе никаких действий не выполняет. Данный вентиль преобразует  $|a, b, c\rangle$  в  $|a, b, c + ab\rangle$ .

Квантовые вентили, приведенные в табл. 2.6, являются базовыми строительными блоками квантовых схем по аналогии с классическими логическими операторами из табл. 2.1, принятыми для цифровых схем.

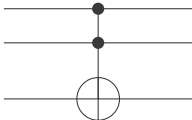
Таблица 2.6. Основные квантовые вентили

Вентиль	Условное обозначение	Описание
Измерительный		Принимает на вход кубит в суперпозиции и выдает либо 0, либо 1

Вентиль	Условное обозначение	Описание
X (NOT)		Вращает кубит на $180^\circ$ вокруг оси $X$ . Преобразует $ 0\rangle$ в $ 1\rangle$ и $ 1\rangle$ в $ 0\rangle$
Y		Производит поворот вокруг оси $Y$ сферы Блоха на $\pi$ радиан. Представлен матрицей Паули: $Y = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix},$ где $i = \sqrt{-1}$ — мнимая единица
Z		Производит поворот вокруг оси $Z$ сферы Блоха на $\pi$ радиан. Представлен матрицей Паули: $Y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Адамара		Представляет собой поворот на $\pi$ радиан вокруг оси $(X+Y)/\sqrt{2}$ . Иначе говоря, преобразует состояние $ 0\rangle$ в $( 0\rangle+ 1\rangle)/\sqrt{2}$ и $ 1\rangle$ в $( 0\rangle- 1\rangle)/\sqrt{2}$
SWAP (S)		Меняет состояния двух кубитов в соответствии с базисом $ 00\rangle,  01\rangle,  10\rangle,  11\rangle$ . Представлен матрицей: $S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Управляемый (сX сY сZ)		Действует на два и более кубита, где один кубит и более являются управляющими для какой-либо операции. В общем виде описывается матрицей: $C(U) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix},$ где $U$ — одна из матриц Паули, $\sigma_x$ , $\sigma_y$ или $\sigma_z$

Продолжение ⇨

Таблица 2.6 (продолжение)

Вентиль	Условное обозначение	Описание
Тоффоли (CCNOT)		<p>Это обратимый вентиль, что означает, что его выход можно восстановить по его входу (состояния перемещаются без увеличения физической энтропии). Имеет трехбитные входы и выходы, если для первых двух бит установлено значение 1, то инвертирует третий бит, в противном случае все биты остаются неизменными.</p> <p>Обратимые вентили важны, потому что они рассеивают меньше тепла. Когда логический элемент поглощает свои входные данные, информация теряется, так как на выходе меньше информации, чем на входе. Такие потери информации — это рассеивание энергии в окружающую среду в виде тепла. В квантовых вычислениях вентиль Тоффоли очень важен, так как квантовая механика требует, чтобы преобразования были обратимыми, и допускает более общие состояния (суперпозиции) вычислений, чем классические компьютеры</p>

Таким образом, квантовый вентиль управляет вводом суперпозиций, вращает вероятности и создает другую суперпозицию в качестве своего выхода. Физически кубиты могут создаваться различными способами, и в настоящее время промышленные предприятия начинают действовать в разных направлениях, каждое из которых имеет свои достоинства и недостатки. Рассмотрим их.

## Проектирование кубитов

Когда дело доходит до проектирования кубитов, только крупные компании могут вступить в гонку по созданию практического квантового компьютера. Ввиду странности и сложности квантовой механики эта задача не из



легких. В статье для журнала *Science Magazine*<sup>1</sup> писатель Габриэль Попкин в общих чертах обрисовал, какие действия предпринимают технологические гиганты. Кажется, что все они хотят сделать ставку на разные типы устройства. Пока явного победителя нет, и гонка продолжается. Далее рассмотрим самые распространенные типы кубитов, по словам Попкина.

## Сверхпроводящие контуры

Когда электрический ток проходит через проводник, часть энергии теряется в виде тепла и света. Это явление называется сопротивлением и зависит от типа материала: некоторые металлы, например медь и золото, хорошо проводят электричество и, следовательно, имеют низкое сопротивление. Ученые обнаружили, что чем холоднее материал, тем лучшим проводником электричества он становится. Таким образом, чем ниже температура, тем ниже сопротивление. Но независимо от того, насколько холодными будут золото или медь, они всегда будут показывать некоторый уровень сопротивления.

Ртуть в этом смысле отличается от других металлов. В 1911 году ученые обнаружили, что при ее охлаждении до 4,2 К (выше абсолютного нуля) сопротивление становится равным нулю. Этот эксперимент привел к открытию сверхпроводника — материала, который имеет нулевое электрическое сопротивление при очень низких температурах. С тех пор было обнаружено много сверхпроводящих материалов: алюминий, галлий, ниобий и др., которые демонстрируют нулевое сопротивление при критической температуре. Самое замечательное в сверхпроводниках заключается в том, что электрический ток в них протекает без каких-либо потерь, поэтому теоретически в замкнутом контуре он может циркулировать вечно.

---

### ПРИМЕЧАНИЕ

Этот принцип был доказан экспериментально, когда ученые смогли поддерживать электрический ток, протекающий через сверхпроводящие кольца, в течение многих лет.

---

<sup>1</sup> Ученые близки к созданию квантового компьютера, который сможет превзойти обычный: <http://www.sciencemag.org/news/2016/12/scientists-are-close-building-quantumcomputer-can-beat-conventional-one/>.

В кубите, изготовленном из сверхпроводящего контура, ток колебательно меняет направление с прямого на обратное. Микроволновое излучение побуждает ток перейти в суперпозицию состояний (рис. 2.11). Рассмотрим преимущества и недостатки такого устройства.

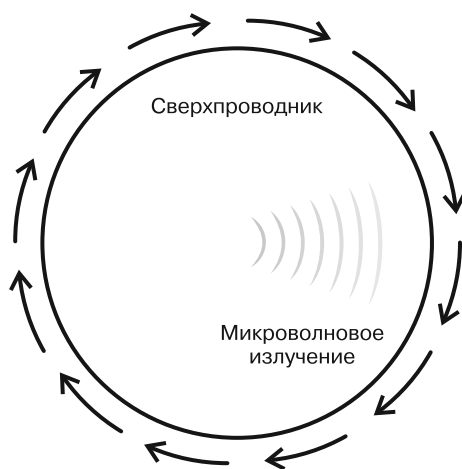


Рис. 2.11. Сверхпроводящий кубит

Преимущества:

- низкий уровень ошибок (около 99,4 % успешных логических операций);
- быстрый, изготовлен из существующего материала;
- приличное количество (девять) запутанных кубитов, способных выполнять двухкубитные операции.

Недостатки:

- малое время сохранения когерентности состояний — 0,00005 с. Это минимальное количество времени, в течение которого сохраняется суперпозиция состояний;
- должно находиться при сверхнизких температурах,  $-271^{\circ}\text{C}$ .

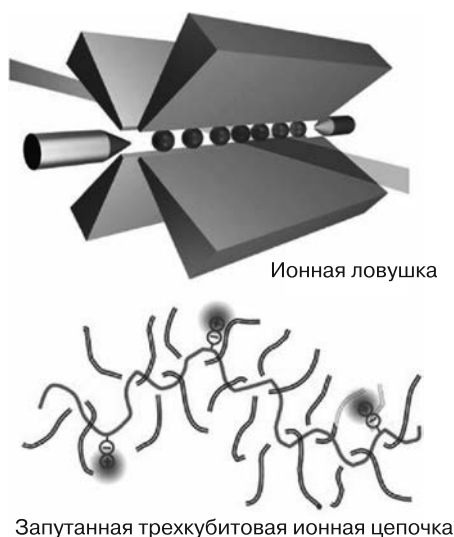
На таком устройстве работает облачная платформа IBM Q Experience, на базе которой создан код, используемый в этой книге. Применяется также

Google и частным предприятием Quantum Circuits, Inc. (QCI), которое стремится разработать практичный квантовый компьютер на основе сверхпроводников.

## Ионные ловушки

Ионная ловушка — это методика управления квантовыми состояниями в кубите. При этом задействуется комбинация электрических или магнитных полей для захвата заряженных частиц (ионов) в системе, изолированной от внешней среды. Лазеры применяются для создания состояний пар кубитов для одиночных операций или связи между внутренними состояниями и внешними динамическими состояниями для запутывания.

Применение ионных ловушек — это стремление воплотить в жизнь мечту о крупномасштабных универсальных квантовых вычислениях путем масштабирования с использованием *массивов ионных ловушек*. Данная методика позволяет также создавать большие запутанные состояния с помощью сетей с фотонной связью из запутанных удаленных друг от друга *ионных цепочек* или комбинации этих двух идей (рис. 2.12).



**Рис. 2.12.** Ионные ловушки и цепочки для крупномасштабных квантовых вычислений

Рассмотрим преимущества и недостатки ионных ловушек.

Преимущества:

- большое время сохранения когерентности состояний. Эксперты утверждают, что ионные ловушки могут сохранять запутанность более 1000 с, а это весьма существенно в сравнении со сверхпроводящими контурами (0,00005 с);
- более высокий уровень успешного выполнения (99,9 %), чем у сверхпроводников (99,4 %). Ненамного, но все же;
- наибольшее количество (14) запутанных кубитов, способных выполнять двухкубитные операции, на настоящее время.

Недостатки:

- медленное выполнение операций;
- требуется большое количество лазеров.

Лидером в разработке данной технологии является компания IonQ, расположенная в штате Мэриленд, США.

## **Кремниевые квантовые точки**

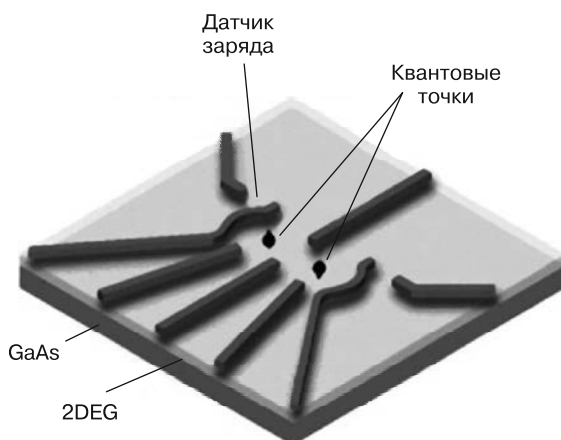
Intel, гигант в области создания ЦПУ для ПК, возглавляет этот проект. В кремниевой квантовой точке электроны вертикально ограничены основным состоянием квантовой ямы в арсениде галлия (GaAs), образуя двумерный электронный газ (2DEG). Электронный газ 2DEG может свободно перемещаться в двух измерениях, но сильно ограничен третьим (рис. 2.13). Такое жесткое ограничение приводит к появлению квантованных уровней энергии для движения в третьем направлении, что может представлять большой интерес для квантовых структур.

---

### **ПРИМЕЧАНИЕ**

На сегодняшний день найден способ получения 2DEG на транзисторах, изготовленных из полупроводников. Они также демонстрируют квантовые эффекты, такие как эффект Холла, в которых двумерная электронная проводимость квантуется при низких температурах и сильных магнитных полях.

---



**Рис. 2.13.** Квантовая точка в арсениде галлия

Перечислим преимущества и недостатки кремниевых квантовых точек.

Преимущества:

- стабильны, созданы на основе существующих полупроводниковых материалов;
- лучшее время сохранения когерентности состояний, чем у сверхпроводящих материалов, составляющее 0,03 с.

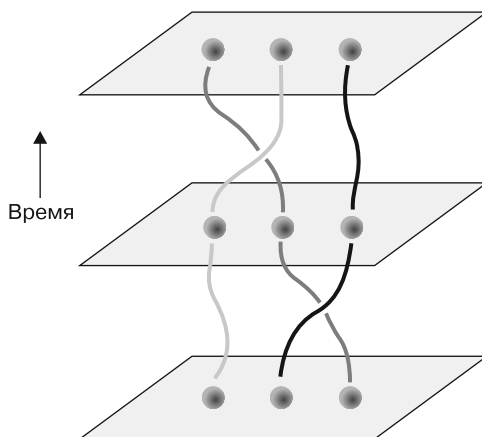
Недостатки:

- малое количество (два) запутанных кубитов, способных выполнять двухкубитные операции;
- уровень успешного выполнения ниже, чем у сверхпроводящих контуров и ионных ловушек, но все еще выше 99 %.

## Топологические кубиты

Топологические кубиты призваны устранить ошибки, характерные для квантовых компьютеров. Ошибки происходят из-за вероятностного характера квантовой механики и касаются времени сохранения когерентности состояний или продолжительности запутывания кубитов. Топологический кубит использует двумерные квазичастицы, называемые анионами, пути которых огибают друг друга, образуя косы в трехмерном

пространстве-времени. Эти косы образуют логические вентили, из которых состоит компьютер (рис. 2.14).



**Рис. 2.14.** Топологический кубит с косами, действующими как логические вентили

Преимущества:

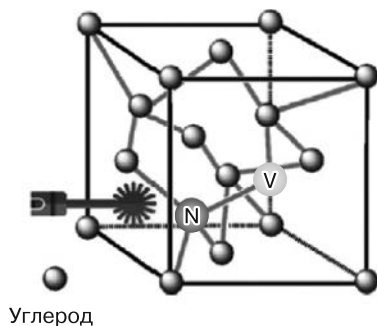
- стабильность;
- отсутствие ошибок (понятие времени жизни неприменимо).

Недостаток в том, что на данный момент это чисто теоретические разработки, хотя недавние эксперименты показывают, что такие элементы могут быть созданы в реальных условиях с использованием полупроводников, изготовленных из арсенида галлия при температуре, близкой к абсолютному нулю, и подвергнутых воздействию сильных магнитных полей. Microsoft и Bell Labs — в числе компаний, которые поддерживают данный проект.

## NV-центры в алмазе

NV-центры в алмазе — это места в кристаллической решетке алмаза, где должен находиться, но отсутствует атом углерода. NV-центры используют нанометровые атомные дефекты в алмазных материалах, чтобы функцио-

нирывать как кубиты. С помощью атомно-силовой микроскопии было обнаружено, что на поверхности природных алмазов присутствует несколько типов дефектов. Этот дефект, или вакансия, наряду с атомом азота добавляет электрон в решетку алмаза. Затем квантовый спин электрона можно контролировать с помощью лазера (рис. 2.15).



**Рис. 2.15.** Кубит на основе NV-центра в алмазе

По словам Дирка Энглунда и его коллег с кафедры электротехники и компьютерных наук Массачусетского технологического института, NV-центры в алмазе решают извечную проблему легкого считывания информации из кубитов. Алмазы являются естественными излучателями света, поэтому частицы света, испускаемые NV-центрами, сохраняют суперпозицию состояний, так что могут перемещать информацию между квантовыми вычислительными устройствами. Лучше всего они работают при комнатной температуре, не требуется охлаждение до  $-272\text{ }^{\circ}\text{C}$ !

По словам Энглунда, одним из подводных камней в случае NV-центров в алмазе является то, что они имеются только на 2 % поверхности алмаза. Поэтому исследователи разрабатывают процессы бомбардировки поверхности алмаза пучками электронов с целью создания большего количества NV-центров.

Преимущества:

- продолжительное сохранение когерентности состояний — 10 с;
- высокий уровень успешного выполнения — 99,2 %;

- приличное количество (девять) запутанных кубитов, способных выполнять двухкубитные операции;
- кубиты функционируют при комнатных температурах. Это просто невероятно.

Недостатки:

- малое количество NV-центров на поверхности материала — около 2 %;
- сложно произвести запутывание.

В целом со времен Ричарда Фейнмана квантовые компьютеры прошли долгий путь развития благодаря ведущим мировым компаниям, стремящимся заработать. Прямо сейчас сверхпроводниковые контуры занимают лидирующие позиции. Однако благодаря новым исследованиям, например, NV-центров в алмазе, есть шанс реализовать мечту о крупномасштабных квантовых вычислениях.

## Квантовые компьютеры в сравнении с традиционным аппаратным обеспечением

Квантовые компьютеры превосходят классические машины при решении определенных задач. Рассмотрим табл. 2.7, в которой показаны временные сложности двух конкретных задач для квантового и классического компьютеров.

**Таблица 2.7.** Сравнение временной сложности некоторых задач при их решении на квантовом и классическом компьютерах

Задача	Квантовый компьютер	Временная сложность	Классический компьютер	Временная сложность
Поиск	Алгоритм Гровера	$\sqrt{n}$	Быстрый поиск	$n / 2$
Факторизация больших чисел	Алгоритм Шора	$\log(n^3)$	Метод решета цифрового поля	$\exp(1,9 \log(n^{1/3}) \log(\log(n))^{2/3})$



Поиск по алгоритму Гровера обеспечивает лучшую производительность, чем традиционный. Это может сильно повлиять на центры обработки данных для таких компаний, как Google, MS и Yahoo. Представьте, что ваш веб-поиск основан на квантовых процессорах в облаке. Сейчас мы далеки от этого, но все же. Это одна из причин, по которой крупные IT-компании вкладывают значительные средства в разработку своих квантовых платформ.

Другая задача и, возможно, главная причина, по которой квантовые вычисления набирают такие обороты, — это факторизация больших целых чисел. Когда Питер Шор придумал алгоритм квантовой факторизации, он нанес серьезный удар по криптографической безопасности, используемой сегодня повсеместно. Алгоритм Шора угрожает современным системам шифрования, быстро раскладывая на множители большие целые числа. Эти числа используются с целью создания криптографических ключей для кодирования всех данных в Интернете: банковских счетов, бизнес-транзакций, чатов, видео с котиками и т. д. Алгоритм Шора настолько быстр, что фактически может вычислить самые большие целые числа за считанные минуты. Сравните это с нынешним классическим чемпионом — методом решета числового поля, которому могут понадобиться миллиарды лет для разложения таких чисел.

Квантовые компьютеры могут быть бесценными инструментами не только в сферах поиска и шифрования, но и для симуляций, молекулярного моделирования, искусственного интеллекта, нейронных сетей и многого другого.

Давайте посмотрим, как это происходит.

## **Сложные симуляции**

Физики согласны с тем, что симуляции на атомном уровне — та область, в которой квантовые машины превзошли самих себя. В конце концов, это идеальное соответствие: машина, построенная на основе атомов, сможет симулировать квантово-механические системы с гораздо большей точностью, чем классический компьютер. Было подсчитано, что квантовый

компьютер с несколькими десятками квантовых битов может выполнять симуляцию, на которую классическому компьютеру потребуется невероятное количество времени. Например, модель Хаббарда, названную в честь британского физика Джона Хаббарда, которая описывает движение электронов в кристалле, можно симулировать квантовым компьютером<sup>1</sup>. По мнению Хаббарда, эта задача выходит за рамки возможностей классического компьютера.

## Молекулярное моделирование и новые материалы

В статье, опубликованной в журнале *Science Magazine*, сообщалось, что химики из Итальянского технологического института в Генуе смоделировали молекулу гидрида бериллия<sup>2</sup> — соединения, состоящего из двух атомов водорода и одного атома бериллия, — в квантовом компьютере. Ничего особенного по сегодняшним классическим стандартам, однако это трамплин в будущее, полное надежд на открытие новых лекарств.

Молекулярное моделирование — это непаханое поле для квантовых машин, поскольку физики и химики регулярно используют компьютеры для симуляции поведения атомов и молекул. Математики утверждают, что большинство симуляций требует огромных вычислительных мощностей. Это особенно верно для молекулярного моделирования, поскольку с увеличением числа частиц взаимодействие между ними становится экспоненциально более сложным. Кроме того, странные законы квантовой механики затрудняют расчет распределения этих электронов в молекуле. Некоторые из экспериментов в данной области описаны в табл. 2.8.

<sup>1</sup> *Hubbard J.* Electron Correlations in Narrow Energy Bands // *Proceedings of the Royal Society of London* 276 (1365): 238–257. 1963. Bibcode:1963RSPSA.276..238H. doi:10.1098/rspa.1963.0204. JSTOR 2414761.

<sup>2</sup> *Popkin G.* Quantum computer simulates largest molecule yet. Sep. 13, 2017 // <http://www.sciencemag.org/news/2017/09/quantum-computer-simulates-largest-molecule-yet-sparking-hope-future-drug-discoveries>.

**Таблица 2.8.** Квантовые эксперименты по молекулярному моделированию

Год	Компания	Эксперимент
2016	Google	Исследователи из лаборатории квантовых вычислений в Венеции, штат Калифорния, использовали три кубита для расчета расположения электронов с наименьшей энергией в молекуле водорода
2017	IBM	IBM разрабатывает интерактивный алгоритм для расчета основных состояний определенных молекул. Ученые применяли до шести сверхпроводниковых кубитов для анализа водорода, гидрида лития и гидрида бериллия, кодируя расположение электронов каждой молекулы в квантовом компьютере и переводя молекулу в ее основное состояние, которое они измеряли и кодировали на обычном компьютере

В целом молекулярное моделирование имеет скромное начало, но его будущее выглядит блестящим для химических и фармацевтических компаний. Молекулярное моделирование намерено стать прекрасным приложением для квантовых вычислений.

## Усовершенствованное глубокое обучение

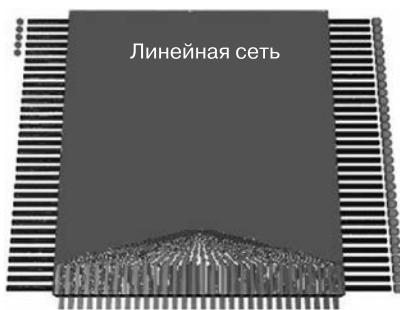
Когда речь идет о глубоком обучении, традиционные задачи делятся на три категории: симуляция, оптимизация и сэмплирование. В предыдущих разделах мы видели, насколько квантовый компьютер выделяется при симуляции, особенно на молекулярном и атомном уровнях, а что насчет оптимизации? Некоторые задачи оптимизации неразрешимы для традиционного оборудования из-за большого количества взаимодействующих переменных, необходимых для их решения. К таким задачам относятся свертка белков, симуляция полета космического корабля и др. Квантовые компьютеры могут эффективно заниматься оптимизацией, используя технику, называемую стохастическим градиентным спуском. Это метод поиска лучшего решения среди большого набора возможных, сравнимый с поиском самой низкой точки ландшафта, состоящего из холмов и долин.

### ПРИМЕЧАНИЕ

Канадская компания D-Wave уже продает коммерческие квантовые компьютеры, предназначенные для решения задач оптимизации с помощью стохастического градиентного спуска и других методов. Среди их клиентов — военно-промышленная компания Lockheed Martin и Google.

Задачи квантового сэмплирования попадают в набор вычислительных задач, которые производят выборки из вероятностных распределений. Двумя классами задач сэмплирования, которые демонстрируют мощь квантовых алгоритмов, являются бозонное и мгновенное сэмплирование за квантовое полиномиальное время. Несколько мелкомасштабных реализаций этих двух методов были выполнены с помощью квантовой оптики.

На рис. 2.16 схематически изображена задача бозонного сэмплирования для образца с 32 модами. Пять фотонов (*слева*) вводятся в линейную сеть с матрицей рассеяния (*внизу*), и все выходы наблюдаются в базисе Фока (*справа*). Согласно статье А. П. Лунда, М. Дж. Бремнера и Т. С. Ральфа, опубликованной в *Nature* в разделе «Квантовая информация», эта задача неразрешима для классических компьютеров, даже для систем среднего масштаба, таких как 50 бозонов на 2500 траекториях. Даже для небольших систем (20 бозонов и 400 траекторий) неизвестен выполнимый классический алгоритм, который может реализовать эту симуляцию<sup>1</sup>.



**Рис. 2.16.** Схема задачи бозонного сэмплирования

<sup>1</sup> Lund A. P., Bremner M. J., Ralph T. C. Quantum sampling problems, BosonSampling and quantum supremacy // [www.nature.com/articles/s41534-017-0018-2](http://www.nature.com/articles/s41534-017-0018-2).

Не так уж много для задач квантового сэмплирования, экспериментально демонстрирующих превосходство квантовых алгоритмов в данной области. Глубокое обучение и искусственный интеллект (ИИ) — вот две дисциплины, идущие рука об руку в сложных вычислениях с нейронными сетями, которые являются жемчужиной современных исследований.

## Квантовые нейронные сети и искусственный интеллект

Квантовые нейронные сети (QNN) — пока больше научная фантастика, чем научный факт. Тем не менее теоретическая основа заложена в 1990-х годах, и широко проводятся исследования по многим направлениям.

- *Использование квантовой обработки информации для улучшения существующих моделей нейронных сетей*<sup>1</sup>. Все дело в расширении существующих моделей с помощью более быстрых и эффективных алгоритмов. Это область, где квантовые вычисления непревзойденны. Немаловажным мотивом для этого исследования является сложность обучения классических нейронных сетей, особенно для приложений с большими данными. И есть надежда, что такие возможности квантовых вычислений, как параллелизм или эффекты интерференции и запутывания, могут сыграть на руку.
- *Потенциальные квантовые эффекты в мозге*<sup>2</sup>. Это направление объединяет квантовую физику и нейробиологию, здесь ведутся бурные дебаты, выходящие за рамки науки. Есть пионеры, усердно работающие в теоретической области квантовой биологии, которая набирает обороты благодаря следующим открытиям:
  - признакам эффективного переноса энергии при фотосинтезе за счет квантовых эффектов;
  - сообщениям об эффектах Mag-Lag у пациентов, прошедших МРТ, из-за чего предполагают, что тонкие взаимодействия в мозге могут быть квантовыми по своей природе.

<sup>1</sup> Schuld M., Sinayskiy I., Petruccione F. The quest for a Quantum Neural Network // Quantum Information Processing, 13, 11. 2014. — P. 2567–2586.

<sup>2</sup> Loewenstein W. Physics in mind. A quantum view of the brain. Basic Books, 2013.

- *Квантовая ассоциативная память.* Это новый алгоритм, введенный Дэном Вентурой и Тони Мартинесом в 1999 году<sup>1</sup>. Они предлагают квантовый компьютер на основе схем, имитирующий ассоциативную память. Алгоритм записывает состояния памяти в суперпозиции, а затем использует квантовый поиск, аналогичный алгоритму Гровера, для извлечения состояния памяти, наиболее близкого к заданному входу, с конечной целью — имитировать особенности человеческого мозга.
- *Черные дыры.* Хотите верьте, хотите нет, существуют идеи моделирования черных дыр как QNN. А также предположения, что черные дыры и мозг могут хранить воспоминания схожим образом<sup>2</sup>.

В целом, если в будущем человечество поработит квантовый компьютер, подобный SkyNet, возможно, он будет сделан из своего рода QNN. Прямо сейчас это может казаться шуткой, но великие ученые, такие как Стивен Хокинг, предупреждали об этом. Нам осталось только прислушаться. В следующем разделе рассмотрим подводные камни, из-за которых сложно построить квантовые компьютеры.

## Подводные камни квантовых компьютеров: декогеренция и интерференция

Декогеренция и интерференция — базовые принципы квантовой механики, вызывающие трудности при крупномасштабных квантовых вычислениях.

### Декогеренция

В квантовой механике частицы описываются волновой функцией. Фундаментальное свойство квантовой механики называется *когерентностью*

<sup>1</sup> Ventura D., Martinez T. A quantum associative memory based on Grover's algorithm // Proceedings of the International Conference on Artificial Neural Networks and Genetics Algorithms, 1999. — P. 22–27.

<sup>2</sup> Dvali G. and colleagues. Black Holes as Brains: Neural Networks with Area Law Entropy // <https://arxiv.org/pdf/1801.03918.pdf>.

или определенным фазовым соотношением между состояниями. Когерентность необходима для функционирования квантовых компьютеров. Но когда квантовая система контактирует со своей средой, со временем когерентность ослабевает, и этот процесс называется *квантовой декогеренцией*. Формально декогеренция — это время, которое требуется для исчезновения суперпозиции состояний, и на него влияет вероятностный характер волновой функции. Эту ситуацию можно рассматривать как утечку информации из системы в окружающую среду.

---

### ПРИМЕЧАНИЕ

Понятие декогеренции было введено немецким физиком Х. Дитером Цее в 1970 году, чтобы было проще понять коллапс волновой функции<sup>1</sup>.

---

Декогеренцию можно обнаружить экспериментальным путем: квантовая механика говорит, что частицы могут находиться в нескольких состояниях (невозбужденном или возбужденном или в двух разных местах) одновременно. Только акт наблюдения дает случайное значение для конкретного состояния. Если возбуждение измеряется энергетическими уровнями частицы (где низкий энергетический уровень означает отсутствие возбуждения, а высокий — его наличие), когда электромагнитная волна посылается к частице с соответствующей частотой, будут происходить колебания энергетических уровней частицы между высокими и низкими. Затем состояние частицы можно измерить и усреднить, создав так называемые осцилляции Раби. Поскольку частица никогда не бывает полностью изолирована из-за столкновений атомов, электромагнитных полей или тепловых резервуаров, например, то суперпозиция прекратится и колебания исчезнут.

Таким образом, декогеренция дает информацию о взаимодействии квантового объекта и его среды, и это имеет решающее значение для

---

<sup>1</sup> Schlosshauer M. Decoherence, the measurement problem, and interpretations of quantum mechanics // Reviews of Modern Physics, 76 (4), 2005. — P. 1267–1305. arXiv:quant-ph/0312059 Freely accessible. Bibcode:2004RvMP...76.1267S. doi:10.1103/RevModPhys.76.1267.

квантовых вычислений. То есть чем выше когерентность (время, в течение которого сохраняется суперпозиция), тем выше будет качество кубита. Некоторые виды кубитов, такие как сверхпроводящие контуры, сохраняют когерентность состояний совсем недолго и должны находиться при очень низких температурах ( $-271^{\circ}\text{C}$ ), чтобы противостоять этому эффекту. Другие, такие как ионные ловушки и NV-центры в алмазе, очень долго сохраняют когерентность состояний и могут находиться при комнатной температуре. Коммерческие предприятия, работающие в области создания квантовых компьютеров, сталкиваются с огромной проблемой, пытаясь бороться за сохранение когерентности состояний кубитов. Более подробное описание этих усилий смотрите в подразделе «Проектирование кубитов» на с. 72.

## Квантовая коррекция ошибок

Квантовая коррекция ошибок (QEC) направлена на достижение отказоустойчивых квантовых вычислений путем защиты информации от ошибок из-за декогеренции и других помех окружающей среды. Когда квантовый компьютер настраивает некоторые кубиты, он использует квантовые вентили, чтобы запутать их и манипулировать вероятностями, а затем, наконец измерив выходные коллапсирующие суперпозиции, получает конечную последовательность из 0 или 1. Это означает, что серия вычислений в целом выполняется в тот момент, когда все ваши схемы отработали. В конечном счете вы можете измерить только один выход из всего спектра возможных решений. У каждого возможного решения есть вероятность оказаться правильным, поэтому его, возможно, придется перепроверить и попытаться получить повторно. Этот процесс называется квантовой коррекцией ошибок.

В классическом мире коррекция ошибок выполняется с избыточностью, то есть путем создания копий данных, последующего присвоения вероятностей возможным условиям возникновения ошибки и, наконец, сравнения условия с наибольшей вероятностью с исходным сообщением, чтобы определить, произошла ли ошибка. Чтобы проиллюстрировать этот процесс, рассмотрим следующую таблицу, представляющую один бит информации.



Сообщение	Избыточные копии	Ошибка (1)	Ошибка (1, 2)
0	0	1	1
	0	0	1
	0	0	0
Вероятность		$(1/3) = 0,33$	$(1/3) \cdot (1/3) = 0,11$

Допустим, есть однобитовое сообщение (0) и мы создаем три избыточные копии для исправления ошибок. Предполагая, что ошибки, вызванные шумами, независимы и происходят с некоторой вероятностью, более вероятно, что ошибка возникает в одном бите, а переданное сообщение равно трем 0. Возможно также, что появляется двухбитовая ошибка и переданное сообщение равно трем 1, но этот результат менее вероятен. Таким образом, мы можем использовать этот метод для исправления сообщения в случае ошибок в классической системе. К сожалению, это невозможно в квантовых масштабах из-за *теоремы о запрете клонирования квантовых состояний* (no-cloning theorem).

#### ПРИМЕЧАНИЕ

Теорема о запрете клонирования квантовых состояний утверждает, что невозможно создать точную копию произвольного неизвестного квантового состояния. Она была сформулирована и доказана физиком Джеймсом Л. Парком в 1970 году<sup>1</sup>.

Теорема о запрете клонирования создает проблемы для квантовых вычислений, поскольку для исправления ошибок нельзя создавать избыточные копии кубитов. Тем не менее можно распространить информацию одного кубита на сильно запутанное состояние нескольких физических кубитов. Эта техника была открыта Питером Шором с помощью метода с использованием кода коррекции ошибок путем хранения информации одного кубита на девяти запутанных кубитах. Однако данная схема предотвращает появление лишь некоторых ошибок. Со временем было

<sup>1</sup> Wootters W., Zurek W. A Single Quantum Cannot be Cloned // Nature, 299, 1982. — P. 802–803. Bibcode:1982Natur.299..802W.doi:10.1038/299802a0.

разработано несколько схем квантовых кодов коррекции ошибок. Наиболее значимыми являются следующие.

### Трехкубитный код

Это основная отправная точка для квантовой коррекции ошибок. Метод кодирует один логический кубит в три физических так, что может исправить единичную ошибку инвертирования разрядов в матрице Паули  $X (\sigma_x)$ . Этот код способен исправлять ошибки без измерения состояния исходного кубита с помощью двух дополнительных кубитов для извлечения так называемой информации о *синдроме* (информации о возможных ошибках) из блока данных и не нарушая исходное состояние. Однако этот код не может исправлять битовые и фазовые (знаковые) инвертирования разрядов одновременно, возможно только однобитовое инвертирование. Питер Шор использовал данный метод для разработки девятикубитного кода, исправляющего ошибки.

### Код Шора

Этот код коррекции ошибок основан на трехкубитном коде и способен исправлять инверсию битов, инверсию знака или то и другое одновременно. Код Шора работает путем кодирования одного логического кубита в девять физических кубитов, задействуя это дополнительное реальное состояние для хранения информации о синдроме возможных ошибок. Обратите внимание, что этот код может исправлять ошибки только в одном кубите. Код, как правило, проще и позволяет создавать структуры схем, лучше совместимые с физическими ограничениями компьютерной архитектуры. Другие современные разработки в области квантовой коррекции ошибок включают:

- *бозонные коды* — пытаются хранить информацию об исправлении ошибок в бозонных модах, используя то преимущество, что осцилляторы имеют бесконечно большое количество уровней энергии в одной физической системе<sup>1</sup>;

<sup>1</sup> *Cochrane P. T., Milburn G. J., Munro W. J.* Macroscopically distinct quantum superposition states as a bosonic code for amplitude damping // *Physical Review A*, 59 (4), 1999. — P. 2631–2634. doi:10.1103/PhysRevA.59.2631.

○ *топологические коды* — были введены физиком Алексеем Китаевым при разработке *торического* кода для исправления топологических ошибок. Структура этого кода определяется на двумерной решетке с использованием цепочек ошибок, которые определяют нетривиальные топологические пути по поверхности кода<sup>1</sup>.

В общем, декогеренция и коррекция квантовых ошибок не облегчают работу ИТ-компаний, стремящихся реализовать мечту о крупномасштабных отказоустойчивых квантовых вычислениях. Тем не менее прогресс идет быстрыми темпами благодаря новым конструкциям кубитов с длительным сохранением когерентности состояний и улучшенным кодам квантовой коррекции ошибок. На самом деле темп такой быстрый, что эксперты в этой области придумали новый звучный термин для крупномасштабных квантовых вычислений — *квантовое превосходство*.

## Процессор на 50 кубитах и задача для квантового превосходства

Квантовое превосходство — звучный термин. Он был придуман физиком Джоном Прескиллом, чтобы описать переломный момент, при котором квантовый компьютер сможет решать задачи, неразрешимые для классических компьютеров. Это очень мощное утверждение, так как оно требует доказательства суперполиномиального ускорения по сравнению с лучшими классическими аналогами.

---

### ПРИМЕЧАНИЕ

Суперполиномиальное ускорение — это улучшение времени выполнения алгоритма до выхода за пределы полиномиального времени. Например, алгоритм, который выполняется за время  $k1n^{c1} + k2n^{c2} + \dots$ , где  $k$  и  $c$  — произвольные константы, а  $n$  — размер входных данных, называется полиномиальным временем. Алгоритм, время работы которого  $2^n$ , где  $n$  — размер входного сигнала, называется алгоритмом суперполиномиального времени.

---

<sup>1</sup> Kitaev A. Y. Quantum Computations: algorithms and error correction. 52:1191, 1997.

Исследователи усердно трудятся над доказательством квантового превосходства с помощью нескольких уже существующих алгоритмов, которые обеспечивают суперполиномиальное ускорение, превосходя классических чемпионов. Далее подробно описывается хроника этой деятельности.

- 1982 год. Ричард Фейнман, гигант квантовой механики, предлагает квантовый компьютер, который может использовать атомные принципы суперпозиции, интерференции и запутывания. Такая машина изменит правила игры.
- 1994 год. Математик Питер Шор предлагает пресловутый алгоритм факторизации для квантового компьютера. Алгоритм признается сенсацией, когда становится понятно, что по временной сложности он обошел классического суперчемпиона (метод решета числового поля, NFS) благодаря суперполиномиальным ускорениям. Алгоритм не реализован и не доказан экспериментально, тем не менее джинн вылетел из бутылки, так как волнение нарастает почти так же быстро, как ускорение алгоритма Шора по сравнению с NFS.
- 2012 год. Физик Джон Прекилл ввел термин «*квантовое превосходство*» в статье «Квантовые вычисления и граница запутанности», чтобы формально описать момент, когда квантовые компьютеры возьмут верх. Гонка идет среди гигантов информационных технологий.
- 2016 год. Google, крупнейший поисковик, решает взять на себя задачу доказать квантовое превосходство к концу 2017 года, создав 49-кубитный чип, который сможет производить выборки из распределений, недоступные для любых современных классических компьютеров, за разумное время. Усилия не увенчались успехом.
- 2017 год. Исследователи из лаборатории Т. Дж. Уотсона в IBM симулируют 49- и 56-кубитные схемы на обычном суперкомпьютере Blue Gene/Q в Ливерморской национальной лаборатории им. Лоуренса, увеличивая количество кубитов, необходимых для квантового превосходства<sup>1</sup>.

---

<sup>1</sup> Pednault E. and colleagues. Breaking the 49-Qubit Barrier in the Simulation of Quantum Circuits // <https://arxiv.org/pdf/1710.05867.pdf>.

- *2018 год.* Растет скептицизм по поводу доказательства квантового превосходства, поскольку подводные камни квантовых вычислений становятся все более очевидными: оценки квантовой коррекции ошибок достигают 3 % от входных данных в каждом цикле. Квантовые компьютеры намного более зашумлены и сильнее подвержены ошибкам по сравнению с классическими аналогами. Отказоустойчивый квантовый компьютер становится Святым Граалем.

Несмотря на то что до окончательного доказательства квантового превосходства еще далеко, ИТ-специалисты считают, что компании начнут получать отдачу от инвестиций в квантовые технологии в ближайшие несколько лет. Когда или при каком количестве кубитов наступит это так называемое квантовое превосходство, когда даже суперкомпьютеры не смогут держать марку? Хотите верить, хотите нет, но в Канаде есть компания D-Wave Systems, которая продает 2000-кубитные компьютеры для коммерческого использования. Правда, их работа остается противоречивой из-за процесса, называемого квантовым отжигом. В следующем разделе показано почему.

## Полемика о квантовом отжиге и минимизации энергии

Квантовый отжиг (QA), иногда называемый адиабатическими квантовыми вычислениями (AQC), является формой квантовых вычислений с использованием адиабатической теоремы. Не вдаваясь в технические подробности, приведу список концепций, облегчающих понимание этого процесса.

- *Адиабатическая теорема.* Предложена Максом Борном и Владимиром Фоком в 1928 году. Она гласит: «Квантово-механическая система, подверженная постепенному изменению внешних условий, адаптирует свою функциональную форму, но, когда подвергается быстро меняющимся условиям, для адаптации функциональной формы недостаточно времени, поэтому пространственная плотность вероятности остается неизменной».

- *Гамильтониан ( $H$ )*. Важное понятие в квантовой механике, особенно для квантового отжига. В квантовой механике гамильтониан является оператором, соответствующим полной энергии системы в большинстве случаев. Другими словами, это сумма кинетических энергий всех частиц плюс потенциальная энергия частиц, связанных с системой.

---

### СОВЕТ

Адиабатическую теорему легче понять, рассмотрев простой пример колебания маятника в вертикальной плоскости. Если опора маятника движется резко, режим колебаний изменится. Но если опора перемещается очень медленно, движение маятника относительно нее останется неизменным. В этом суть адиабатического процесса: постепенное изменение внешних условий позволяет системе адаптироваться так, что она сохраняет первоначальный характер.

---

Схематично квантовый отжиг можно описать следующими шагами.

1. Найдите потенциально сложный гамильтониан, основное состояние которого описывает решение интересующей задачи.
2. Подготовьте систему с простым гамильтонианом и инициализируйте до основного состояния.
3. Используйте адиабатический процесс, чтобы развить простой гамильтониан в желаемый сложный гамильтониан. Согласно адиабатической теореме система остается в основном состоянии, таким образом, конечное состояние системы описывает решение задачи.

Пионером в этом виде квантовых вычислений является компания D-Wave Systems, которая продала несколько квантовых компьютеров с довольно большим количеством кубитов.

## Две тысячи кубитов: все не так, как кажется

Рассмотрим хронику для серий квантовых систем, проданных D-Wave.

- *2007 год*. D-Wave продемонстрировала свое первое 16-кубитное оборудование.

- 2011 год. D-Wave One, 128-кубитный компьютер, проданный Lockheed Martin за 10 миллионов долларов.
- 2013 год. D-Wave Two, 512-кубитный компьютер, проданный Google для лаборатории квантового искусственного интеллекта, пытающейся доказать квантовое превосходство.
- 2015 год. D-Wave 2X, преодолел 1000-кубитный предел, продан неизвестному покупателю.
- 2017 год. D-Wave 2000Q, их последний, 2000-кубитный компьютер, проданный фирме, занимающейся обеспечением кибербезопасности, под названием Temporal Defense Systems за 15 миллионов долларов.

Трудно поверить в то, что 2000-кубитный квантовый компьютер уже продан, когда такие гиганты, как IBM и Google, только начинают создавать 16-кубитные системы. Это притом, что IBM — компания, которая специализируется на крупномасштабном оборудовании и располагает самыми большими капиталами среди всех. Однако здесь дело в том, что, несмотря на большое количество кубитов, D-Wave 2000Q не способен решить большинство задач, с которыми может столкнуться система IBM Q.

На самом деле компьютер D-Wave может решать только задачи квантового отжига, то есть задачи, решаемые в соответствии с адиабатической теоремой.

## **Квантовый отжиг: подмножество квантовых вычислений**

Специалисты в данной области назвали квантовый отжиг ограничительным, что вызвало некоторые споры ввиду следующих фактов.

- Платформы наподобие IBM Q используют логические вентили для управления кубитами, тогда как компьютеры, реализующие квантовый отжиг, их не содержат и поэтому не могут полностью управлять состояниями кубитов.
- Системы D-Wave отличаются тем, что их кубиты стремятся к минимальному состоянию энергии. Ими нельзя управлять с помощью квантовых вентилях, но их поведение можно предсказать по адиабатической

теореме. Это делает их хорошим инструментом для решения задач минимизации энергии.

- Квантовый отжиг применяется главным образом для задач комбинаторной оптимизации, где пространство поиска дискретно с локальным минимумом (например, для нахождения основного состояния разупорядоченного магнетика или спинового стекла<sup>1</sup>). QA пользуется тем, что все физические системы стремятся к состоянию с минимальной энергией. Проиллюстрируем это на примере чашки горячего кофе: если на некоторое время оставить ее на прилавке, кофе начнет остывать, пока не достигнет температуры, равной температуре окружающей среды. Таким образом, кофе стремится к состоянию с минимальной энергией.

---

#### ПРИМЕЧАНИЕ

Математическая оптимизация — метод из семейства локального поиска. Это итеративный метод, который начинается с произвольного решения задачи, а затем пытается найти лучшее решение, постепенно изменяя один из его элементов. Если это приводит к лучшему решению, в него вносятся дополнительные изменения, повторяющиеся до тех пор, пока не будут найдены более оптимальные решения.

---

Вопрос о том, сможет ли машина D-Wave QA превзойти классические компьютеры, остается без ответа. Есть несколько исследований, каждое из которых идет своим путем: в январе 2016 года ученые из Google использовали систему D-Wave для проведения серии тестов по *туннелированию с конечным радиусом действия* с QA-решателем в сравнении с *имитацией отжига (SA)* и *симулированным квантовым методом Монте-Карло (QMC)* на одноядерном классическом процессоре<sup>2</sup>. Результат: QA-решатель превзошел SA и QMC в  $10^8$  раз.

---

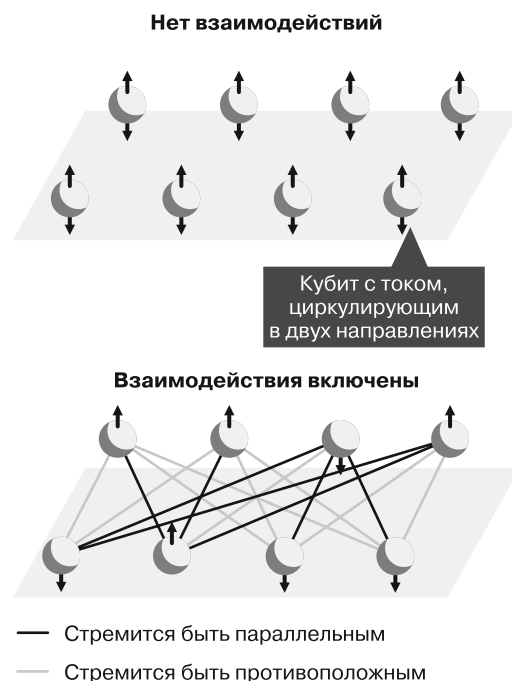
<sup>1</sup> Ray P., Chakrabarti B. K., Chakrabarti A. Sherrington-Kirkpatrick model in a transverse field: Absence of replica symmetry breaking due to quantum fluctuations // Phys. Rev. B 39, 11828. 1989.

<sup>2</sup> Denchev V. S. and colleagues. What is the Computational Value of Finite Range Tunneling? // Google Labs, Jan 2016 // <https://arxiv.org/pdf/1512.02206.pdf>.



Это впечатляет. Однако другие говорят, что не стоит торопиться: исследователи из Швейцарского федерального технологического института не сделали заявления о существенном ускорении для чипа D-Wave, но не исключили того, что его можно достичь в будущем.

На рис. 2.17 показаны основные внутренние процессы процессора QA, такого как в D-Wave. Он состоит из двумерного массива кубитов из сверхпроводящих контуров, которые переносят электрический ток. Кубиты действуют как магниты, которые могут указывать вверх, вниз или, согласно свойствам квантовой механики, вверх и вниз одновременно. Каждый кубит в массиве может взаимодействовать с другими через связующие элементы, которые можно запрограммировать так, чтобы они могли снизить свою энергию, указывая в том же или противоположном направлении. Идея состоит в том, чтобы закодировать задачу, указав все возможные взаимодействия в микросхеме, и решить ее, найдя минимальную энергию или *основное состояние* кубита.



**Рис. 2.17.** Схема процессора на основе квантового отжига

Чтобы найти основное состояние, машина запускает массив в запутанном состоянии и медленно начинает взаимодействия. Затем система ищет состояние с самой низкой энергией аналогично тому, как шарик катится по неровной поверхности в поисках наиболее глубокой точки. В классической физике колебания тепловой энергии ведут шар к нижней точке, это называется *термическим отжигом*. Однако в квантовой механике шар может пройти через низкие точки, чтобы еще быстрее найти самые низкие. Это причина, по которой квантовый отжиг считается более быстрым для таких задач, как распознавание образов или машинное обучение.

Таким образом, архитектура D-Wave отличается от архитектуры традиционных квантовых компьютеров тем, что может решить только задачу минимизации энергии. Это привело к полемике с некоторыми сотрудниками из IBM, назвавшими ее тупиковой. Даже ученые из Google, которые проводили эксперимент с QA на D-Wave 2X для примеров бинарной оптимизации  $K$ -го порядка, в своих сводках утверждали, что имитация отжига — для «невежественных или отчаянных». Усилил полемику тот факт, что D-Wave не может выполнить алгоритм Шора, потому что это не процесс минимизации энергии. Шору требуется так называемый универсальный квантовый компьютер, который может выполнять любой квантовый алгоритм.

## Универсальные квантовые вычисления и будущее

Универсальный квантовый компьютер, известный также как квантовая машина Тьюринга (QTM), является конечной квантовой машиной. Он был определен как абстрактная машина, способная охватить всю мощь квантовых вычислений. То есть он способен выполнять любой квантовый алгоритм. Несмотря на то что на реализацию этой мечты потребуются десятилетия, началась новая глобальная гонка, в рамках которой крупные игроки ИТ-отрасли и правительства вкладывают значительные средства в исследования и разработку этих машин.

## Google и квантовый искусственный интеллект

Google был первым клиентом D-Wave и использовал их машины в серии экспериментов по оптимизации, результаты которых показали, что квантовый отжиг может протекать значительно быстрее, чем имитация отжига на одноядерном процессоре. Кроме того, Google объявил, что разрабатывает собственную технологию квантовых вычислений, что имеет смысл, учитывая количество ресурсов, находящихся в их распоряжении. Хотя в настоящее время ничего подходящего для демонстрации нет, похоже, что идет работа над гибридом, основанным на вентильном подходе IBM и квантовом отжиге D-Wave.

Фактически в июне 2017 года Google объявил, что они тестируют квантовый компьютер с 20 кубитами, надеясь к 2018 году создать компьютер с 49 кубитами. Кажется, они хотят бросить вызов IBM в достижении квантового превосходства. Google ясно дал понять, чего они хотят в области квантовых вычислений: создать искусственный интеллект (ИИ). В статье *Commercialize Early Quantum Technologies* для *Springer Nature* они представляют лабораторию квантового ИИ, организованную для создания отказоустойчивой квантовой машины, способной решить любые задачи. Усилия Google сосредоточены в трех ключевых областях машинного обучения и искусственного интеллекта.

- *Симуляция.* Одним из наиболее естественных способов применения является моделирование химических реакций и материалов: более прочных полимеров для самолетов, улучшенных каталитических нейтрализаторов для автомобилей, более эффективных материалов для солнечных батарей, новых фармацевтических препаратов и воздухопроницаемых тканей. Квантовые вычисления обещают сэкономить огромные суммы, перенеся компьютерную мощность, необходимую для создания этих материалов, на новый уровень. Разработка вычислительных материалов представляет собой крупную отрасль со множеством бизнес-моделей, созданных для квантового моделирования: оплата подписки на доступ, консультации, обмен акций на квантовые инновации и др.
- *Оптимизация.* Задачи оптимизации трудно решить на обычных компьютерах. Лучшие классические методы используют статистические

методы, такие как минимизация энергии (термический отжиг). Квантовые принципы могут обеспечить значительное ускорение путем туннелирования через тепловые барьеры, чтобы найти минимально возможную точку или лучшее решение. В целом квантовая оптимизация может быть полезна для большинства задач машинного обучения. Логистические компании, диагностика пациентов в учреждениях здравоохранения и компании, занимающиеся поиском в Сети, могут получить значительные инновации.

- *Сэмплирование.* В основном связана с задачами машинного обучения, такими как вывод и распознавание образов. Квантовое сэмплирование может обеспечить превосходную производительность в запросах с распределением вероятностей. И не только это. Благодаря значительному параллелизму, достигнутому квантовыми компьютерами, можно использовать сэмплирование, чтобы окончательно доказать квантовое превосходство.

Google делает большие ставки на квантовую оптимизацию и управление рисками в будущем, но сейчас у IBM есть преимущество: 20-кубитная платформа для коммерческих клиентов и 16-битная бесплатная *Q Experience* для всех облачных платформ. Одно можно сказать наверняка: в ближайшее время у каждого крупного поставщика появятся облачные квантовые платформы.

## **Квантовые машины в центрах обработки**

Проектирование и конструирование кубитов основано на экстремальных разработках. Поскольку квантовая механика имеет странную природу, кубиты очень чувствительны к шуму окружающей среды, склонны к ошибкам из-за принципа декогеренции, вдобавок, как правило, трудно управлять ими и создавать их в больших масштабах. Таким образом, не ожидайте, что вскоре увидите квантовый компьютер в местном магазине. Не рассчитывайте, что в ближайшие десятилетия ваш внук сможет купить квантовый компьютер и поставить его в гостиной. Если не будет существенного прогресса в технологиях, это вряд ли произойдет вообще. Частично это связано с тем, что кубиты

должны храниться при ультранизкой температуре (0,015 К, или около  $-273\text{ }^{\circ}\text{C}$ ), чтобы избежать воздействия шума от окружающей среды. Для того чтобы иметь некоторое представление об этой температуре, рассмотрите таблицу, в которой приведены средние температуры для различных областей Вселенной.

Объект	Температура, К	Температура, $^{\circ}\text{C}$
Кубит	0,015	$-273$
Космический вакуум (температура, создаваемая равномерным фоновым излучением или послесвечениями после Большого взрыва)	2,7	$-270$
Средняя температура Земли	331	58
Температура Луны в дневное/ночное время	373/100	100/ $-173$

### ПРИМЕЧАНИЕ

Кельвин — основная единица измерения температуры в физике. В классической термодинамике ноль по шкале Кельвина определяется как абсолютный ноль или температура, при которой все тепловые движения прекращаются.

В краткосрочной перспективе очень вероятно, что квантовые компьютеры заполонят центры обработки данных. Это означает, что они не придут на смену персональным, а будут выполнять большинство сложных задач, таких как поиск, симуляция, моделирование и др. Кроме того, инсайдеры ожидают, что квантовые компьютеры дополняют традиционные, в результате чего можно будет развивать такие сферы, такие как шифрование, научная разведка и искусственный интеллект.

Итак, через несколько лет можно ожидать, что цифровой помощник в вашем телефоне или дома будет работать на базе квантового компьютера. Вот еще пища для размышлений: примерно через десять лет мы будем проводить большую часть нашего времени, общаясь с квантовыми компьютерами.

## Гонка становится глобальной

Происходит переход на новый уровень, когда в игру вступают правительства, обеспечивающие крупные инвестиции в этой области. Согласно пресс-релизу Digital Single Market, Европейская комиссия планирует начиная с 2018 года вложить 1 миллиард евро в создание Единого цифрового рынка (со значительным финансированием в течение следующих 20 лет<sup>1</sup>). Это дополнительные инвестиции к 550 млн евро, потраченным на отдельные инициативы, чтобы вывести Европу в авангард того, что они считают второй квантовой революцией.

Кроме того, согласно пресс-релизу Alibaba Cloud, выпущенному в июле 2015 года, Китайская академия наук (CAS) объединяется с Alibaba, крупнейшим игроком в области электронной коммерции в Китае, для создания *лаборатории квантовых вычислений CAS*. Квантовые вычисления превратились в глобальную гонку, и последствия будут значительными.

## Будущие приложения

Нет предела тому, чего можно достичь благодаря огромному потенциалу квантовых вычислений. Приведу примеры возможных будущих приложений и описание того, как они повлияют на наше общество, в следующих отраслях.

- *Авиационная промышленность.* Авиастроительные компании корпят над разработкой и использованием квантовых алгоритмов для моделирования воздушного потока, позволяющих экономить годы по сравнению со своими классическими аналогами. Это приведет к созданию за короткий промежуток времени более надежного и эффективного самолета с низким уровнем шума и выбросов.
- *Освоение космоса.* НАСА играло с системой D-Wave, решая различные задачи, начиная от оптимальных конструкций до оптимального размещения коммерческих грузов в космическом корабле. Другие приложе-

---

<sup>1</sup> European Commission will launch €1 billion quantum technologies flagship // Digital Single Market // <https://ec.europa.eu/digital-single-market/en/news/european-commission-will-launch-eu1-billion-quantum-technologies-flagship>.

ния включают в себя алгоритмы квантового искусственного интеллекта и квантово-классические гибридные алгоритмы.

- *Медицина.* Квантовые вычисления могут обеспечить превосходное молекулярное моделирование, что приведет к молниеносному моделированию белка, более быстрому появлению и тестированию новых лекарств. Это сократит жизненный цикл их доставки пациенту. Лекарства следующего поколения и средства от рака — в наших руках.

Это лишь некоторые из возможных способов применения квантовых вычислений. Обратите внимание, что я не упоминаю существующие достижения, такие как шифрование данных и безопасность: квантовая факторизация и возможность победить асимметричное шифрование, вероятно, являются основными причинами, по которым квантовые вычисления в последнее время набирают обороты. В следующей главе вы познакомитесь с IBM Q Experience. Это первая платформа квантовых вычислений в облаке, которая предоставляет реальные квантовые устройства для использования по своему усмотрению.

# 3

## IBM Q Experience: уникальная платформа для квантовых вычислений в облаке

В этой главе мы рассмотрим квантовые вычисления в облаке с помощью IBM Q Experience — первой платформы такого типа. Глава начинается с обзора Composer — веб-консоли, используемой для визуального создания схем, запуска экспериментов, исследования аппаратных устройств и многого другого. Далее вы узнаете, как создать свой первый эксперимент и запустить его на симуляторе или реальном квантовом устройстве. IBM Q Experience обладает мощным REST API для управления жизненным циклом эксперимента, и в главе будет показано, как это делать, с подробным описанием конечных точек и параметров запроса. Глава заканчивается примером практической реализации официальной библиотеки Python (названной `IBMQuantumExperience`) для Node JS. Эта пользовательская библиотека Node JS позволит проверить ваши навыки работы с асинхронным JavaScript и REST API. Приступим!

IBM, безусловно, занимает первое место в гонке квантовых вычислений в облаке. Они придумали действительно классную платформу для удаленного запуска экспериментов под названием Q Experience. Мне кажется или в названиях этих инструментов действительно много аналогий с теорией музыки? Проверим: визуальный редактор, используемый для создания квантовых схем, называется *Composer* («Композитор»). Не странно ли? Квантовые схемы, построенные с помощью редактора, называются *партитурами* (как в нотах), не говоря уже о том, что



визуально окно редактора очень похоже на страницу нотной тетради. Я давно играю на классической гитаре и, только взглянув на Composer, сразу вспомнил гитарную партитуру (с вентилями, напоминающими ноты). Все еще думаете, что я сумасшедший? Платформа называется Q Experience, а вы когда-нибудь слышали о Experience Джимми Хендрикса? Возможно, Composer — это сборник партитур, в котором вы создадите великолепный шедевр, чтобы все мы смогли им наслаждаться. Квантовые вычисления действительно способны изменить текущее положение дел.

## Первое знакомство с IBM Q Experience

Q Experience — это платформа IBM для квантовых вычислений в облаке, и она действительно крутая. Давайте посмотрим (все материалы публикуются с разрешения © International Business Machines Corporation).

1. Создайте аккаунт на <https://quantumexperience.ng.bluemix.net/qx/experience>. Вам понадобится указать адрес электронной почты. Затем дождитесь письма-подтверждения и продолжите регистрацию.
2. Войдите в веб-консоль и перейдите на вкладку Composer сверху (рис. 3.1).

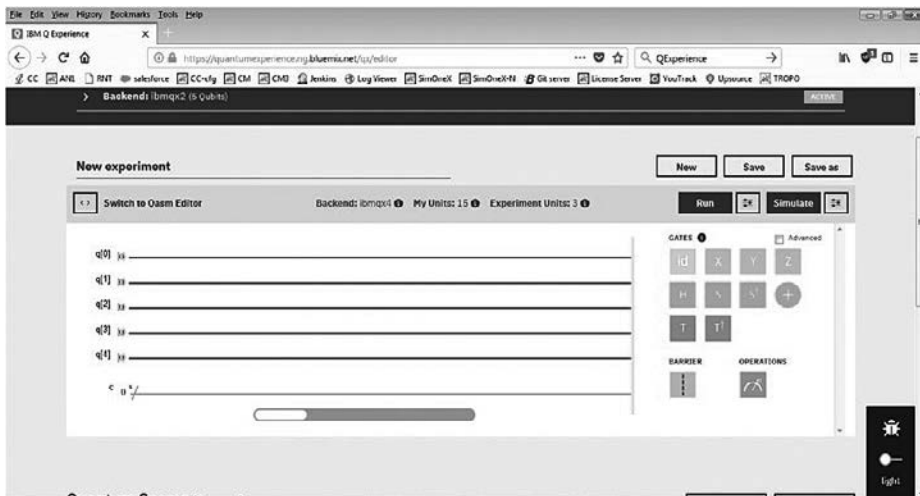


Рис. 3.1. Главное окно IBM Q Experience

## Квантовый Composer

Composer — это визуальный инструмент для создания квантовых схем, или партитур. Вверху показана гистограмма эксперимента с доступными для использования кубитами (рис. 3.2).

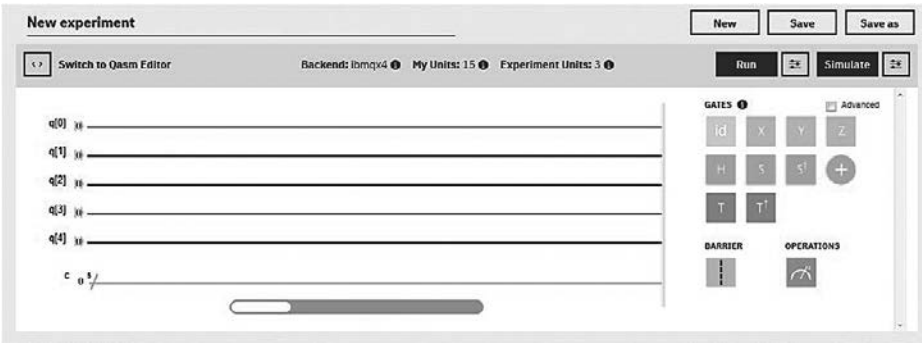



Рис. 3.2. Окно эксперимента в Composer








- В левой части гистограммы видны пять кубитов, доступных из процессора `ibmqx4`. Все они инициализированы в основное состояние  $|0\rangle$ . Линия внизу — это линия измерения, где будут собраны результаты схемы. Помните, что измерение должно выполняться последним в схеме, так как все операции вентилей выполняются параллельно и с наложением состояний.
- В правой части — квантовые вентили. Перетащите их к местоположению определенного кубита на гистограмме, чтобы начать строить схему.

## Квантовые вентили

Квантовые вентили, поддерживаемые IBM Q Experience, описаны в табл. 3.1.







Таблица 3.1. Квантовые вентили для IBM Q Experience

Вентиль	Описание
Паули X 	Вращает кубит на $180^\circ$ вокруг оси X. Преобразует $ 0\rangle$ в $ 1\rangle$ и $ 1\rangle$ — в $ 0\rangle$ . Известен также как инвертирование разрядов или НЕ-вентиль. Представлен матрицей $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Вентиль	Описание
Паули Y 	Производит поворот вокруг оси Y сферы Блоха на $\pi$ радиан. Представлен матрицей Паули $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ , где $i = \sqrt{-1}$ — мнимая единица
Паули Z 	Производит поворот вокруг оси Z сферы Блоха на $\pi$ радиан. Представлен матрицей Паули $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Адамара 	Представляет собой поворот на $\pi$ радиан вокруг оси $(X + Z)/\sqrt{2}$ . Иначе говоря, преобразует состояния: <ul style="list-style-type: none"> <li>• <math> 0\rangle</math> в <math>( 0\rangle +  1\rangle)/\sqrt{2}</math>;</li> <li>• <math> 1\rangle</math> в <math>( 0\rangle -  1\rangle)/\sqrt{2}</math>.</li> </ul> Этот вентиль требуется для создания суперпозиций
Фазового сдвига $\sqrt{Z}$ 	Обладает свойством преобразования $X \rightarrow Y$ и $Z \rightarrow Z$ . Данный вентиль расширяет вентиль Адамара для создания сложных суперпозиций
Эрмитово-сопряженная матрица для вентилей фазового сдвига $\sqrt{Z}$ 	Осуществляет преобразования $X \rightarrow -Y$ и $Z \rightarrow Z$
Управляемое НЕ (CNOT) 	Двухкубитный вентиль, который инвертирует целевой кубит (применяет оператор Паули X), если управляющий находится в состоянии 1. Этот вентиль требуется для создания запутывания
Фазового сдвига $\sqrt{S}$ 	Вентиль $\sqrt{S}$ выполняет половину обмена состояниями между двумя кубитами. Является универсальным: любой квантовый мультикубитный вентиль можно построить только из вентилей типа $\sqrt{\text{swap}}$ и однокубитных вентилей. Он представлен матрицей: $\sqrt{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2(1+i) & 1/2(1-i) & 0 \\ 0 & 1/2(1-i) & 1/2(1+i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Продолжение ➞

Таблица 3.1 (продолжение)

Вентиль	Описание
<p>Эрмитово-сопряженная матрица для вентилья фазового сдвига <math>\sqrt{S}</math> или оператор ИЛИ-НЕ, примененный к вентилью фазового сдвига <math>\sqrt{S}</math></p> 	<p>Представлен матрицей <math>\sqrt{S} = \begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 \\ 0 &amp; 1/2(1-i) &amp; 1/2(1+i) &amp; 0 \\ 0 &amp; 1/2(1+i) &amp; 1/2(1-i) &amp; 0 \\ 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}</math></p>
<p>Барьера</p> 	<p>Предотвращает преобразования вдоль его исходной линии</p>
<p>Измерительный</p> 	<p>Принимает на вход кубит в суперпозиции состояний и выдает либо 0, либо 1. Кроме того, вывод не является случайным. Есть вероятность того, что вывод примет значение 0 или 1, которое зависит от первоначального состояния кубита</p>
<p>Условный</p> 	<p>Применяет квантовую операцию при выполнении условия</p>
<p>Физическое частичное вращение (вентили U)</p> 	<p>U1 — однопараметрический однокубитный вентиль.  U2 — однокубитный двухпараметрический одноимпульсный вентиль.  U3 — однокубитный трехпараметрический двухимпульсный вентиль</p>
<p>Единичный (identity)</p> 	<p>Выполняет операцию простоя (idle) на кубите в течение одной единицы времени</p>

Вы можете перетаскивать вентили из правой части Composer, чтобы создать схему, а если предпочитаете писать код на ассемблере, переключитесь в режим редактора QASM (рис. 3.3).

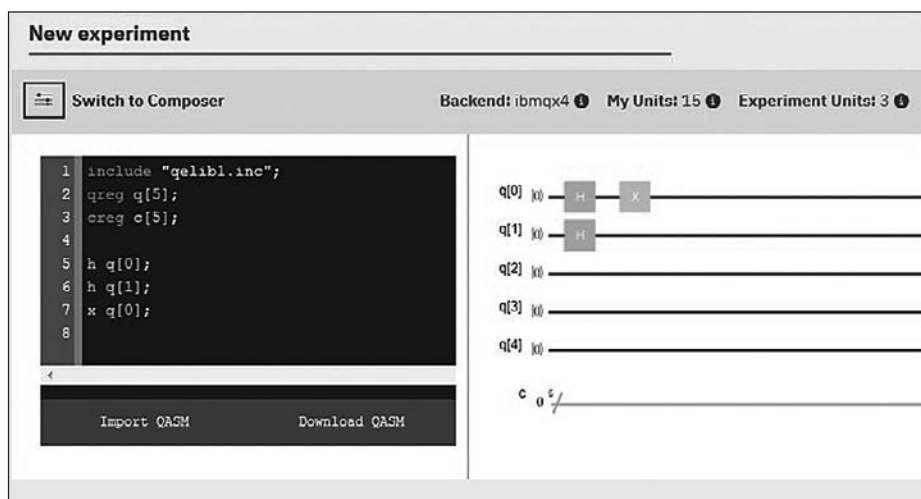


Рис. 3.3. Редактор экспериментов в режиме QASM

### ПРИМЕЧАНИЕ

QASM — это квантовый язык ассемблера, созданный на базе платформы OPENQASM. Используется для экспериментов с неглубокими квантовыми схемами. Несмотря на то что умение писать на ассемблере стало чем-то вроде утраченного искусства, некоторым QASM может показаться более удобным, чем Python SDK или даже визуальный редактор.

Теперь рассмотрим различные квантовые процессоры.

## Доступное квантовое серверное ПО

Есть несколько квантовых процессоров, которые можно выбрать для проведения экспериментов. В табл. 3.2 приведен официальный список, ранжированный в соответствии с количеством кубитов (по информации с сайта о серверном ПО для IBM Q Experience).

**Таблица 3.2.** Официальный список квантового серверного ПО, доступного для пользователей IBM Q Experience

Наименование	Подробности
Ibmqx2	Условное название: Sparrow. Количество кубитов: 5. Доступен онлайн с 24 января 2017 года
Ibmqx4	Условное название: Raven. Количество кубитов: 5. Доступен онлайн с 25 сентября 2017 года
Ibmqx3	Условное название: Albatross. Количество кубитов: 16. Доступен онлайн с июня 2017 года
Ibmqx5	Условное название: Albatross. Количество кубитов: 16. Доступен онлайн с 28 сентября 2017 года

В табл. 3.2 приведен официальный список процессоров, доступных на момент написания книги, но существует куда более интересный способ получить обновленный список доступных машин в реальном времени с помощью превосходного REST API. Более подробное описание этого API дано в разделе «Удаленный доступ через REST API» текущей главы, а пока продемонстрирую, как получить всегда актуальный список серверного ПО, используя конечную точку *REST Available Backend List*: [https://quantumexperience.ng.bluemix.net/api/Backends?access\\_token=ACCESS-TOKEN](https://quantumexperience.ng.bluemix.net/api/Backends?access_token=ACCESS-TOKEN).

---

### СОВЕТ

Чтобы получить токен доступа, смотрите подраздел «Аутентификация» раздела «Удаленный доступ через REST API» на с. 130. Обратите внимание, что токен API не совпадает с токеном доступа. Токены API используются для выполнения квантовых программ через Python SDK. Токены доступа применяются для вызова REST API.

---

Перейдя по URL, приведенному в предыдущем абзаце, вы увидите список квантовых процессоров в формате JSON. Вот как это выглядит на момент написания книги (листинг 3.1). Учтите, что полученный вами результат может отличаться.

**Листинг 3.1.** HTTP-ответ на вызов Backend Information REST API

```
[{
  "name": "ibmqx2",
  "version": "1",
  "status": "on",
  "serialNumber": "Real5Qv2",
  "description": "5 transmon bowtie",
  "basisGates": "u1,u2,u3,cx,id",
  "onlineDate": "2017-01-10T12:00:00.000Z",
  "chipName": "Sparrow",
  "id": "28147a578bdc88ec8087af46ede526e1",
  "topologyId": "250e969c6b9e68aa2a045ffbceb3ac33",
  "url": "https://ibm.biz/qiskit-ibmqx2",
  "simulator": false,
  "nQubits": 5,
  "couplingMap": [
    [0, 1],
    [0, 2],
    [1, 2],
    [3, 2],
    [3, 4],
    [4, 2]
  ]
}, {
  "name": "ibmqx5",
  "version": "1",
  "status": "on",
  "serialNumber": "ibmqx5",
  "description": "16 transmon 2x8 ladder",
  "basisGates": "u1,u2,u3,cx,id",
  "onlineDate": "2017-09-21T11:00:00.000Z",
  "chipName": "Albatross",
  "id": "f451527ae7b9c9998e7addf1067c0df4",
  "topologyId": "ad8b182a0653f51dfbd5d66c33fd08c7",
  "url": "https://ibm.biz/qiskit-ibmqx5",
  "simulator": false,
  "nQubits": 16,
  "couplingMap": [
    [1, 0],
    ...
  ]
}]
```

```

        [15, 14]
    ]
}, {
    "name": "Device Real5Qv1",
    "status": "off",
    "serialNumber": "Real5Qv1",
    "description": "Device Real5Qv1",
    "id": "cc7f910ff2e6860e0d4918e9ee0ebae0",
    "topologyId": "250e969c6b9e68aa2a045fffbceb3ac33",
    "simulator": false,
    "nQubits": 5,
    "couplingMap": [
        [0, 1],
        [0, 2],
        [1, 2],
        [3, 2],
        [3, 4],
        [4, 2]
    ]
}, {
    "name": "ibmqx_hpc_qasm_simulator",
    "status": "on",
    "serialNumber": "hpc-simulator",
    "basisGates": "u1,u2,u3,cx,id",
    "onlineDate": "2017-12-09T12:00:00.000Z",
    "id": "084e8de73c4d16330550c34cf97de3f2",
    "topologyId": "7ca1eda6c4bfff274c38d1fe66c449dfff",
    "simulator": true,
    "nQubits": 32,
    "couplingMap": "all-to-all"
}, {
    "name": "ibmqx4",
    "version": "1",
    "status": "on",
    "serialNumber": "ibmqx4",
    "description": "5 qubits transmon bowtie chip 3",
    "basisGates": "u1,u2,u3,cx,id",
    "onlineDate": "2017-09-18T11:00:00.000Z",
    "chipName": "Raven",
    "id": "c16c5ddebbf8922a7e2a0f5a89cac478",
    "topologyId": "3b8e671a5a3b56899e6e601e6a3816a1",
    "url": "https://ibm.biz/qiskit-ibmqx4",
    "simulator": false,
    "nQubits": 5,
    "couplingMap": [
        [1, 0],
        [2, 0],
        [2, 1],
    ]
}

```



```

        [2, 4],
        [3, 2],
        [3, 4]
    ]
}, {
    "name": "ibmqx3",
    "version": "1",
    "status": "off",
    "serialNumber": "ibmqx3",
    "description": "16 transmon 2x8 ladder",
    "basisGates": "u1,u2,u3,cx,id",
    "onlineDate": "2017-06-06T11:00:00.000Z",
    "chipName": "Albatross",
    "id": "2bcc3cdb587d1bef305ac14447b9b0a6",
    "topologyId": "db99eef232f426b45d2d147359580bc6",
    "url": "https://ibm.biz/qiskit-ibmqx3",
    "simulator": false,
    "nQubits": 16,
    "couplingMap": [
        ...
    ]
}, {
    "name": "QS1_1",
    "version": "1",
    "status": "standby",
    "serialNumber": "QS1_1",
    "description": "20 qubit device v1",
    "basisGates": "SU2+CNOT",
    "onlineDate": "2017-10-20T11:00:00.000Z",
    "chipName": "Qubert",
    "id": "cb141f7bb641b8a10487a6fab8483b86",
    "topologyId": "25197b9b73c4b52ca713ca4d126417b5",
    "simulator": false,
    "nQubits": 20,
    "couplingMap": [
        ...
    ]
}, {
    "name": "ibmqx_qasm_simulator",
    "status": "on",
    "description": "online qasm simulator",
    "basisGates": "u1,u2,u3,cx,id",
    "id": "18da019106bf6b5a55e0ef932763a670",
    "topologyId": "250e969c6b9e68aa2a045ffbcecb3ac33",
    "simulator": true,
    "nQubits": 24,
    "couplingMap": "all-to-all"
}]

```

В листинге 3.1 показан текущий список доступных процессоров, который по большей части совпадает с официальным перечнем с сайта IBM Q Experience. Тем не менее там есть много дополнительной интересной информации о конструктивных схемах машин.

○ Дополнительные процессоры и симуляторы.

- Похоже, что для использования доступны два удаленных симулятора — `ibmqx_qasm_simulator` и `ibmqx_hqc_qasm_simulator`, хотя в официальной документации упоминается только `ibmqx_qasm_simulator`. Эта информация может пригодиться при тестировании сложных схем: чем больше симуляторов, тем лучше.
- Уже давно ходят слухи о 20-кубитном процессоре. Поговаривают даже о запланированном выходе 50-кубитного монстра к концу 2018 года<sup>1</sup>. Этот список, по-видимому, подтверждает по крайней мере существование 20-кубитной машины. Но пока рано радоваться, она будет доступна только для корпоративных клиентов.

○ Помимо обычной информации, такой как название машины, версия, состояние, количество кубитов и т. д., есть термины, с которыми мы должны ознакомиться.

- *basisGates* — физические кубитные вентили процессора. Это основа, на которой могут быть построены более сложные логические элементы. Большинство процессоров в списке используют `u1`, `u2`, `u3`, `cx`, `id`:
  - вентили `u1`, `u2`, `u3` называются *частичными НЕ-вентильями*, они вращают кубит вокруг осей  $X$ ,  $Y$ ,  $Z$  на  $\theta$ ,  $\phi$  или  $\lambda$  радиан;
  - `cx` называется *управляемым НЕ-вентилем* (CNOT или CX). Он задействует два кубита и выполняет операцию НЕ на втором кубите, только когда первый кубит находится в состоянии  $|1\rangle$ , и оставляет его неизменным в противном случае;
  - `id` — это единичный вентиль, который выполняет операцию простоя (*idle*) на кубите в течение одной единицы времени.

---

<sup>1</sup> IBM уже предоставляет доступ к Q System One с 20 кубитами и в ближайшее время откроет для партнеров доступ к квантовому компьютеру с 50 кубитами. — *Примеч. ред.*

- *couplingMap* — карта связей. Определяет взаимодействия между отдельными кубитами, сохраняя при этом квантовую когерентность (или чистое состояние — представьте, что солдаты, переходя реку по старому мосту, идут не в ногу, чтобы амплитуды их шагов не сошлись и это не привело к разрушению моста). Связывание кубитов в пары используется для упрощения квантовой схемы и позволяет разбить систему на более мелкие единицы.

Теперь вернемся к Composer, чтобы создать первую квантовую композицию.

## Опуск 1: вариации на тему состояний Белла и GHZ

Здесь мы рассмотрим два умопомрачительных квантовых эксперимента, используемых для демонстрации странности квантовой механики:

- *состояния Белла* — показывают, что физика не описывается локальной реальностью. Это то, что Эйнштейн назвал *мистическим дальнодействием*;
- *GHZ-состояния* — даже еще более странные, чем состояния Белла, GHZ-состояния (названные в честь своих создателей Гринбергера, Хорна и Цейлингера) являются трехкубитным обобщением состояний Белла.

Рассмотрим их подробнее.

### Состояния Белла и мистическое дальнодействие

Состояния Белла являются экспериментальной проверкой известных неравенств Белла. В 1964 году ирландский физик Джон Белл предложил способ проверки квантовой запутанности (мистического дальнодействия). Он вывел ряд неравенств, которые стали очень востребованы в физическом сообществе. Они известны как неравенства Белла (сегодня называют теоремой Белла).

Рассмотрим поляризацию фотона (когда свет колеблется в определенной плоскости) под тремя углами:  $A = 0^\circ$ ,  $B = 120^\circ$  и  $C = 240^\circ$ . Здравый смысл подсказывает нам, что фотон имеет определенные значения одновременно для этих трех настроек поляризации и они должны соответствовать восьми случаям, приведенным в табл. 3.3.

**Таблица 3.3.** Перестановки для поляризации фотонов под тремя углами

Номер	$A (0^\circ)$	$B (120^\circ)$	$C (240^\circ)$	$[AB]$	$[BC]$	$[AC]$	Сумма	Среднее
1	$A+$	$B+$	$C+$	1(++)	1(++)	1(++)	3	1
2	$A+$	$B+$	$C-$	1(++)	0	0	1	1/3
3	$A+$	$B-$	$C+$	0	0	1(++)	1	1/3
4	$A+$	$B-$	$C-$	0	1(--)	0	1	1/3
5	$A-$	$B+$	$C+$	0	1(++)	0	1	1/3
6	$A-$	$B+$	$C-$	0	0	1(--)	1	1/3
7	$A-$	$B-$	$C+$	1(--)	0	0	1	1/3
8	$A-$	$B-$	$C-$	1(--)	1(--)	1(--)	3	1

Теперь по теореме Белла зададимся вопросом: какова вероятность того, что поляризация у любого соседа будет такой же, как у первого? Мы также рассчитываем сумму и среднее значение поляризаций. Если предположить, что в реальности все именно так, то в соответствии с табл. 3.3 ответ на этот вопрос таков: вероятность должна быть больше или равна 1/3. Это то, что дает неравенство Белла: средство проверить данное утверждение. Но вот что невероятно: хотите верьте, хотите нет, квантовая механика противоречит неравенству Белла, давая вероятности менее 1/3. Это было экспериментально доказано в 1982 году французским физиком Аленом Аспе.

#### ПРИМЕЧАНИЕ

Более подробное описание эксперимента Аспе и неравенства Белла приведено в разделе «ЭПР-парадокс разгромлен: Бор смеется последним» главы 1.

Итак, теперь переведем поляризацию фотонов из табл. 3.3 в эксперимент, который можно запустить на квантовом компьютере. В 1969 году Джон

Клаузер, Майкл Хорн, Абнер Шимони и Ричард Холт представили доказательство теоремы Белла — неравенство Клаузера — Хорна — Шимони — Холта, которое формально утверждает<sup>1</sup>:

$$S = \langle A, B \rangle - \langle A, B' \rangle + \langle A', B \rangle + \langle A', B' \rangle;$$

$$S \leq 2.$$

Проиллюстрировать это нам помогут два персонажа, Алиса и Боб. На стороне Алисы имеются настройки детектора  $A$  и  $A'$ , на стороне Боба —  $B$  и  $B'$ , а также четыре комбинации для проверки в отдельных экспериментах. Реальное положение дел свидетельствует о том, что для пары запутанных частиц таблица четности, показывающая все возможные перестановки, выглядит следующим образом.

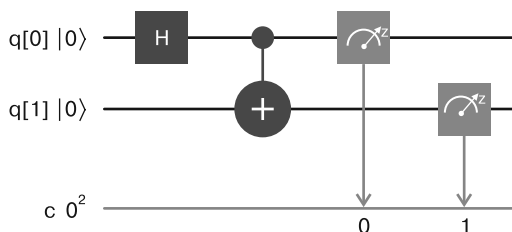
$A$	$B$	1
$A$	$B'$	0
$A'$	$B$	0
$A'$	$B'$	1

В классическом реализме неравенство Клаузера — Хорна — Шимони — Холта превращается в  $|S| = 2$ . Однако математический формализм квантовой механики предсказывает максимальное значение для  $S$ , соответствующее  $|S| = 2\sqrt{2}$ , что нарушает это неравенство. Это можно проверить с помощью четырех отдельных квантовых схем (по одной на измерение) по два кубита в каждой. Для упрощения задачи примем, что измерения на детекторе Алисы  $A = Z$  и  $A' = X$ , а на детекторе Боба  $B = W$  и  $B' = V$  (табл. 3.4). Чтобы начать эксперимент, следует построить базовое состояние Белла (рис. 3.4), которое соответствует тождеству  $1/\sqrt{2}(|00\rangle + |11\rangle)$ .

Предыдущее выражение, по существу, означает, что кубит, находящийся у Алисы, может иметь значение 0 или 1. Если бы Алиса измерила свой кубит в стандартном базисе, результат был бы совершенно случайным с вероятностью  $1/2$  для каждого возможного варианта. И если бы Боб измерил свой кубит, результат был бы таким же, как у Алисы. Итак, если бы измерения выполнил Боб, то также получил бы результат, на первый взгляд кажущийся случайным, а если бы Алиса и Боб общались,

<sup>1</sup> Скобки задают корреляции между результатами измерения (среднее произведения отдельных исходов экспериментов  $A$  и  $B$ ). — *Примеч. науч. ред.*

они бы обнаружили, что, хоть результаты казались случайными, они коррелируют.



**Рис. 3.4.** Базовое состояние Белла

На рис. 3.4 два кубита первоначально находятся в основном состоянии  $|0\rangle$ . Вентиль H создает суперпозицию первого кубита, соответствующую состоянию  $1/\sqrt{2}(|00\rangle + |10\rangle)$ . Затем вентиль CNOT инвертирует второй кубит, если первый возбужден, создавая состояние  $1/\sqrt{2}(|00\rangle + |11\rangle)$ . Это исходное запутанное состояние, необходимое для четырех измерений в табл. 3.4 (все материалы публикуются с разрешения © International Business Machines Corporation).

- Для вращения измерительного базиса вокруг оси  $ZW$  используйте серию вентилях S-H-T-H.
- Для поворота измерительного базиса вокруг оси  $ZV$  задействуйте серию вентилях S-H-T-H.
- Измерения  $XW$  и  $XV$  выполняются так же, как описано ранее, и измерение  $X$  через вентиль Адамара проводится перед стандартным измерением.

### СОВЕТ

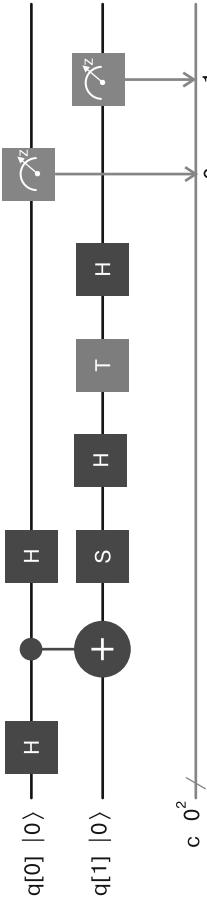
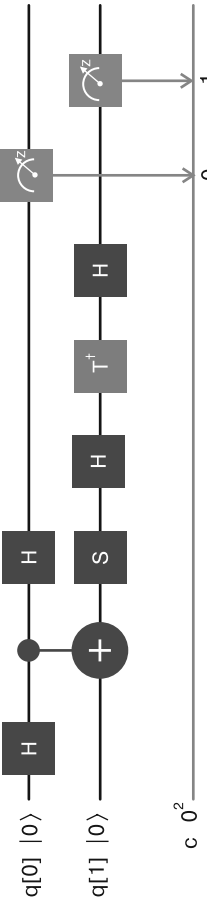
Перед выполнением эксперимента в Composer убедитесь, что для его топологии (количество кубитов и целевое устройство) в партитуре установлено значение 2 для симулятора. Некоторые топологии (например, пять кубитов в реальном квантовом устройстве) не поддерживают запутывание для кубитов 0 и 1, что приводит к ошибкам при проектировании. Обратите внимание, что целевым устройством может быть настоящий квантовый процессор или симулятор. В общем, до тех пор, пока вы используете симулятор, у вас все будет хорошо.

Таблица 3.4. Квантовые схемы для состояний Белла

Измерение состояния Белла	Результаты для 100 запусков	
<div><b>AB (ZW)</b>  <math>c \ 0^2</math></div>	c[2] 11 10 00 01	Вероятность 0,39 0,06 0,46 0,09
<div><b>AB' (ZV)</b>  <math>c \ 0^2</math></div>	c[2] 11 10 00 01	Вероятность 0,49 0,07 0,36 0,08

Продолжение ⇨

Таблица 3.4 (продолжение)

Измерение состояния Белла		Результаты для 100 запусков	
<b>A'B' (XIV)</b>		c[2] 11 10 00 01	Вероятность 0,42 0,05 0,49 0,04
<b>A'B' (XV)</b>		c[2] 11 10 00 01	Вероятность 0,05 0,52 0,03 0,40



Теперь нужно построить таблицу с результатами каждого измерения плюс вероятность корреляции между  $A$  и  $B$   $\langle AB \rangle$ . Сумма вероятностей для четности запутанных частиц определяется как:

$$\langle AB \rangle = P(1,1) + P(0,0) - P(1,0) - P(0,1).$$

Помните, что конечная цель — определить, что верно:  $S \leq 2$  или  $|S| = 2$ . Таким образом, скомпоновав результаты всех измерений, мы получаем табл. 3.5.

**Таблица 3.5.** Объединение результатов для экспериментов Белла

	$P(00)$	$P(11)$	$P(01)$	$P(10)$	$\langle AB \rangle$
$AB(ZW)$	0,46	0,39	0,09	0,06	0,68
$AB'(ZV)$	0,36	0,49	0,08	0,07	0,73
$A'B(XW)$	0,49	0,42	0,04	0,05	0,47
$A'B'(XV)$	0,03	0,05	0,40	0,52	−0,32

Складываем абсолютные значения из столбца  $\langle AB \rangle$  и получаем  $|S| = 2,2$ . Этот результат противоречит неравенству Белла (как предсказывает квантовая механика) и очень близок к результатам официальных испытаний, проведенных 2 мая 2017 года учеными из IBM, которые провели 8192 запуска. Как насчет ваших?

## Еще более необычно: проверка GHZ-состояний

GHZ-состояния названы в честь физиков Гринбергера, Хорна и Цейлингера, которые придумали обобщенную проверку для  $N$  запутанных кубитов. Наиболее простым является GHZ-состояние с тремя кубитами:

$$|\text{GHZ}\rangle = 1/\sqrt{2}(|000\rangle - |111\rangle).$$

### ПРИМЕЧАНИЕ

Важность GHZ-состояний заключается в том, что они показывают: запутывание более чем двух частиц приводит к конфликту с концепцией локального реализма не только для статистических (вероятностных), но и для нестатистических (детерминистских) предсказаний.

Проще говоря, GHZ-состояния демонстрируют более сильное нарушение неравенства Белла. Рассмотрим это на примере простой головоломки. Представьте три независимых блока, каждый из которых содержит две переменные,  $X$  и  $Y$ . Каждая переменная имеет два возможных значения: 1 и  $-1$ . Нужно найти для  $X$  и  $Y$  набор значений, который является решением следующего набора тождеств.

1.  $XYX = 1$ .
2.  $YXY = 1$ .
3.  $YYX = 1$ .
4.  $XXX = -1$ .

Для нетерпеливых: решения не существует. Например, замените  $Y = 1$  в тождествах 1, 2 и 3, а затем перемножьте их. Набор примет следующий вид.

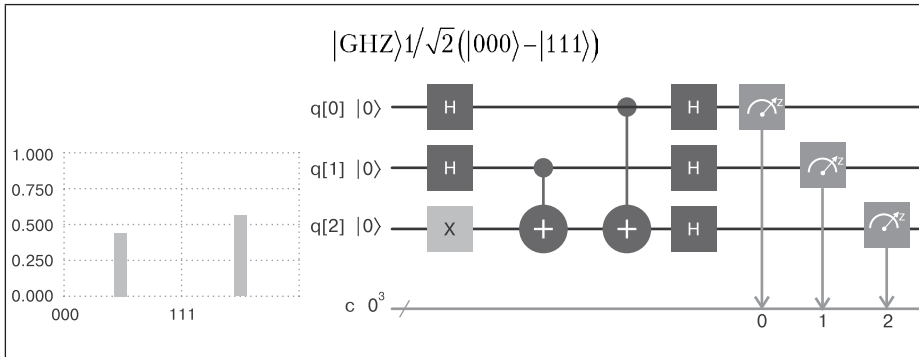
1.  $X11 = 1$ .
2.  $1X1 = 1$ .
3.  $11X = 1$ .
4.  $XXX = -1$ .
5. Перемножили тождества 1, 2 и 3 и получили  $XXX = 1$ .

Решения не существует, потому что тождество 4  $XXX = -1$  противоречит тождеству 5  $XXX = 1$ . Пугает то, что GHZ-состояние действительно может дать решение этой задачи, что кажется невозможным в детерминистском представлении классической реальности. Но в мире квантовой механики нет ничего невозможного, есть просто маловероятное.

Удивительно, но тесты GHZ могут с уверенностью исключить описание локальной реальности после одного прогона эксперимента, однако сначала мы должны построить базовое GHZ-состояние.

Базовое GHZ-состояние для запуска эксперимента (так же как и результаты для вероятности, которые должны составлять около половины) показано в табл. 3.6.

Таблица 3.6. Базовое GHZ-состояние



1. В базовой схеме вентиль Адамара, примененный к кубитам 1 и 2, переводит их в суперпозицию  $|00,01,01,11\rangle$ . В то же время вентиль X производит операцию отрицания над кубитом 3. Таким образом, мы получаем состояния  $1/\sqrt{2}(|001\rangle + |101\rangle + |011\rangle + |111\rangle)$
2. Два вентиля CNOT запутывают все кубиты в следующее состояние:  $1/\sqrt{2}(|001\rangle + |010\rangle + |100\rangle + |111\rangle)$ .
3. Наконец, три вентиля Адамара преобразуют шаг 2 в следующее состояние:  $1/2(|000\rangle - |111\rangle)$ .

Теперь создайте квантовые схемы для тождеств  $XY\bar{Y}$ ,  $Y\bar{X}Y$ ,  $X\bar{Y}\bar{Y}$  и  $XXX$  из предыдущего раздела, как показано в табл. 3.7 (все материалы публикуются с разрешения © International Business Machines Corporation).

- Для измерения  $X$  примените вентиль  $H$  к соответствующему кубиту.
- Для каждого экземпляра  $Y$  примените  $S^\dagger$  (ИЛИ-НЕ к  $S$ ) и вентиль  $H$  к соответствующему кубиту.

Наконец, сравните результаты предыдущего эксперимента с официальными данными из IBM Q Experience. Каков ваш итоговый результат? В целом принципы квантовой механики, описанные в этом разделе, были поставлены под сомнение теорией, названной супердетерминизмом, которая дает возможность выйти из сложившейся ситуации.

Таблица 3.7. Квантовые схемы для GHZ-состояний

Измерение		Результаты для 100 запусков	
УУХ		c[3] 011 101 110 000	Вероятность 0,34 0,23 0,23 0,20
УХУ		c[3] 011 101 110 000	Вероятность 0,23 0,28 0,25 0,24

Измерение	Результаты для 100 запусков	
<div><b>xyy</b> </div>	<div>c[3] 011 101 110 000</div>	<div>Вероятность 0,23 0,26 0,35 0,16</div>
<div><b>xxx</b> </div>	<div>c[3] 010 100 111 001</div>	<div>Вероятность 0,25 0,32 0,22 0,21</div>

## **Супердетерминизм: уход от мистичности. Был ли Эйнштейн прав все это время?**

В интервью BBC в 1969 году физик Джон Белл рассказал о своей работе в области квантовой механики. Он сказал, что мы должны принять предположение о том, что действия между запутанными частицами передаются со скоростью большей, чем скорость света, но в то же время мы никак не можем это использовать. Информация не может распространяться со скоростью большей, чем скорость света, что также предсказывает квантовая механика. Как будто природа подшучивает над нами.

Он также упомянул, что существует возможность решить эту загадку — принцип под названием «супердетерминизм».

Запутывание частиц подразумевает, что измерения, выполненные на одной частице, мгновенно влияют на другую, даже отстоящую от нее на большое расстояние (представьте противоположные стороны Галактики или Вселенной), даже когда они разделены во времени. Эйнштейн был яростным противником этой теории: вспомните его знаменитое послание Нильсу Бору о том, что Бог не бросает кости. Он не мог принять вероятностный характер квантовой механики, поэтому в 1935 году вместе с коллегами Подольским и Розеном придумал печально известный парадокс ЭПР, чтобы оспорить ее основы.

Согласно парадоксу ЭПР, измерение в одной из двух запутанных частиц, разделенных огромным расстоянием, не может мгновенно повлиять на другую, поскольку событие должно будет проходить со скоростью большей, чем скорость света (предел максимальной скорости во Вселенной). Это противоречит общей теории относительности, создавая парадокс: ничто не движется быстрее скорости света, что является абсолютным законом относительности.

Тем не менее в 1982 году предположения квантовой механики были подтверждены французским физиком Аленом Аспе. Он провел эксперимент, который показал, что неравенство Белла нарушается запутанными фотонами. Он также доказал, что измерение на одном из запутанных фотонов распространяется со скоростью большей, чем скорость света, чтобы передать его состояние другому.

С тех пор результаты, полученные Аспе, вновь и вновь подтверждаются (подробности эксперимента приведены в главе 1). Ирония заключается в следующем: есть вероятность, что Эйнштейн был прав с самого начала, а запутанность — всего лишь иллюзия. Это принцип супердетерминизма.

---

### ПРИМЕЧАНИЕ

Проще говоря, супердетерминизм говорит, что с самого начала существования Вселенной свободы выбора не было. Все корреляции и запутанность частиц были заданы в момент Большого взрыва. Таким образом, нет необходимости в сигнале, передающемся со скоростью, превышающей скорость света, чтобы сообщать частице В, каков результат частицы А.

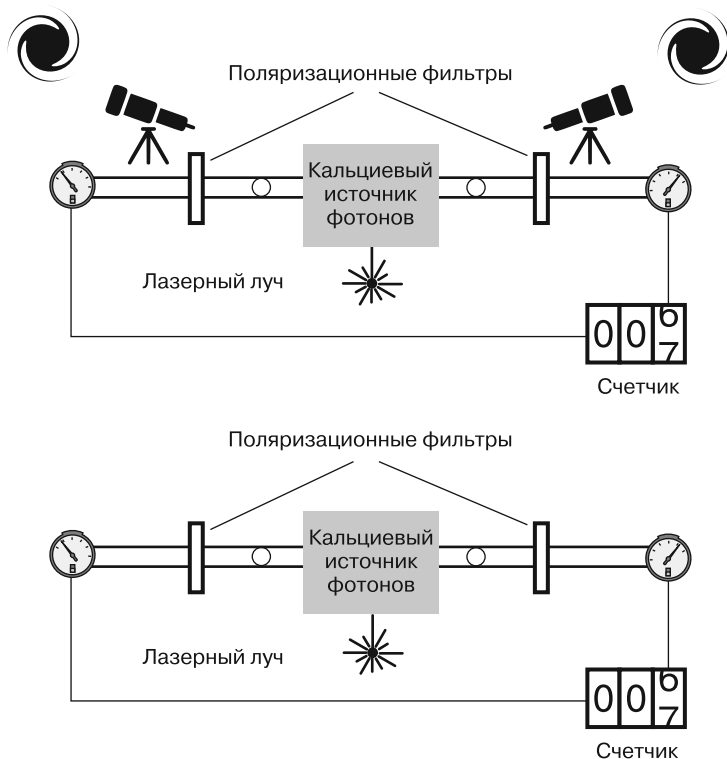
---

Если это верно, данная уловка докажет, что Эйнштейн был прав, когда предположил парадокс ЭПР, и вся наша тяжелая работа в области квантового программирования — всего лишь иллюзия. Но этот принцип звучит скорее как догма, ведь согласно ему все результаты предопределены судьбой, тогда как Белл с научной точки зрения утверждал, что супердетерминизм неправдоподобен. Он рассуждает о том, что свобода выбора фактически ничем не ограничена для поставленной цели из-за изменений, вызванных множеством очень мелких эффектов. Супердетерминизм назвали непроверяемым, поскольку экспериментаторы никогда не смогут устранить корреляции, которые были созданы в начале существования Вселенной. Однако это не заставило ученых отказаться от попыток доказать, что Эйнштейн прав, а запутывание частиц — иллюзия. На самом деле существует труднореализуемый оригинальный эксперимент, который призван разрешить все противоречия. Рассмотрим его.

На рис. 3.5 проиллюстрирован стандартный эксперимент по проверке неравенства Белла (*внизу*) и вариант эксперимента с использованием космических фотонов (*вверху*), выполненные Эндрю Фридманом и его коллегами из MIT<sup>1</sup>.

---

<sup>1</sup> Gallicchio J., Friedman A. S., Kaiser D. I. Testing Bell's Inequality with Cosmic Photons: Closing the Setting-Independence Loophole // [http://web.mit.edu/asf/www/Papers/Gallicchio\\_Friedman\\_Kaiser\\_2014.pdf](http://web.mit.edu/asf/www/Papers/Gallicchio_Friedman_Kaiser_2014.pdf).



**Рис. 3.5.** Эксперимент по проверке нарушения неравенства Белла с космическими фотонами в сравнении со стандартной проверкой

#### ПРИМЕЧАНИЕ

Полное описание стандартной проверки неравенства Белла смотрите в разделе «ЭПР-парадокс разгромлен: Бор смеется последним» главы 1.

Фридман и его коллеги предложили новый вариант стандартного эксперимента Белла с использованием космических лучей. Суть его состоит в том, чтобы использовать астрономические наблюдения в реальном времени за далекими звездами нашей Галактики, далекими квазарами или пятнами космического микроволнового фона, чтобы, по существу,



позволить Вселенной решить, как организовать эксперимент, вместо того чтобы задействовать стандартный генератор квантовых случайных чисел. То есть использовать фотоны из далеких галактик для управления ориентацией поляризационных фильтров непосредственно перед прибытием запутанных фотонов.

В случае успеха последствия будут грандиозными. Если результаты такого эксперимента не нарушат неравенства Белла, это будет означать, что супердетерминизм в конце концов может оказаться верным. Запутывание частиц будет иллюзией, и сигнал между запутанными частицами не может передаваться быстрее, чем свет, как предсказывает теория относительности. Эйнштейн окажется прав: мистических дальнодействий не существует.

К счастью для нас, любителей квантовой механики, до сих пор ничего подобного не происходило. Имейте в виду, что Фридман и его коллеги — не единственная команда, участвующая в эксперименте.

Есть несколько коллективов, пытающихся разгадать эту загадку. И большинство их результатов согласуется с квантовой механикой. Иначе говоря, они нарушают неравенство Белла. Кажется, что разрыв между позициями Эйнштейна и Бора в борьбе теории относительности с квантовой механикой давно устранен. Я все еще делаю ставку на квантовую механику.

В следующем разделе показано, как получить доступ к IBM Q Experience удаленно с помощью удобного REST API.

## Удаленный доступ через REST API

Q Experience имеет относительно малоизвестный REST API, который скрыто обрабатывает все удаленные коммуникации. Используется текущими Python SDK.

- *QISKit* (Quantum Information Science Kit) является платформой квантовых вычислений с открытым кодом для квантового программирования на Python.

- *IBMQExperience* — менее известная библиотека в комплекте с QISKit, которая упаковывает REST API в клиент Python.

В этом разделе мы заглянем в IBMQExperience и рассмотрим различные конечные точки REST для удаленного доступа. Но сначала требуется аутентификация.

## Аутентификация

Прежде чем выполнить любой вызов REST API, мы должны получить токен доступа. Это будет ключ доступа для любых вызовов в этом разделе. Обратите внимание, что токен доступа не совпадает с токеном API (токен API задействуется для выполнения квантовых программ на Python). Существует два способа получения токена доступа.

- *Использование токена API.* Чтобы получить токен API, войдите в консоль IBM Q Experience и выполните инструкции, приведенные в следующем разделе.
- *Использование имени пользователя и пароля вашей учетной записи.* Давайте посмотрим, как это делается с помощью REST.

---

### СОВЕТ

Чтобы получить токен API, войдите в консоль IBM Q Experience, выберите свое имя пользователя, затем My Account (Моя учетная запись) и перейдите на вкладку Advanced (Дополнительно), расположенную в правом верхнем углу. Далее нажмите кнопку Generate (Генерировать), а затем скопируйте токен API (рис. 3.6). Следите за тем, чтобы токен был в безопасности.

---

Аутентификация через токены API:

- метод HTTP: POST;
- URL: <https://quantumexperience.ng.bluemix.net/api/users/loginWithToken>;
- полезные данные: {"apiToken": "YOUR\_API\_TOKEN"}.

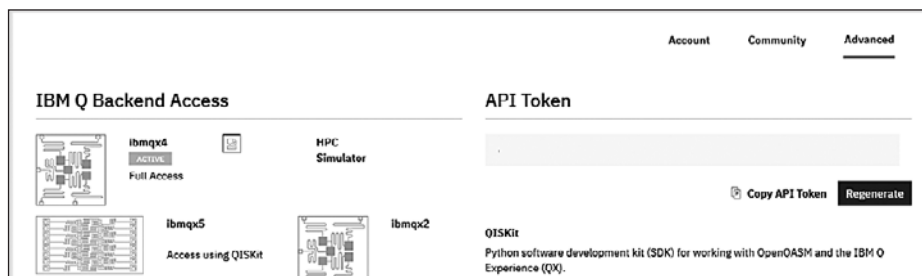


Рис. 3.6. Получение токена API из консоли

Аутентификация с помощью пароля пользователя:

- метод HTTP: POST;
- URL: <https://quantumexperience.ng.bluemix.net/api/users/login>;
- полезные данные: {"email": "USER-NAME", "password": "YOUR-PASSWORD"}.

Ответ для обоих методов:

```
{
  "id": "ACCESS_TOKEN",
  "ttl": 1209600,
  "created": "2018-04-15T20:21:03.204Z",
  "userId": "USER-ID"
}
```

Здесь `id` — это ваш токен доступа, `ttl` — время жизни (или время окончания действия) в миллисекундах, а `userId` — ваш идентификатор пользователя. Сохраните токен доступа и идентификатор пользователя для выполнения заданий этого раздела. Обратите внимание, что по истечении сеанса необходимо создать новый токен доступа.

## Перечисление доступного серверного ПО

Этот вызов возвращает список всего доступного серверного ПО и симуляторов в IBM Q Experience в формате JSON:

- метод HTTP: GET;
- URL: [https://quantumexperience.ng.bluemix.net/api/Backends?access\\_token=ACCESS-TOKEN](https://quantumexperience.ng.bluemix.net/api/Backends?access_token=ACCESS-TOKEN).

## Параметры запроса

Наименование	Значение
access_token	Токен доступа к вашему аккаунту

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-ru

## Пример ответа

Типом содержимого ответа для всех вызовов API является `application/json`. Следующий абзац показывает частичный результат вызова этой конечной точки. Обратите внимание, что она будет возвращать как реальные процессоры, так и симуляторы:

```
[{
  "name": "ibmqx2",
  "version": "1",
  "status": "on",
  "serialNumber": "Real5Qv2",
  "description": "5 transmon bowtie",
  "basisGates": "u1,u2,u3,cx,id",
  "onlineDate": "2017-01-10T12:00:00.000Z",
  "chipName": "Sparrow",
  "id": "28147a578bdc88ec8087af46ede526e1",
  "topologyId": "250e969c6b9e68aa2a045ffbceb3ac33",
  "url": "https://ibm.biz/qiskit-ibmqx2",
  "simulator": false,
  "nQubits": 5,
  "couplingMap": [
    [0, 1],
    [0, 2],
    [1, 2],
    [3, 2],
    [3, 4],
    [4, 2]
  ]
},...]
```

Наиболее важные ключи из предыдущего ответа описаны в табл. 3.8.

**Таблица 3.8.** Ключи для ответа на запрос о доступном серверном ПО

Ключ	Описание
Name	Идентификатор имени процессора, который будет применяться при выполнении кода
Version	Строка или положительное целое число, по всей вероятности, используемое для отслеживания изменений в процессоре
Description	<p>Это, вероятно, описание оборудования, применяемого для сборки чипа. Вы можете увидеть что-то вроде:</p> <ul style="list-style-type: none"> <li>• развязка из пяти трансмонов;</li> <li>• цепная схема <math>2 \times 8</math> из 16 трансмонов.</li> </ul> <p>Примечание: трансмон — тип помехоустойчивого сверхпроводящего кубита. Разработан Робертом Дж. Шозлкоффом, Мишелем Деворе, Стивеном М. Гирвином и их коллегами из Йельского университета в 2007 году</p>
basisGates	Это физические кубитные вентили процессора. Они являются основой, на которой могут быть построены более сложные логические вентили
nQubits	Количество кубитов, используемых процессором
couplingMap	Карта связей определяет взаимодействия между отдельными кубитами, сохраняя при этом квантовую когерентность. С ее помощью можно упростить квантовую схему и разбить систему на более мелкие единицы

## Получение информации о калибровке заданного процессора

Результатом этого вызова является список в формате JSON калибровочных параметров для заданного процессора в Q Experience. Эти параметры подробно описаны на информационном сайте серверного ПО IBMQX:

- метод HTTP: GET;
- URL: [https://quantumexperience.ng.bluemix.net/api/Backends/NAME/calibration?access\\_token=ACCESS-TOKEN](https://quantumexperience.ng.bluemix.net/api/Backends/NAME/calibration?access_token=ACCESS-TOKEN).

### Параметры запроса

Наименование	Значение
access_token	Токен доступа к вашему аккаунту

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-ру (значение по умолчанию для официального клиента, также, я предполагаю, оно может быть любым)

## Пример ответа

Кубиты очень чувствительны к ошибкам и шуму окружающей среды. Информация о калибровке дает представление о качестве кубитов внутри процессора. В листинге 3.2 показан упрощенный ответ с параметрами калибровки для ibmqx4. Некоторые из наиболее примечательных параметров:

- `gateError` — частота возникновения ошибок для операции применения вентилей на кубите в данный момент времени;
- `readoutError` — частота возникновения ошибок для операции считывания данных кубита в данный момент времени.

---

### ПРИМЕЧАНИЕ

Оценка качества кубитов включает четыре этапа (операции): подготовку, запоминание, применение вентилей и считывание. Для отслеживания качества кубита частота ошибок рассчитывается на этапах применения вентилей и считывания. Это информация, возвращаемая в ответе на данный вызов API. Обратите внимание на то, что после использования кубиты необходимо сбросить (охладить) до базового состояния.

---

**Листинг 3.2.** Упрощенная форма ответа на запрос о калибровочных параметрах для ibmqx4

```
{
  "lastUpdateDate": "2018-04-15T10:47:03.000Z",
  "qubits": [{
    "gateError": {
```

```

        "date": "2018-04-15T10:47:03Z",
        "value": 0.0012019552727863259
      },
      "name": "Q0",
      "readoutError": {
        "date": "2018-04-15T10:47:03Z",
        "value": 0.049
      }
    }, ...
  ],
  "multiQubitGates": [{
    "qubits": [1, 0],
    "type": "CX",
    "gateError": {
      "date": "2018-04-15T10:47:03Z",
      "value": 0.03024023736391171
    },
    "name": "CX1_0"
  }, ...
]}

```

Информацию, приведенную в листинге 3.2, можно увидеть на вкладке **Devices** (Устройства) консоли IBM Q Experience, в главном меню (рис. 3.7). Получите информацию о калибровке через REST и сравните ее с отображенной на веб-консоли (все материалы публикуются с разрешения © International Business Machines Corporation).

## Получение параметров серверного ПО

В результате данного вызова будет возвращен список параметров серверного ПО для заданного процессора в Q Experience в формате JSON. Некоторые из этих параметров:

- температура охлаждения кубита в Кельвинах: Например, я получил значение 0,021 К для `ibmqx4`, что является сверхнизкой температурой и соответствует  $-459,6^{\circ}\text{F}$  или  $-273,1^{\circ}\text{C}$ ;
- время буфера (нс);
- время вентиля (нс).

Другие квантовые спецификации более подробно описаны на информационном сайте для серверного ПО.

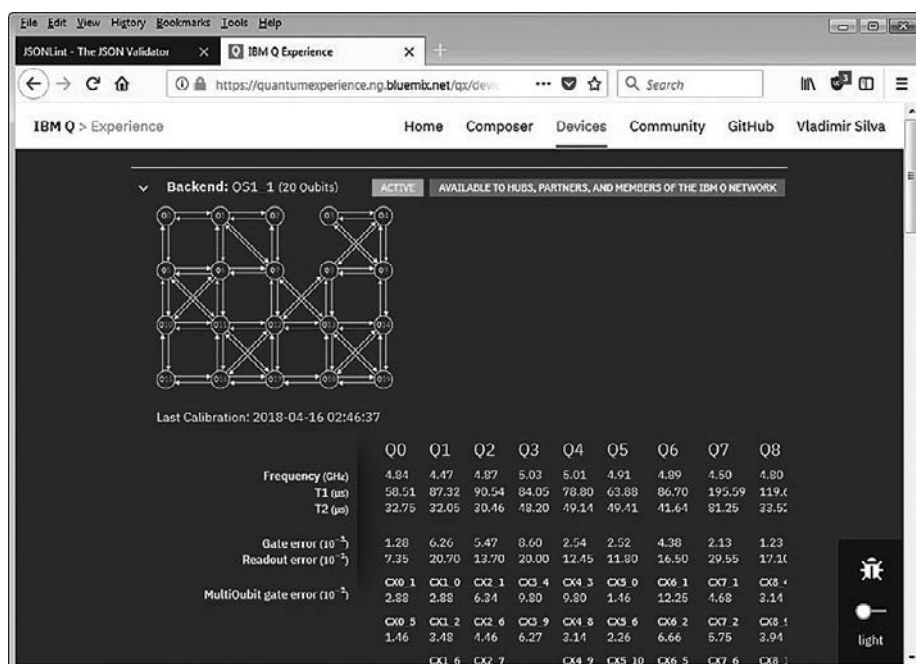


Рис. 3.7. Информация о калибровке, переданная веб-консолю

Тип запроса и конечная точка URL:

- метод HTTP: GET;
- URL: [https://quantumexperience.ng.bluemix.net/api/Backends/NAME/parameters?access\\_token=ACCESS-TOKEN](https://quantumexperience.ng.bluemix.net/api/Backends/NAME/parameters?access_token=ACCESS-TOKEN).

## Параметры запроса

Наименование	Значение
access_token	Токен доступа к вашему аккаунту

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-py



## Пример ответа

В листинге 3.3 приведен упрощенный ответ с параметрами `ibmqx4` в формате JSON.

**Листинг 3.3.** Упрощенный ответ на запрос о параметрах `ibmqx4`

```
{
  "lastUpdateDate": "2018-04-15T10:47:03.000Z",
  "fridgeParameters": {
    "cooldownDate": "2017-09-07",
    "Temperature": {
      "date": "2018-04-15T10:47:03Z",
      "value": 0.021,
      "unit": "K"
    }
  },
  "qubits": [{
    "name": "Q0",
    "buffer": {
      "date": "2018-04-15T10:47:03Z",
      "value": 10,
      "unit": "ns"
    },
    "gateTime": {
      "date": "2018-04-15T10:47:03Z",
      "value": 50,
      "unit": "ns"
    },
    "T2": {
      "date": "2018-04-15T10:47:03Z",
      "value": 16.5,
      "unit": "µs"
    },
    "T1": {
      "date": "2018-04-15T10:47:03Z",
      "value": 45.2,
      "unit": "µs"
    },
    "frequency": {
      "date": "2018-04-15T10:47:03Z",
      "value": 5.24208,
      "unit": "GHz"
    }
  }, ...]
}
```

## Получение статуса очереди процессора

Этот вызов возвращает состояние определенной очереди событий квантового процессора:

- метод HTTP: GET;
- URL: <https://quantumexperience.ng.bluemix.net/api/Backends/NAME/queue/status>.

## Параметры запроса

Как ни странно, данный вызов API не запрашивает токен доступа.

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-py

## Пример ответа

Например, чтобы получить очередь событий для `ibmqx4`, скопируйте следующий URL в адресную строку своего браузера: <https://quantumexperience.ng.bluemix.net/api/Backends/ibmqx4/queue/status>.

Ответ выглядит как `{"state":true, "status":"active", "lengthQueue":0}`, где:

- `state`: — состояние процессора. Если он работает, значение `true`, иначе — `false`;
- `status`: — состояние очереди выполнения, значения — активно или занято;
- `lengthQueue`: — размер очереди выполнения или количество симуляций, ожидающих выполнения.

---

### ПРИМЕЧАНИЕ

Когда вы отправляете эксперимент в IBM Q Experience, он попадает в очередь выполнения. Этот вызов API полезен для отслеживания загрузки процессора в данный момент.

---

## Перечисление заданий в очереди выполнения

Этот вызов возвращает список заданий в очереди выполнения процессора:

- метод HTTP: GET;
- URL: `https://quantumexperience.ng.bluemix.net/api/Jobs?access_token=ACCESS-TOKEN&filter=FILTER`.

### Параметры запроса

Наименование	Значение
access_token	Токен доступа к вашему аккаунту
filter	Указание размера результата в JSON. Например, {"limit":2} возвратит не более двух записей

### Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-py

### Пример ответа

В листинге 3.4 показан формат ответа на данный вызов. Судя по всему, эта информация представляет собой хронологию выполненных экспериментов, где указаны статус, даты, результаты, код, калибровка и многое другое.

#### Листинг 3.4. Упрощенная форма ответа на вызов API Get Jobs

```
[{
  "qasms": [{
    "qasm": "...",
    "status": "DONE",
    "executionId": "331f15a5eed1a4f72aa2fb4d96c75380",
    "result": {
      "date": "2018-04-05T14:25:37.948Z",
      "data": {
```

```

        "creg_labels": "c[5]",
        "additionalData": {
            "seed": 348582688
        },
        "time": 0.0166247,
        "counts": {
            "11100": 754,
            "01100": 270
        }
    }
}
}],
"shots": 1024,
"backend": {
    "name": "ibmqx_qasm_simulator"
},
"status": "COMPLETED",
"maxCredits": 3,
"usedCredits": 0,
"creationDate": "2018-04-05T14:25:37.597Z",
"deleted": false,
"id": "d405c5829274d0ee49b190205796df87",
"userId": "ef072577bd26831c59ddb212467821db",
"calibration": {}
}, ...]

```

---

#### ПРИМЕЧАНИЕ

В зависимости от размера очереди выполнения вы можете получить пустой результат ([ ]), если в очереди нет заданий, или результат в формате, показанном в листинге 3.4.

---

В любом случае убедитесь, что код ответа HTTP — 200 (ОК).

## Получение информации о балансе аккаунта

При создании учетной записи каждому пользователю присваивается определенное количество кредитов на выполнение (по умолчанию 15),

которые расходуются при проведении экспериментов. Этот вызов предоставляет информацию о балансе вашего аккаунта:

- метод HTTP: GET;
- URL: `https://quantumexperience.ng.bluemix.net/api/users/USER-ID?access_token=ACCESS-TOKEN`.

---

### СОВЕТ

Идентификатор пользователя можно получить из ответа на запрос аутентификации через токен API или с помощью пароля пользователя. Подробнее см. в подразделе «Аутентификация» данного раздела на с. 130.

---

## Параметры запроса

Наименование	Значение
access_token	Токен доступа к вашему аккаунту

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-py

## Пример ответа

В листинге 3.5 показан типичный ответ на данный вызов.

### Листинг 3.5. Типичный ответ на запрос о балансе аккаунта

```
{
  "institution": "Private Research",
  "status": "Registered",
  "blocked": "None",
  "dpl": {
```

```
    "blocked": false,
    "checked": false,
    "wordsFound": {},
    "results": {}
  },
  "credit": {
    "promotional": 0,
    "remaining": 150,
    "promotionalCodesUsed": [],
    "lastRefill": "2018-04-12T14:05:09.136Z",
    "maxUserType": 150
  },
  "additionalData": {
  },
  "creationDate": "2018-04-01T15:36:16.344Z",
  "username": "",
  "email": "",
  "emailVerified": true,
  "id": "",
  "userId": "...",
  "firstName": "...",
  "lastName": "..."
}
```

## Список экспериментов пользователя

Ответом на такой вызов является список всех экспериментов для заданного идентификатора пользователя:

- метод HTTP: GET;
- URL: [https://quantumexperience.ng.bluemix.net/api/users/USER-ID/codes/latest?access\\_token=ACCESS-TOKEN&includeExecutions=true](https://quantumexperience.ng.bluemix.net/api/users/USER-ID/codes/latest?access_token=ACCESS-TOKEN&includeExecutions=true).

## Параметры запроса

Наименование	Значение
USER-ID	Ваш идентификатор пользователя, полученный на этапе аутентификации
access_token	Токен доступа к вашему аккаунту
includeExecutions	Если задано значение true, включает выполнение в результат

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-py

## Пример ответа

В листинге 3.6 показан типичный ответ на данный вызов.

**Листинг 3.6.** Ответ на запрос о списке экспериментов

```
{
  "total": 17,
  "count": 17,
  "codes": [{
    "type": "Algorithm",
    "active": true,
    "versionId": 1,
    "idCode": "...",
    "name": "3Q GHZ State YXY-Measurement 1",
    "jsonQASM": {
      ...
    },
    "qasm": "",
    "codeType": "QASM2",
    "creationDate": "2018-04-14T19:09:51.382Z",
    "deleted": false,
    "orderDate": 1523733740504,
    "userDeleted": false,
    "displayUrls": {
      "png": "URL"
    },
    "isPublic": false,
    "id": "...",
    "userId": "..."
  }]
}
```

## Запуск эксперимента

В результате данного вызова эксперимент будет удаленно запущен на IBM Q Experience:

- метод HTTP: POST;
- URL: [https://quantumexperience.ng.bluemix.net/api/codes/execute?access\\_token=ACCESS-TOKEN&shots=SHOTS&deviceRunType=RUN-TYPE](https://quantumexperience.ng.bluemix.net/api/codes/execute?access_token=ACCESS-TOKEN&shots=SHOTS&deviceRunType=RUN-TYPE).

### Параметры запроса

Наименование	Значение
shots	Количество запусков на выполнение. Чем их больше, тем выше точность результатов. Обратите внимание на то, что уровень ваших кредитов будет уменьшаться на 3 за каждые 1024 запуска. В квантовом мире пространство ценится на вес золота
access_token	Токен доступа к вашему аккаунту
deviceRunType	Устройство, на котором запускается эксперимент. Это может быть: <ul style="list-style-type: none"><li>• для реальных процессоров — реальное наименование устройства, такое как <code>ibmqx2</code> и <code>ibmqx3</code>;</li><li>• для моделирующих устройств — моделирующее устройство или <code>sim_trivial_2</code></li></ul>
seed (необязательный параметр)	Необязательное случайное значение, которое требуется только для моделирующих устройств

### Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-py
Content-Type	Приложение/формат json

### Формат полезных данных

Телом ответа является документ в формате JSON, описывающий эксперимент, как показано в следующем отрывке:

```
{
  "name": "Experiment NAME",
  "codeType": "QASM2",
  "qasm": "CODE"
}
```



## Пример ответа

Это, пожалуй, самый важный вызов API. В качестве упражнения возьмем одно из состояний Белла из предыдущего раздела и запустим его как на моделирующем, так и на реальном устройстве с использованием REST API (листинг 3.7).

### Листинг 3.7. Измерение состояния Белла XW

```
IBMQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];

h q[0];
cx q[0],q[1];
h q[0];
s q[1];
h q[1];
t q[1];
h q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

В листинге 3.7 приведен код на ассемблере из эксперимента с одним из состояний Белла (XW), выполненного в веб-консоли в предыдущем разделе. Возьмите этот код и создайте полезные данные в формате JSON в виде {"name": "NAME", "codeType": "QASM2", "qasm": "ONE-LINE-QASM"}. Обратите внимание, что нужно передать название эксперимента и код на QASM должен быть отформатирован как одна строка, включая символ конца строки. Таким образом, полезные данные принимают окончательный вид:

```
{"name": "REST Bell State XW", "codeType": "QASM2", "qasm": "IBMQASM 2.0;\ninclude \"qelib1.inc\";\nqreg q[2];\ncreg c[2];\nh q[0];\ncx q[0],q[1];\nh q[0];\ns q[1];\nh q[1];\nt q[1];\nh q[1];\nmeasure q[0] -> c[0];\nmeasure q[1] -> c[1];"};
```

Теперь мы готовы отправить наш эксперимент через REST. Не забудьте, что вы должны сначала аутентифицироваться для получения токена доступа.



Убедитесь, что код ответа — 200 (ОК), и взгляните на вывод ответа. Удостоверьтесь, что эксперимент был записан в консоли Q Experience (рис. 3.9).

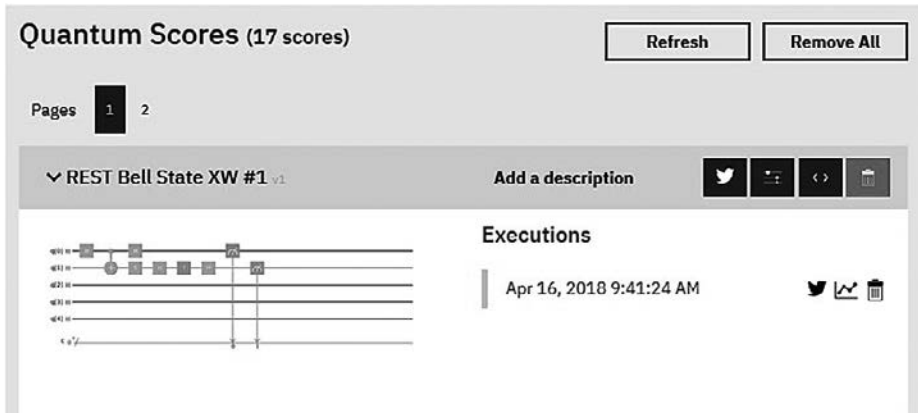


Рис. 3.9. Веб-консоль, показывающая эксперимент с состоянием Белла XW, отправленный через REST

## Отправка на реальное устройство

Для отправки на реальное устройство (в данном случае `ibmqx4`) измените параметры запроса на:

```
access_token=ACCESS_TOKEN &shots=1&deviceRunType=ibmqx4
```

### ПРИМЕЧАНИЕ

Реальные квантовые устройства могут быть отключены для технического обслуживания или по любой другой причине. Если это так, то отправка не удастся и вы получите ответ HTTP 400 (неверный запрос). Перед отправкой на реальное квантовое устройство убедитесь, что оно находится в Сети!

Если все пройдет хорошо, задание будет помещено в очередь выполнения и записано в веб-консоли. В листинге 3.8 показан результат отправки на реальное устройство со статусом `PENDING_IN_QUEUE`.

**Листинг 3.8.** Упрощенный HTTP-ответ из эксперимента с состоянием Белла XW, отправленного через REST

```
{
  "startDate": "2018-04-16T13:05:43.440Z",
  "modificationDate": 1523883943441,
  "typeCredits": "plan",
  "status": {
    "id": "WORKING_IN_PROGRESS"
  },
  "deviceRunType": "real",
  "ip": {
    "ip": "...",
    "city": "Raleigh",
    "country": "United States",
    "continent": "North America"
  },
  "shots": 1,
  "paramsCustomize": {},
  "deleted": false,
  "userDeleted": false,
  "id": "...",
  "codeId": "...",
  "userId": "...",
  "infoQueue": {
    "status": "PENDING_IN_QUEUE",
    "position": 21,
    "estimatedTimeInQueue": 735
  },
  "code": {
    "type": "Algorithm",
    "active": true,
    "versionId": 1,
    "idCode": "...",
    "name": "REST Bell State XW #1",
    "jsonQASM": {
      ...
      "numberGates": 7,
      "hasMeasures": true,
      "numberColumns": 11,
      "include": "include \"qelib1.inc\";"
    },
    "qasm": "...",
    "codeType": "QASM2",
    "creationDate": "2018-04-16T13:05:42.547Z",
    "deleted": false,
  },
}
```

```
    "orderDate": 1523883943351,  
    "userDeleted": false,  
    "isPublic": false,  
    "id": "...",  
    "userId": "..."  
  }  
}
```

На этом этапе вы успешно отправили свой первый эксперимент через REST. Попробуйте увеличить количество запусков вашего эксперимента, чтобы добиться большей точности.

## Запуск задания

Этот вызов очень похож на предыдущий запуск эксперимента, однако у него есть две конечные точки.

1. Для обычных пользователей IBM Q Experience.
2. Для корпоративных клиентов. Требуются идентификаторы хаба, группы и проекта.

У корпоративных клиентов есть как премиальный аккаунт, так и доступ к мощным 20-кубитным процессорам и, возможно, 50-кубитному чипу, который, по слухам, выйдет в конце 2018 года:

- метод HTTP: POST;
- URL 1 (5, 16 кубитов): [https://quantumexperience.ng.bluemix.net/api/Jobs?access\\_token=ACCESS-TOKEN](https://quantumexperience.ng.bluemix.net/api/Jobs?access_token=ACCESS-TOKEN);
- URL 2 (20+ корпоративных кубитов): [https://quantumexperience.ng.bluemix.net/api/Network/HUB/Groups/GROUP/Projects/PROJECT/jobs?access\\_token=ACCESS-TOKEN](https://quantumexperience.ng.bluemix.net/api/Network/HUB/Groups/GROUP/Projects/PROJECT/jobs?access_token=ACCESS-TOKEN).

## Параметры запроса

Наименование	Значение
access_token	Токен доступа к вашему аккаунту

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-ru
Content-Type	Приложение/формат json

## Формат полезных данных

Формат полезных данных включает все параметры выполнения: название серверного ПО, запуски и код в одном документе формата JSON, как показано в следующем отрывке:

```
{
  "backend": {
    "name": "simulator"
  },
  "shots": 1,
  "qasms": [{
    "qasm": "qams"
  }, ...]
}
```

---

### ПРИМЕЧАНИЕ

Эксперименты, отправленные через конечную точку Run Job, не записываются в раздел «Итоги» Composer, но помещаются в очередь выполнения для обработки.

---

В то же время отправка через конечную точку Run Experiment приведет к записи в Composer. Обратите внимание и на то, что для любого эксперимента, отправленного на моделирующее устройство, результаты возвращаются немедленно. Эксперименты, отправленные на реальное квантовое устройство, всегда будут входить в очередь выполнения в состоянии PENDING. По завершении пользователю будет отправлено уведомление по электронной почте. Давайте быстро отправим работу на реальное устройство ibmqx4. Вставьте в свой клиент REST следующую конечную точку: [https://quantumexperience.ng.bluemix.net/api/jobs?access\\_token=access\\_token](https://quantumexperience.ng.bluemix.net/api/jobs?access_token=access_token).

Установите метод HTTP POST, токен доступа и заголовки, как описано в предыдущем разделе. Используйте следующие полезные данные:

```
{
  "qasms": [{
    "qasm": "\n\ninclude \"qelib1.inc\";\nqreg q[5];\ncreg c[5];\n
    nu2(-4*pi/3,2*pi) q[0];\nu2(-3*pi/2,2*pi) q[0];\nu3(-pi,0,-pi)
    q[0];\nu3(-pi,0, -pi/2) q[0];\nu2(pi,-pi/2) q[0];\nu3(-pi,0,-pi/2)
    q[0];\nmeasure q -> c;\n" }],
    "shots": 1024,
    "backend": {
      "name": "ibmqx4"
    }
  },
  "maxCredits": 3
}
```

Эти полезные данные передают случайный эксперимент реальному устройству `ibmqx4`. Перед отправкой убедитесь, что оно в Сети (или используйте моделирующее устройство). Удостоверьтесь также, что код QASM, включая символ перевода строки (`\n`), расположен в одной строке. Обратите внимание на то, что двойные кавычки должны быть экранированы. Если отправка не удалась, это, вероятно, означает, что устройство находится в автономном режиме или QASM в ваших полезных данных неверный. Дважды и трижды проверьте, чтобы убедиться, что все правильно. Ответ, который я получил, говорит мне, что мое задание *выполняется*:

```
{
  "qasms": [
    {
      "qasm": "...",
      "status": "WORKING_IN_PROGRESS",
      "executionId": "5ba6955fd867ef0046615172"
    }
  ],
  "shots": 1024,
  "backend": {
    "id": "5ae875670f020500393162b3",
    "name": "ibmqx4"
  },
  "status": "RUNNING",
  "maxCredits": 3,
  "usedCredits": 3,
  "creationDate": "2018-09-22T19:17:51.448Z",
  "id": "5ba6955fd867ef0046615171",
  "userId": "5ae875060f0205003931559a",
}
```

```
    "infoQueue": {  
      "status": "PENDING_IN_QUEUE",  
      "position": 11  
    }  
  }
```

Обратите внимание, что мое задание не будет показано Composer, тем не менее я получу по электронной почте уведомление со ссылкой на результаты.

## Получение версии API

Возвращает версию REST API в Q Experience:

- метод HTTP: GET;
- URL: [https://quantumexperience.ng.bluemix.net/api/version?access\\_token=ACCESS-TOKEN](https://quantumexperience.ng.bluemix.net/api/version?access_token=ACCESS-TOKEN).

## Параметры запроса

Наименование	Значение
access_token	Токен доступа к вашему аккаунту

## Заголовки HTTP

Наименование	Значение
x-qx-client-application	Значения по умолчанию для qiskit-api-py

## Формат ответа

Возвращает строку с версией API (на момент написания этой книги — 6.4.8).

Заглянем внутрь написанного на Python REST API в IBMQuantumExperience, чтобы посмотреть, что происходит за кулисами. В качестве упражнения создадим собственный клиент для Node JS.



## Клиент Node JS для IBMQuantumExperience

В этом разделе описано простое упражнение, призванное симитировать один из компонентов Python SDK, известный как QISKit (Quantum Information Software Kit). В следующей главе мы более подробно рассмотрим Python SDK, а здесь укажем, что данный SDK создан на основе двух базовых библиотек.

- *IBMQuantumExperience*. Это реализация REST-клиента в Python, в которую я заглянул, чтобы представить REST API из предыдущего раздела. Данная модульная библиотека плохо документирована, что логично, поскольку она может измениться в будущем.
- *QISKit SDK*. Это главная точка входа во все ваши квантовые программы. Она обортывает логику вентилей, трансляцию текстов на ассемблере, симуляторы — локальный симулятор Python и быстрый симулятор C++ — и многое другое. Эта библиотека также вызывает IBMQuantumExperience для всех взаимодействий с платформой IBM Q Experience через REST.

Python — прекрасный язык, но центры обработки данных сейчас повально увлеклись Node JS. В этом разделе представлена простая реализация REST API для Node JS. Вот несколько причин, почему эта библиотека будет полезной.

- Node JS — рабочая лошадка для сетевых асинхронных вызовов ввода/вывода. Она быстрая, и это идеальная платформа для клиентов REST.
- Python — хороший язык с множеством великолепных числовых, математических, графических библиотек, но у него есть некоторые особенности. Например, важность отступов в Python (вы не можете одновременно использовать табуляцию и пробелы — здесь нет скобок для разграничения блоков). Поначалу это сводило меня с ума, ведь почти во всех компьютерных языках применяются фигурные скобки для деления логических блоков. В Python такого нет — вы должны использовать пробелы или табуляцию, причем не одновременно. Мне это не нравится. Я думаю, это неудачная реализация: если вы допустили

ошибку, то не знаете, где заканчивается логический блок. Вот для чего нужны скобки.

- Разнообразие всегда приветствуется: преодолев все проблемы с Python, я подумал, что эта библиотека может стать основой для клона QISKit для Node JS.

Приступим.

## Построение модуля Node для IBMQuantumExperience

Я пытался сохранить названия как можно более близкими к применяемым в версии Python. Чтобы создать модуль Node JS, сделайте папку с именем `IBMQuantumExperience` и инициализируйте Node JS, как показано далее:

```
$ mkdir IBMQuantumExperience
$ cd IBMQuantumExperience
$ npm init
```

---

### ПРИМЕЧАНИЕ

Пользователям Linux: я использовал Windows, чтобы написать этот раздел. Более того, я предполагаю, что вы установили Node JS, знакомы с модулями Node и умеете писать код на JavaScript. В общем, вы можете получить установщики Node для всех платформ по адресу <https://nodejs.org/en/>. Также обратите внимание на то, что Linux не нравятся названия пакетов прописными буквами. Избавьте себя от головной боли и воспользуйтесь Windows, чтобы проработать этот раздел.

---

Node предоставляет менеджер пакетов под названием `npm` (он в значительной степени совпадает с `pip` в Python), который можно использовать для инициализации вашего модуля. Третья команда создаст в текущей папке два файла:

- `index.js` — это код вашего модуля. Примечание для пользователей Linux: создайте этот файл самостоятельно;
- `Package.json` — это дескриптор модуля, содержащий такую информацию, как название, версия, автор, зависимости и многое другое.

Там же создайте папку `test` для юнит-тестов и установите мощный REST-клиент Node `request`:

```
$ mkdir test
$ npm install request
```

Вторая команда устанавливает популярный пакет HTTP-запросов и все его зависимости в текущую папку. Теперь мы готовы реализовать REST API из предыдущего раздела. Откройте `index.js` в своем любимом редакторе и начинайте.

## Экспорт методов API

Чтобы предоставить API через Node, используйте библиотеку `module.exports`, как показано в листинге 3.9. Обратите внимание, что это частичная реализация библиотеки и все фрагменты будут собраны в следующих разделах. Полная реализация доступна в папке `Workspace\Ch03\IBMQuantumExperience`. При этом вы должны суметь вставить все эти листинги в `index.js`. Опять же я предполагаю, что вы понимаете, как модули написаны в Node JS.

### Листинг 3.9. Предоставление открытых методов API через Node

```
const log = require('./log');
// Простая настраиваемая библиотека журналов (см. раздел отладки)
const request = require('request');

//...
module.exports = {
  init: function (cfg) {
    _config = cfg;
    var debug = _config.debug ? _config.debug : false;
    log.init (debug);
    return loginWithToken ();
  },

  getCalibration : calibration,
  getBackends : backends,
  getParameters : parameters,
  runExperiment : experiment

  // Оставлено в качестве упражнения getJobs : jobs,
  // Оставлено в качестве упражнения getMyCredits : credits
}
```

В листинге 3.9 происходит импорт внешней библиотеки с помощью ключевого слова `require`.

- Во второй строке импортируется клиентская библиотека HTTP-запросов для взаимодействия с Q Experience.
- В пятой строке декларируются предоставляемые данным модулем публичные методы:
  - `init` — осуществляет аутентификацию на платформе Q Experience, как описано в разделе «Удаленный доступ через REST API» на с. 129;
  - `getCalibration` — возвращает параметры калибровки платформы для заданного устройства;
  - `getBackends` — возвращает список доступных для использования квантовых (и моделирующих) устройств;
  - `getParameters` — возвращает параметры устройств в соответствии с описанием в разделе **Devices (Устройства)** веб-консоли Composer;
  - `runExperiment` — запускает эксперимент удаленно на моделирующем или реальном квантовом устройстве;
  - `getJobs` — возвращает список текущих заданий в очереди выполнения эксперимента;
  - `getMyCredits` — возвращает пользовательский баланс кредитов на выполнение и другую полезную информацию.

## Аутентификация с использованием токена

Перед аутентификацией библиотека инициализируется так же, как в Python, с конфигурационным объектом JSON, содержащим URL-адрес платформы, API-токен и многое другое. Например, так мы бы протестировали вызов REST API для получения информации о серверном ПО (взято из `test.js`):

```
// Запросите файл `index.js` из той же папки
const qx = require('index.js');
// Вставьте ваш токен API здесь
var config = { APIToken: 'YOUR_API_TOKEN' }
```

```

    , debug : true
    , 'url' : 'https://quantumexperience.ng.bluemix.net/api'
    , 'hub' : 'MY_HUB'
    , 'group' : 'MY_GROUP'
    , 'project' : 'MY_PROJECT'
  }
  // Получите информацию о серверном ПО
  async function testBackends() {
    await qx.init(config);
    var result = await qx.getBackends();
    console.log("---- BACKENDS ----\n" + JSON.stringify(result) +
      "\n-----" );
  }

```

Помните, что хаб, группа и проект являются параметрами только для корпоративных клиентов. Таким образом, они не используются в этой реализации, однако поддержку для них легко добавить. После инициализации просто отправьте запрос POST для входа в систему с помощью токена, как описано в REST API (листинг 3.10). Данный код, как и все листинги в этих разделах, находится в файле `index.js`.

### Листинг 3.10. Аутентификация с помощью токена через REST

```

function loginWithToken () {
  let options = {
    url: _config.url + '/users/loginWithToken',
    form: {'apiToken': _config.APIToken}
  };
  return new Promise(function(resolve, reject) {
    // Выполнить асинхронно задание
    // {"id":"Access tok","ttl":1209600,"created":"2018-04-
    // 17T23:30:21.089Z","userId":"userid"}
    request.post(options, function(err, res, body) {
      if (err) {
        reject(err);
      }
      else {
        var json = JSON.parse(body);
        _accessToken = json.id;
        _userId = json.userId;
        log.debug("Got User:" + _userId + " Tok:" + _accessToken);
        resolve(JSON.parse(body));
      }
    });
  });
}

```

В листинге 3.10:

- системный вызов `request.post` задействуется для отправки запроса HTTP POST в конечную точку `https://quantumexperience.ng.bluemix.net/api/users/loginWithToken` с использованием полезных данных в формате JSON `{'apiToken': 'YOUR_TOKEN'}`. В соответствии с описанием REST API в ответ на данный вызов будет возвращен документ в формате JSON: `{"id": "TOKEN", "ttl": 1209600, "created": "DATE", "userId": "USERID"}`. Этот документ анализируется, а токен доступа (`id`) и идентификатор пользователя (`userId`) сохраняются для последующего применения;
- обратите внимание: поскольку все сетевые операции ввода/вывода в Node асинхронные, все методы возвращают объект `Promise`. По сути, это асинхронная задача, инкапсулирующая необходимость ожидания выполнения задачи до считывания результатов. Таким образом, если вызов HTTP-запроса завершится успешно, будет выполнен обратный вызов `resolve` из `Promise` с данными HTTP-ответа, в противном случае — обратный вызов `reject` из `Promise`.

---

#### ПРИМЕЧАНИЕ

Объекты `Promise` являются убедительной альтернативой обратным вызовам для асинхронного кода. Тем не менее они могут иногда вводить в заблуждение. В общем, объекты `Promise` становятся стандартом де-факто для асинхронного программирования на JavaScript.

---

Если вы находите, что код обработки `Promise` сложен, существует более простой способ. Он будет показан в следующем разделе, где мы реализуем метод для извлечения списка серверного ПО.

## Перечисление серверного ПО

В листинге 3.11 показан запрос Node для получения серверного ПО из Q Experience.

- Посылается HTTP-запрос GET на `https://quantumexperience.ng.bluemix.net/api/Backends?access_token=TOKEN`.

- Возвращается объект `Promise`, который можно вызвать из любой асинхронной функции с помощью нового функционала `async/await` в JavaScript.

**Листинг 3.11.** Получение списка серверного ПО через Node

```
const _defaultHdrs = {
  'x-qx-client-application': _userAgent
};
function backends () {
  let options = {
    url: _config.url + '/Backends?access_token=' + _accessToken,
    headers: _defaultHdrs
  };
  return new Promise(function(resolve, reject) {
    // Выполнить асинхронно задание
    request.get(options, function(err, res, body) {
      if (err) {
        reject(err);
      }
      else {
        resolve(JSON.parse(body));
      }
    });
  });
}
```

Чтобы протестировать предыдущий метод, можно воспользоваться функционалом `async/await` из Node.js  $\geq 7.6$ , как показано в следующем фрагменте:

```
async function testBackends() {
  await qx.init(config);
  var result = await qx.getBackends();
  console.log("---- BACKENDS ----\n" + JSON.stringify(result) +
    "\n-----" );
}
```

---

### ПРИМЕЧАНИЕ

Асинхронная функция может содержать выражение `await`: тогда ее выполнение приостанавливается, программа ожидает сообщения об успешном выполнении от переданного объекта `Promise`, а затем возобновляет выполнение функции и возвращает значение результата.

---

## Перечисление параметров калибровки

В листинге 3.12 показано, как получить параметры калибровки и оборудования для конкретного серверного ПО в IBM Q Experience.

- Получить калибровочные данные, отправив запрос GET на [https://quantumexperience.ng.bluemix.net/api/Backends/NAME/calibration?access\\_token=TOKEN](https://quantumexperience.ng.bluemix.net/api/Backends/NAME/calibration?access_token=TOKEN), где NAME — это серверное ПО, о котором вы хотите сделать запрос, TOKEN — токен доступа, полученный на шаге аутентификации.
- Получить параметры серверного ПО, отправив аналогичный запрос GET на [https://quantumexperience.ng.bluemix.net/api/Backends/NAME/parameters?access\\_token=TOKEN](https://quantumexperience.ng.bluemix.net/api/Backends/NAME/parameters?access_token=TOKEN).

Формат ответа на оба запроса описан в разделе «Удаленный доступ через REST API» данной главы.

**Листинг 3.12.** Получение данных о калибровке и параметрах устройства

```
function calibration (name) {
  let options = {
    url: _config.url + '/Backends/' +
      name + '/calibration?access_token=' + _accessToken,
    headers: _defaultHdrs
  };
  return new Promise(function(resolve, reject) {
    request.get(options, function(err, res, body) {
      if (err) {
        reject(err);
      }
      else {
        resolve(JSON.parse(body));
      }
    });
  });
}

function parameters (name) {
  let options = {
    url: _config.url + '/Backends/' +
      name + '/parameters?access_token=' + _accessToken,
    headers: _defaultHdrs
  };
  return new Promise(function(resolve, reject) {
```



```

    request.get(options, function(err, res, body) {
      if (err) {
        reject(err);
      }
      else {
        resolve(JSON.parse(body));
      }
    });
  });
}

```

Для тестирования кода создайте асинхронную функцию и используйте ключевое слово `await`, чтобы получить ответ от асинхронной задачи, как показано в следующем фрагменте:

```

async function testCalibration() {
  await qx.init(config);
  var result1 = await qx.getCalibration('ibmqx4');
  var result2 = await qx.getParameters('ibmqx4');
  console.log(JSON.stringify(result1) );
  console.log(JSON.stringify(result1) );
}

```

В качестве последнего шага рассмотрим выполнение эксперимента.

## Старт эксперимента

Это наиболее важный вызов API, и после его выполнения эксперимент должен быть записан в раздел результатов вашей веб-консоли IBM Q Experience (листинг 3.13).

Чтобы отправить эксперимент программным путем, выполните HTTP-запрос POST в конечную точку `/codes/execute` с полезными данными в формате JSON:

```
{'name': name, "codeType": "QASM2", "qasm": "YOUR_QASM_CODE"}
```

- Помните, что код на ассемблере должен быть отформатирован как одна строка с символами конца строки (`\n`) для разделения инструкций. Например, в следующем коде объявляются пять кубитов и пять классических регистров: `"\n\ ninclude \"qelib1.inc\";\nqreg q[5];\n ncreg c[5];\n"`.

- Параметр `name` определяет название эксперимента, которое будет записано в веб-консоли.
- Параметр `shots` — это количество запусков, выполняемых квантовым процессором.
- Параметр `device` может быть названием моделирующего (для удаленного моделирующего устройства) или реального квантового устройства, таким как `ibmqx4`.

---

### СОВЕТ

Если вы запустите эксперимент на реальном устройстве, он будет помещен в очередь выполнения для дальнейшей обработки. По завершении вы получите письмо. А если запустите эксперимент на удаленном моделирующем устройстве, результаты будут возвращаться синхронно.

---

### Листинг 3.13. Запуск эксперимента

```
const _userAgent = 'qiskit-api-py'; // Глобальная переменная

function experiment (name, qasm, shots, device) {
  let options = {
    url: _config.url + '/codes/execute?access_token=' + _accessToken
      + '&shots=' + shots + '&deviceRunType=' + device,
    headers: {'Content-Type': 'application/json',
      'x-qx-client-application': _userAgent} ,
    form: {'name': name, "codeType": "QASM2", "qasm": qasm}
  };
  return new Promise(function(resolve, reject) {
    request.post(options, function(err, res, body) {
      if (err) {
        reject(err);
      }
      else {
        resolve(JSON.parse(body));
      }
    });
  });
}
```

Вставьте код из листинга 3.13 в файл `index.js`, используйте следующий фрагмент кода для запуска эксперимента на реальном квантовом устройстве `ibmqx4`, затем убедитесь, что эксперимент записан в веб-консоли, и дождитесь уведомления по электронной почте:

```
async function testExperiment () {
  await qx.init(config);
  var name = "REST Experiment from Node JS #1"
  var qasm = "\n\ninclude \"qelib1.inc\";\nqreg q[5];\ncreg c[5];\n
  nu2(-4*pi/3,2*pi) q[0];\nu2(-3*pi/2,2*pi) q[0];\nu3(-pi,0,-pi)
  q[0];\nu3(-pi,0,-pi/2) q[0];\nu2(pi,-pi/2) q[0];\nu3(-pi,0,-pi/2)
  q[0];\nmeasure q -> c;\n";
  var shots = 1;
  var device = "ibmqx4";
  var result = await qx.runExperiment(name, qasm, shots, device);
  console.log("---- EXPERIMENT " + name + " ----\n" +
  JSON.stringify(result) + "\n-----" )
}
```

---

#### ПРИМЕЧАНИЕ

Код для модуля Node для IBMQuantumExperience находится в разделе `Workspace\Ch03\IBMQuantumExperience` исходного кода книги. Тестовый скрипт проекта размещен в файле `test/tests.js`. Отредактируйте этот файл, добавьте свой токен API и выполните его из IBMQuantumExperience с помощью команды `node test/tests.js`.

---

## Отладка и тестирование

Для простой отладки я создал подмодуль `log.js` (на том же уровне, что и `index.js`) и использовал основной объект консоли для отображения информации, как показано в следующем фрагменте:

```
var _debug = false;

function LOGD( tag, txt ) {
  if ( _debug ) {
    console.log('[DBG-QX] ' + tag + ' ' + (txt ? txt : ""));
  }
}
```

```
function LOGE( tag, txt ) {
    console.error('[ERR-QX] ' + tag + ' ' + (txt ? txt : ''));
}
function init (debug) {
    _debug = debug;
}

exports.init = init;
exports.debug = LOGD;
exports.error = LOGE;
```

Основной модуль (`index.js`) использует этот подмодуль для отображения отладочных сообщений в консоли. Наконец, чтобы протестировать пакет, отредактируйте `test/tests.js` и вставьте фрагменты теста, описанные в этих разделах, как показано в следующей части листинга из `tests.js`:

```
// test/tests.js запрашивает файл `index.js` из той же папки.
const qx = require('../');

// Вставьте ваш токен API сюда
var config = { APIToken: 'API-TOKEN'
  , debug: true
  , 'url': 'https://quantumexperience.ng.bluemix.net/api'
  , 'hub': 'MY_HUB'
  , 'group': 'MY_GROUP'
  , 'project': 'MY_PROJECT'
};

async function testBackends() {
    await qx.init(config);
    var result = await qx.getBackends();
    console.log("---- BACKENDS ----\n" + JSON.stringify(result) +
      "\n-----" );
}

async function testJobs () {
    await qx.init(config);
    var filter = '{"limit":2}';
    var jobs = await qx.getJobs(filter);
    console.log ("---- JOBS----\n" + JSON.stringify(jobs) + "\n-----");
}

// Вставьте все фрагменты тестов сюда...
// ....

try {
    testBackends();
    testJobs ();
    // другие тесты..
```

```
}  
catch (e){  
    console.error(e);  
}
```

Чтобы выполнить тест, запустите `node test\tests.js` в папке `IBMQuantumExperience`. Обратите внимание: я пропустил методы `getJobs` и `getMyCredits`, они остаются для вас в качестве упражнения. У вас достаточно знаний, чтобы легко реализовать и протестировать их.

## Поделитесь с миром — опубликуйте свой модуль

Если хотите поделиться своей работой со всем миром, можете опубликовать свой модуль в реестре `npm`. Для этого вы должны создать учетную запись пользователя на [www.npmjs.com/](http://www.npmjs.com/) или выполнить это вручную, используя команды:

```
npm adduser  
npm publish
```

Убедитесь, что вы документировали свой код, добавив документ на языке разметки `markdown` (`readme.md`) в корневую папку. После публикации перейдите по адресу <https://npmjs.com/package/<package>> и проверьте, действует ли ваш модуль. Теперь другие смогут установить его командой:

```
npm install IBMQuantumExperience
```

Разработчики Node JS могут отправлять эксперименты в Q Experience с помощью такого кода:

```
const qx = require('IBMQuantumExperience');  
...  
async function sendExperiment () {  
    var config = { APItoken: 'API-TOKEN'  
    , 'url': 'https://quantumexperience.ng.bluemix.net/api',  
    'debug': false};  
    await qx.init(config);  
    var name = "REST Experiment from Node JS #1"  
    var qasm = "MY_QASM";  
    var device = "ibmqx4";  
    var result = await qx.runExperiment(name, qasm, 1024, device);  
}
```

В этой главе вы сделали первый шаг в карьере квантового программиста. IBM создала удивительную облачную платформу для изучения этих невероятных машин. Мы должны поблагодарить добрых людей из IBM за то, что они сделали эту платформу доступной для широкого использования. Сегодня квантовые компьютеры — это экспериментальные машины, поэтому не ждите, что сможете приобрести их в местном магазине компьютерной техники. Тем не менее вскоре они будут массово использоваться в центрах обработки данных, поэтому сейчас самое время научиться квантовому программированию.

# 4

## **QISKit — отличный SDK для квантового программирования на Python**

В этой главе вы познакомитесь с QISKit — лучшим SDK для квантового программирования. Вы узнаете, как просто установить SDK в своей локальной системе. Но прежде, чем писать первую квантовую программу, всегда полезно понять, что такое квантовые вычисления и чем они отличаются от классических. Для этой цели приведено очень простое объяснение состояний кубитов и квантовых вентилей с использованием линейной алгебры. В этой главе также показано, как квантовые вычисления могут в точности повторить классический аналог и, кроме того, ускорить получение результатов. Далее в главе рассматривается строение квантовой программы, включая системные вызовы, форматы составления схем, квантовый ассемблер и многое другое.

QISKit содержит набор полезных моделирующих устройств для локального или удаленного выполнения ваших программ, он также позволяет работать в режиме реального времени. Шаг за шагом вы узнаете, как запускать квантовые программы на реальном устройстве, предоставленном потрясающей облачной платформой IBM Q Experience. Итак, перейдите на свой Рабочий стол, и приступим.

## Установка QISKit

QISKit — это Quantum Information Software Kit, SDK для квантового программирования в облаке. Он написан на Python — мощном сценарном языке для решения научных задач. Мой опыт в основном связан с бизнесом, и я мало писал на Python в последнее время, поэтому давайте рассмотрим установку SDK как в Linux CentOS 6 или 7, так и в Windows 64. Начнем с самого простого (Windows), а затем перейдем к самому сложному (CentOS).

### Настройка в Windows

QISKit требует Python 3.5 или более поздней версии. Если у вас Windows, то, скорее всего, Python не установлен. Если это так, вы можете получить установщики с веб-сайта [Python.org](http://Python.org). Загрузите установщик, запустите его и проверьте, произошла ли установка, выполнив в командном окне следующую команду:

```
C:\>Python -V
Python 2.7.6
```

У меня старый добрый Python 2.7, а у вас может быть установлено несколько версий Python одновременно. Я скачал встраиваемый ZIP-файл и развернул его в C:\Python36-64, так что в моем случае:

```
C:\>C:\Python36-64\Python.exe -V
Python 3.6.4
```

В Python есть замечательный менеджер пакетов `pip` (preferred installer program), который сильно упрощает установку модулей. Таким образом, чтобы установить QISKit, просто наберите в консоли:

```
C:\>pip install qiskit
```

Вывод на экран должен выглядеть аналогично коду из листинга 4.1. Убедитесь, что не было сообщений об ошибках.

#### Листинг 4.1. Установка QISKit в Windows 64 бит

```
Collecting qiskit
  Using cached qiskit-0.4.11.tar.gz
```



```
Collecting IBMQuantumExperience>=1.8.28 (from qiskit)
  Using cached IBMQuantumExperience-1.9.0-py3-none-any.whl
Collecting matplotlib<2.2,>=2.1 (from qiskit)
  Using cached matplotlib-2.1.2.tar.gz
Collecting networkx<2.1,>=2.0 (from qiskit)
  Downloading networkx-2.0.zip (1.5MB)
    100% |████████████████████| 1.6MB 400kB/s

145
Collecting numpy<1.15,>=1.13 (from qiskit)
  Downloading numpy-1.14.2-cp36-cp36m-manylinux1_i686.whl (8.7MB)
    100% |████████████████████| 8.7MB 105kB/s
...
Running setup.py install for pycparser ... done
Running setup.py install for matplotlib ... done
Running setup.py install for networkx ... done
Running setup.py install for ply ... done
Running setup.py install for mpmath ... done
Running setup.py install for sympy ... done
Running setup.py install for qiskit ... done
Successfully installed IBMQuantumExperience-1.9.0 qiskit-0.4.11
requests-2.18.4 ...
```

Вот и все. Вы сделали первый шаг в путешествии в качестве квантового программиста. Для пользователей Linux настроим все в CentOS 6 или 7.

## Настройка в Linux CentOS

В CentOS 6 или 7 настройка немного сложнее. Это связано с тем, что CentOS фокусируется в основном на стабильности, а не на ультрасовременном программном обеспечении. Таким образом, CentOS поставляется со встроенным Python 2.7. Более того, официальный дистрибутив не предоставляет пакетов для Python 3.5. Однако это не означает, что Python 3.5 нельзя установить. Посмотрим, как это делается.

---

### ПРИМЕЧАНИЕ

Инструкции, приведенные в этом разделе, должны работать для любых вариантов Linux, основанных на базе Red Hat, таких как RHEL 6–7, CentOS 6–7 и Fedora Core.

---

## Шаг 1. Подготовка системы

Прежде всего убедитесь, что `yum` (менеджер обновлений Linux) актуален, выполнив команду:

```
$ sudo yum -y update
```

Затем установите `yum-utils` — коллекцию утилит и плагинов, расширяющих и дополняющих `yum`:

```
$ sudo yum -y install yum-utils
```

Установите инструменты разработки CentOS. К ним относятся компиляторы и библиотеки, позволяющие создавать и компилировать многие типы программного обеспечения:

```
$ sudo yum -y groupinstall development
```

Теперь установим Python 3. Обратите внимание, что мы будем запускать несколько версий Python: официальную, 2.7 и 3.6 для разработки.

## Шаг 2. Установка Python 3

Чтобы выйти за пределы ограничений поставляемого по умолчанию дистрибутива CentOS, можно воспользоваться проектом, созданным на общественных началах, под названием `Inline with Upstream Stable (IUS)`. Это набор новейших библиотек для тех ОС, которые их не предоставляют, таких как CentOS. Давайте установим в ней IUS через `yum`:

```
$ sudo yum -y install https://centos7.iuscommunity.org/ius-release.rpm  
(CentOS7)  
$ sudo yum -y install https://centos6.iuscommunity.org/ius-release.rpm  
(CentOS6)
```

После завершения установки IUS мы можем установить последнюю версию Python (3.6):

```
$ sudo yum -y install python36u
```

Проверьте, чтобы убедиться, что установка выполнена корректно:

```
$ python3.6 -V  
Python 3.6.4
```

Теперь установим и проверим `pip`:

```
$ sudo yum -y install python36u-pip
$ pip3.6 -V
```

Наконец, нам нужно установить пакет `python36u-devel` из IUS, который предоставляет полезные библиотеки для разработки:

```
$ sudo yum -y install python36u-devel
```

### Шаг 3. Создание виртуальной среды

Этот шаг может быть полезен только в том случае, если у вас многопользовательская система с несколькими версиями Python и вы не хотите мешать другим пользователям. Например, чтобы создать виртуальную среду в своей домашней папке, выполните:

```
$ mkdir $HOME/qiskit
$ cd $HOME/qiskit
$ python3.6 -m venv qiskit
```

С помощью этой последовательности команд в домашней папке пользователя создается папка с названием `qiskit`, где будут храниться все ваши квантовые программы. В ней создается также виртуальная среда Python 3.6, названная `qiskit`. Чтобы активировать среду, выполните команду:

```
$ source qiskit/bin/activate
(qiskit) [centos@localhost qiskit]$
```

В виртуальной среде вы можете использовать команду `python` вместо `python3.6` и `pip` вместо `pip3.6`, если вам так нравится:

```
$ python -V
Python 3.6.4
```

---

#### СОВЕТ

Если вы не активируете виртуальную среду, то используйте `python3.6` и `pip3.6` вместо `python` и `pip`.

---

## Шаг 4. Установка QISKit

Активируйте свою виртуальную среду и установите QISKit с помощью команды:

```
$ pip install qiskit
```

В листинге 4.2 показан стандартный вывод для предыдущей команды.

### Листинг 4.2. Установка QISKit в CentOS 6

```
Collecting qiskit
  Downloading qiskit-0.5.7.tar.gz (4.5MB)
    100% |████████████████████| 4.5MB 183kB/s
Collecting IBMQuantumExperience>=1.8.28 (from qiskit)
  Downloading IBMQuantumExperience-1.9.0-py3-none-any.whl
Collecting matplotlib<2.2,>=2.1 (from qiskit)
  Downloading matplotlib-2.1.2.tar.gz (36.2MB)
    100% |████████████████████| 36.2MB 18kB/s
Complete output from command python setup.py egg_info:
=====
Edit setup.cfg to change the build options
BUILDING MATPLOTLIB
  matplotlib: yes [2.1.2]
  python: yes [3.6.4 (default, Dec 19 2017, 14:48:15) [GCC
    4.4.7 20120313 (Red Hat 4.4.7-18)]]
  platform: yes [linux]
...
Installing collected packages: IBMQuantumExperience, numpy, python-
dateutil, pytz, cyclar, pyarsing, matplotlib, decorator, networkx, ply,
scipy, mpmath, sympy, pillow, qiskit
  Running setup.py install for pycparser ... done
  Running setup.py install for matplotlib ... done
  Running setup.py install for networkx ... done
  Running setup.py install for ply ... done
  Running setup.py install for mpmath ... done
  Running setup.py install for sympy ... done
  Running setup.py install for qiskit ... done
Successfully installed IBMQuantumExperience-1.9.0 qiskit-0.4.11
requests-2.18.4 requests-ntlm-1.1.0 scipy-1.0.1 six-1.11.0 sympy-1.1.1
urllib3-1.22

(qiskit) [centos@localhost qiskit]$
```

---

### СОВЕТ

В виртуальной среде пакеты Python будут устанавливаться в домашнюю папку среды `lib/python3.6/site-packages` вместо системного пути, как показано на рис. 4.1.

---



Рис. 4.1. Расположение папки виртуальной среды Python virtual

Теперь мы готовы начать писать квантовый код. Посмотрим, как это делается.

## Кубит 101: базовая алгебра

Прежде чем приступить к написанию квантовых программ, нужно немного освежить знания в области фундаментальной математики, чтобы понимать, что происходит за кулисами. В главе 2 вы узнали, как кубит представляется сферой Блоха (см. рис. 2.7): это геометрическое представление чистого состояния двухуровневой квантово-механической системы (кубита). Но, возможно, лучший способ понять основную модель кубита и влияние квантовых вентилей — воспользоваться его алгебраическим представлением. Для этого вам нужно вспомнить некоторые базовые понятия линейной алгебры, включая следующие.

*Линейные векторы.* Простые векторы вида  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ , которые будут использоваться для представления базовых состояний кубита.

**Комплексные числа.** Комплексным называется число, состоящее из вещественной и мнимой частей, представляемое как  $a + bi$ , где  $i = \sqrt{-1}$ . Обратите внимание на то, что комплексные числа не могут существовать в нашей физической реальности. Коэффициенты  $\alpha$ ,  $\beta$  состояния суперпозиции кубита  $\psi = \alpha|0\rangle + \beta|1\rangle$  являются комплексными числами.

- **Комплексное сопряжение.** Термин, который вы часто будете слышать в связи с квантовыми вентилями. Чтобы получить комплексное сопряжение, просто поменяйте на противоположный знак при мнимой части:  $a + bi$  становится  $a - bi$  и наоборот.
- **Произведение матриц.** Если  $A$  является матрицей размерности  $n \times t$ , а  $B$  — матрицей размерности  $t \times p$ , то их произведение  $AB$  — это матрица размерности  $n \times p$ , где  $t$  элементов строки матрицы  $A$  умножаются на  $t$  элементов столбца матрицы  $B$ , а произведения суммируются для получения элемента матрицы  $AB$ . Возьмите первую строку первой матрицы и умножьте каждый ее элемент на каждый элемент первого столбца второй матрицы, затем суммируйте произведения, чтобы получить первый элемент матрицы результатов (рис. 4.2). Не стоит паниковать, большинство матриц, которые мы рассмотрим, имеют размерность  $2 \times 2$ , а их элементами являются 0 и 1.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 4 & 8 \\ 5 & 10 \\ 6 & 12 \end{bmatrix} = \begin{bmatrix} 32 & \\ & \end{bmatrix}$$

Рис. 4.2. Базовая операция матричного произведения

## Алгебраическое представление квантового бита

В классической модели основной единицей информации является бит, который представлен 0 или 1. На физическом уровне бит преобразуется в поток напряжения через транзистор. В квантовых вычислениях основной единицей является квантовый бит (кубит), который физически переводится в манипуляции с фотонами, электронами или атомами. Алгебраически кубит представлен кет-вектором.

**ПРИМЕЧАНИЕ**

Обозначение «кет-вектор» было введено в 1939 году физиком Полем Дираком и известно также как нотация Дирака. Кет-вектор обычно представлен в виде вектора-столбца и обозначается  $|\varphi\rangle$ .

**Скобочная нотация Дирака**

В нотации Дирака базовые квантовые состояния кубита представлены векторами  $|0\rangle$  и  $|1\rangle$ . Они называются вычислительными базисными состояниями.

**ПРИМЕЧАНИЕ**

Квантовое состояние кубита представляет собой вектор в двумерном комплексном векторном пространстве. Проиллюстрируем это на примере простого графа.

На рис. 4.3 показано комплексное векторное пространство, используемое для представления состояния кубита. Слева изображено так называемое базисное состояние из двух единичных векторов для состояний  $|0\rangle$  и  $|1\rangle$  в нотации Дирака. Справа показано общее квантовое состояние, представляющее собой линейную комбинацию двух этих состояний. Таким образом, базисные состояния и общие квантовые состояния могут быть записаны как векторы:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix};$$
$$\alpha|0\rangle + \beta|1\rangle,$$

где  $\alpha$  и  $\beta$  — амплитуды единичного вектора. Обратите внимание, что амплитуда единичного вектора должна быть равна 1, в связи с чем  $\alpha$  и  $\beta$  должны подчиняться ограничению  $|\alpha|^2 + |\beta|^2 = 1$ . Это алгебраическое представление является ключом к пониманию воздействия логического вентиля на кубит, как вы увидите позже.

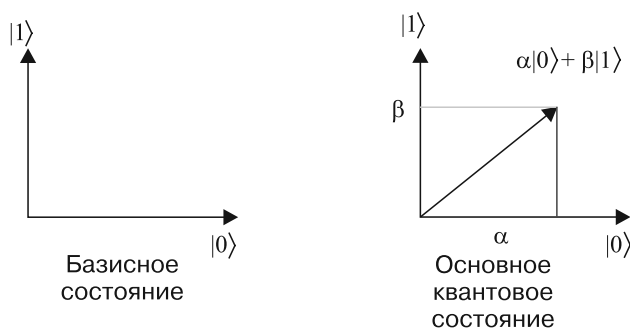


Рис. 4.3. Квантовые состояния кубита

Так почему же состояние кубита представлено как вектор в более сложном представлении, чем его классический аналог? Зачем вообще использовать векторы? Причина в следующем: это позволяет построить лучшую модель вычислений, что будет видно, когда мы станем рассматривать квантовые вентили и суперпозицию состояний. В общем, квантовая механика — это теория, развивавшаяся на протяжении многих десятилетий, и, в конце концов, вектор — очень простой математический объект, который легко понять и которым просто манипулировать. Возможно, это лучший рабочий инструмент.

## Суперпозиция — это просто красивое слово

Суперпозиция определяется физиками как свойство элементарных частиц находиться в нескольких состояниях одновременно. Если вам трудно понять эту концепцию, на помощь придет линейная алгебра.

---

### ПРИМЕЧАНИЕ

Суперпозиция — это просто линейная комбинация состояний  $|0\rangle$  и  $|1\rangle$ , то есть  $\alpha|0\rangle + \beta|1\rangle$ , где длина вектора состояния равна 1, как показано на рис. 4.3.

---



## Скобочная нотация кажется слишком странной? Используйте векторы

Если вам нравится алгебра и вы считаете, что скобочная нотация сбивает вас с толку, просто возьмите вместо нее знакомое векторное представление. Таким образом, суперпозиция из предыдущего раздела может быть записана как:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

Обратите внимание, что кет-векторы подчиняются тем же правилам, что и векторы. Например, для них справедливо умножение на скалярную величину:

$$2(\alpha|0\rangle + \beta|1\rangle) = 2 \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 2\alpha \\ 2\beta \end{bmatrix}.$$

## Изменение состояния кубита с помощью квантовых вентилей

Цель квантовых вентилей — управление состоянием кубита для достижения желаемого результата. Они являются основными строительными блоками квантовых вычислений так же, как классические логические элементы в классическом мире. Некоторые квантовые вентили являются эквивалентом классических аналогов. Рассмотрим их.

### Вентиль НЕ (Паули X)

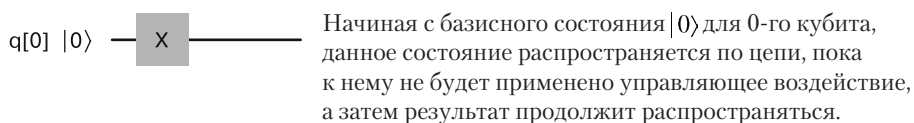
Самый простой вентиль, воздействующий на один кубит. Это квантовый эквивалент классического элемента НЕ, как и его аналог, он изменяет состояние кубита на противоположное, то есть:

$$|0\rangle \rightarrow |1\rangle, |1\rangle \rightarrow |0\rangle.$$

На суперпозицию вентиль  $X$  действует линейно, то есть он изменяет соответствующее состояние. Таким образом,  $|0\rangle$  становится  $|1\rangle$ , а  $|1\rangle$  становится  $|0\rangle$ :

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|1\rangle + \beta|0\rangle.$$

На квантовых схемах вентиль НЕ изображается как  $X$ , он известен также как вентиль Паули  $X$  (назван в честь австрийского физика Вольфганга Паули, одного из основателей квантовой механики).



Существует другой способ увидеть вентиль  $X$  в действии — с помощью его матричного представления. Можно в точности увидеть, как состояние меняется на противоположное, используя матрицу Паули:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Состояние кубита меняется на противоположное с помощью матричного представления вентилля  $X$  и векторов  $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  и  $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ :

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0+0 \\ 1+0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle;$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0+1 \\ 0+0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle.$$

Существует еще более простая квантовая схема, самая простая из всех — квантовая проволока, обозначаемая греческой буквой «пси»  $|\psi\rangle$  — — — — —  $|\psi\rangle$ , которая описывает вычислительное состояние во времени. Оно может показаться элементарным, но физически реализовать его сложнее всего. Из-за атомарного масштаба квантовая проволока (представьте себе фотоны, электроны или отдельные атомы) очень хрупка и подвержена ошибкам, обусловленным окружающей средой.

Другим интересным свойством вентиля  $X$  является то, что два элемента НЕ в строке дают единичную матрицу ( $I$ ) — очень важный инструмент в линейных преобразованиях. Давайте выполним расчеты.

$$q[0] \quad |0\rangle \quad \text{---} \quad \boxed{X} \quad \text{---} \quad \boxed{X} \quad \text{---}$$

$$|\psi\rangle \rightarrow XX|\psi\rangle$$

Чтобы понять, каково влияние схемы, посмотрим, что произойдет, когда мы перемножим две матрицы  $X$ :

$$XX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0+1 & 0+0 \\ 0+0 & 1+0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I.$$

Вентиль  $X$  — это простейший пример квантового логического элемента, схемы и вычислений.

В следующем разделе рассмотрим истинно квантовый вентиль Адамара и то, как он может переключать суперпозиции с помощью схем и алгебры.

## Истинно квантовые вычисления: управление суперпозициями с помощью вентиля Адамара

Воздействие вентиля Адамара на базисные состояния формально определяется как:

$$|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}}; \quad |1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Кроме того, суперпозиции состояний  $\alpha|0\rangle + \beta|1\rangle$  вентиль Адамара сопоставляет:

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) + \beta \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \frac{\alpha + \beta}{\sqrt{2}} |0\rangle + \frac{\alpha - \beta}{\sqrt{2}} |1\rangle.$$

В схеме и матричном представлении фильтр Адамара воздействует на одиночный кубит.

$$q[0] \quad |0\rangle \quad \text{---} \quad \boxed{H} \quad \text{---}$$

Применяя вентиль  $H$  к базисным состояниям  $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  и  $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$  получим:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \left( \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{|0\rangle + |1\rangle}{\sqrt{2}};$$

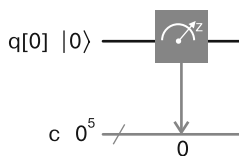
$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \left( \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Так исходя из каких вычислительных соображений используется вентиль Адамара? Что он дает нам?

Если не вдаваться в технические детали, ответ заключается в том, что вентиль Адамара расширяет диапазон состояний, возможных для квантовой схемы. Это важно, потому что расширение состояний позволяет находить быстрые способы решения и, следовательно, ускорять вычисления. Можно провести аналогию с игрой в шахматы. Например, если бы вашему слону было позволено двигаться, как ферзь и слону одновременно (расширение состояний), это дало бы вам преимущество в игре и позволило быстрее поставить мат сопернику. Увеличение мощности квантовой машины — вот что дает вентиль Адамара.

### Измерение квантового состояния сложнее, чем кажется

Представьте, что в подвале вашего дома есть лаборатория. Вам даны кубит в состоянии  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  и измерительный прибор, и вас просят вычислить коэффициенты  $\alpha$  и  $\beta$ . То есть вычислить квантовое состояние. Задача может показаться простой, однако это невозможно сделать. Принципы квантовой механики утверждают, что квантовое состояние системы непосредственно не наблюдается. Лучшее, что можно сделать, — угадать приблизительную информацию об  $\alpha$  и  $\beta$ . Этот процесс называется измерением в вычислительном базисе.



Результатом измерения квантового состояния  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  являются классические биты:

$\alpha|0\rangle + \beta|1\rangle \rightarrow 0$  с вероятностью  $|\alpha|^2$ ;

$\alpha|0\rangle + \beta|1\rangle \rightarrow 1$  с вероятностью  $|\beta|^2$ .

Таким образом, процесс измерения определяет вероятности для классических битов 0 и 1 равными абсолютным значениям коэффициентов  $\alpha$  и  $\beta$ , возведенным в квадрат. Представить на физическом уровне, как протекает этот процесс, можно наблюдением физического фотона, атома или электрона с помощью измерительного устройства. Поэтому при измерениях часто используются квантовые вентили.

Измерение нарушает состояние квантовой системы, давая в результате классические биты. Важно помнить, что по завершении этого процесса коэффициенты  $\alpha$  и  $\beta$  разрушаются. Это означает, что мы не можем хранить большие объемы информации в кубите. Представьте: если бы мы могли измерить точные значения  $\alpha$  и  $\beta$ , то с помощью комплексных чисел теоретически можно было бы хранить бесконечное количество классической информации в состоянии кубита. Вычисляя точные значения  $\alpha$  и  $\beta$ , мы могли бы извлечь всю эту информацию. Однако это невозможно. С точки зрения квантовой механики это недопустимо.

Последнее замечание по поводу измерений касается нормализации квантового состояния: при измерении в вычислительном базисе  $\alpha|0\rangle + \beta|1\rangle$  сумма вероятностей для классических битов 0 и 1 должна быть равна 1. То есть:

$$\text{Вероятность}(0) + \text{Вероятность}(1) = |\alpha|^2 + |\beta|^2 = 1.$$

Значит, длина вектора квантового состояния должна быть равна 1 (нормирована). Это происходит из-за того, что вероятности для измерения в сумме дают 1. В следующем разделе мы поговорим о том, как обобщаются однокубитные логические вентили, что они собой представляют и как используются для построения более сложных схем.

## Обобщенные однокубитные вентили

До сих пор мы видели два простых вентилей  $X$  и  $H$ , представленных матрицами:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Вспомните также, что суперпозиция квантового состояния выражается как вектор  $|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ . Тогда применение обоих вентилей к квантовому состоянию может быть представлено в обобщенном виде для любой единичной матрицы:

$$H \begin{bmatrix} \alpha \\ \beta \end{bmatrix}; \quad X \begin{bmatrix} \alpha \\ \beta \end{bmatrix}; \quad U \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \text{ где } U = H, X.$$

$U$  называется обобщенным однобитовым вентилем с тем ограничением, что  $U$  должен быть унарным.

### ПРИМЕЧАНИЕ

Матрица  $U$  является унитарной, если результатом ее умножения на эрмитово сопряженную  $U^\dagger$  является единичная матрица:  $U^\dagger U = I$ . Эрмитово сопряжение ([https://en.wikipedia.org/wiki/Hermitian\\_conjugate](https://en.wikipedia.org/wiki/Hermitian_conjugate)) обозначается символом  $\dagger$  (<https://en.wikipedia.org/wiki/Dagger>):  $U^\dagger = (U^T)^*$ , то есть комплексное сопряжение транспонированной матрицы.

Транспонированная матрица — это новая матрица, строки которой являются столбцами первоначальной. Например, если  $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ , то  $A^T = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ .

Затем для получения эрмитово сопряжения  $A^\dagger = \begin{bmatrix} a & c \\ b & d \end{bmatrix}^*$  возьмите комплексное сопряжение каждого элемента. (Комплексным сопряжением  $a + bi$ , где  $a$  и  $b$  вещественные, является  $a - bi$ , то есть меняется на противоположный знак при мнимой части, если таковая имеется.)

Обратите внимание, что оба вентиля,  $H$  и  $X$ , должны быть унарными. Это очень просто проверить, вычислив  $X^\dagger X = I$  и  $H^\dagger H = I$ :

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} X^\dagger = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \rightarrow X^\dagger X = XX = I;$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} H^\dagger = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \rightarrow H^\dagger H = HH = I.$$

## Унитарные матрицы хороши для квантовых вентилях

Вопрос, который возникает после прочтения предыдущего раздела: зачем нужны все эти трудности? Почему вентили  $X$  и  $H$  должны быть унарными? Ответ заключается в том, что унитарные матрицы сохраняют длину вектора. Это полезно для квантовых вентилях, потому что они требуют нормализации входных и выходных состояний (имеют длину вектора 1).

На самом деле унитарные матрицы — это единственный тип матриц, сохраняющих длину, и, следовательно, единственный тип матриц, который можно использовать для квантовых вентилях. В общем, возникают более глубокие вопросы: почему квантовые вентили должны быть в первую очередь линейными и зачем вообще применять матричное представление? Мы постараемся ответить на них в следующем разделе, а пока нужно принять это как данность.

## Другие однокубитные вентили

В предыдущем разделе мы рассмотрели однокубитные вентили X и H. В то же время существуют и другие однобитные вентили, которые полезны в квантовых вычислениях.

У вентиля X есть два партнера: Y и Z. Они образуют трио, известное как вентили Паули ( $\sigma$ ):

$$q[0] \quad |0\rangle \quad \text{---} \boxed{X} \text{---} \boxed{Y} \text{---} \boxed{Z} \text{---}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 1 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Вентиль вращения:

$$q[0] \quad |0\rangle \quad \text{---} \boxed{T} \text{---}$$

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Эти три матрицы подходят для задач обработки информации, таких как сверхплотное кодирование (SDC) — процесс, который ориентирован на эффективное хранение классической информации в кубите. Они также используются при анализе свойств атома, таких как спин электрона. Вдобавок тесно связаны с тремя измерениями пространства XYZ.

Представляет собой уже знакомый нам поворот на угол  $\theta$  в реальном пространстве. Это унитарная матрица, и в данном случае вентиль T выполняет вращение на  $\pi/4$  вокруг оси Z. Этот вентиль требуется для универсального контроля.

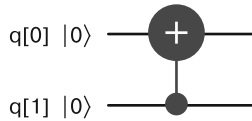
Вентили также могут управлять многими кубитами, как мы увидим в следующем разделе.

## Запутывание кубитов с помощью управляемого вентиля HE

Это последний вентиль в арсенале, необходимом для квантовых вычислений. Управляемый вентиль HE (CNOT) представляет собой двухкубитный вентиль с четырьмя вычислительными состояниями.

Для суперпозиции вентиль CNOT с четырьмя базисными состояниями дает  $\alpha|00\rangle + \beta|01\rangle + \delta|10\rangle + \gamma|11\rangle$ , где  $\alpha, \beta, \delta$  и  $\gamma$  — коэффициенты суперпозиции.

Квантовая схема:



Матричное представление оператора CNOT для базисных состояний:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$|00\rangle$

$|01\rangle$

$|10\rangle$

$|11\rangle$

Символ «плюс» (+) называется управляемым кубитом, а серая точка (под ним) — это управляющий кубит. Вентиль CNOT выполняет простые действия:

- если для управляющего кубита установлено состояние 1, то состояние управляемого кубита меняется на противоположное;
- в противном случае ничего не происходит.

Точнее, если первый бит является управляющим, то:

- $|00\rangle \rightarrow |00\rangle$  значение управляющего кубита 0 — ничего не выполнять;
- $|01\rangle \rightarrow |01\rangle$  значение управляющего кубита 0 — ничего не выполнять;
- $|10\rangle \rightarrow |11\rangle$  значение управляющего кубита 1 — изменить состояние второго на противоположное;
- $|11\rangle \rightarrow |10\rangle$  значение управляющего кубита 1 — изменить состояние второго на противоположное.

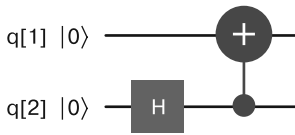
Простое представление данных состояний:  $|xy\rangle \rightarrow |xy \oplus x\rangle$ .

### ПРИМЕЧАНИЕ

Вентиль CNOT необходим для запутывания, что критично для всех видов задач, включая квантовую телепортацию, сверхплотное кодирование и практически любой квантовый алгоритм.

Например, для того, чтобы запутать два кубита, нужно применить вентиль Адамара (H) к первому кубиту и затем вентиль CNOT — ко второму, как показано далее.





Для базисного состояния кубита (2) вентиль Адамара дает  $|00\rangle \rightarrow \frac{|00\rangle + |10\rangle}{\sqrt{2}}$ .

После применения вентиль CNOT состояние второго кубита изменяется на противоположное, если состояние управляющего соответствует 1, то есть  $|00\rangle \rightarrow \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ .

Это создает запутанное состояние между кубитами 1 и 2.

В общем, вентиль CNOT и однокубитные вентили являются мощным арсеналом для квантовых вычислений. Поскольку они позволяют строить унитарные операции на любом количестве кубитов, их называют универсальными для квантовых вычислений. Это означает, что для создания квантового компьютера, который может решить любую квантовую задачу, достаточно использовать однокубитные вентили совместно с вентилем CNOT и вентилями измерения.

## Универсальные квантовые вычисления позволяют получить решение быстрее, чем классические

Вы можете задаться вопросом: как все эти схемы и формулы из предыдущего раздела помогут в решении вычислительных задач, которые легче и, скорее всего, дешевле решить в классической системе. Если принять во внимание так называемую битовую мощность классической системы:

$$x \rightarrow f(x),$$

где при заданном входном значении  $x$  цель состоит в том, чтобы вычислить функцию  $f(x)$  по крайней мере за  $2^{k-1}$  элементарных операции (где  $k$  — это битовая мощность), то универсальные квантовые вычисления могут обеспечить эквивалентную схему примерно того же размера, которая содержит ту же классическую модель:

$$|x\rangle, 0 \rightarrow |x\rangle, f(x).$$

В этой схеме удивительно то, что иногда существуют быстрые пути решения, обеспечивающие большую мощность квантовых вычислений и более быстрое получение результатов. Это означает, что вы можете вычислить  $f(x)$  менее чем за  $2^{k-1}$  операции. Для некоторых квантовых алгоритмов, таких как факторизация, ускорение является экспоненциальным! Это истинная сила квантовых систем. Итак, теперь, когда вы изучили базовую математическую модель квантовых схем, пришло время переключиться в режим программирования и посмотреть, как все это можно превратить в настоящую компьютерную программу, которая будет выполняться на реальном квантовом устройстве.

## Ваша первая квантовая программа

Рассмотрим структуру квантовой программы на простейшем примере. В нем мы создадим один кубит, один классический регистр для его измерения, затем применим вентиль Паули X (битовое инвертирование) к кубиту и, наконец, измерим его значение. Основной псевдокод программы можно воспроизвести следующим образом.

1. Создать квантовую программу.
2. Создать один или несколько кубитов и классических регистров для их измерения.
3. Создать схему, которая объединит кубиты в устройство выполнения логических операций.
4. Применить квантовые вентили к кубитам для достижения желаемого результата.
5. Измерить кубиты в классических регистрах, чтобы получить конечный результат.
6. Скомпилировать программу. На данном шаге создается JSON-представление программы в специфическом формате, который будет описан позже в этом разделе.
7. Запустить на моделирующем или реальном квантовом устройстве.
8. Получить результаты.

Теперь подробно рассмотрим как код на Python, так и схему в Composer.

### Листинг 4.3. Строение квантовой программы

```
#####
import sys
import qiskit
import logging
from qiskit import QuantumProgram

# Main sub
def main():

    # Создать программу
    qp = QuantumProgram()

    # Создать один кубит
    quantum_r = qp.create_quantum_register("qr", 1)

    # Создать один классический регистр
    classical_r = qp.create_classical_register("cr", 1)

    # Создать схему
    qp.create_circuit("Circuit", [quantum_r], [classical_r])

    # Получить схему по названию
    circuit = qp.get_circuit('Circuit')

    # Включить ведение журналов
    qp.enable_logs(logging.DEBUG);

    # Применить вентиль Паули X к первому кубиту в квантовом регистре "qr"
    circuit.x(quantum_r[0])

    # Измерительный вентиль, приводящий нулевой кубит к классическому биту 0
    circuit.measure(quantum_r[0], classical_r[0])

    # Серверное моделирующее устройство
    backend = 'local_qasm_simulator'

    # Набор исполняемых схем
    circuits = ['Circuit']

    # Компиляция вашей программы
    qobj = qp.compile(circuits, backend)

    # Запуск на моделирующем устройстве
    result = qp.run(qobj, timeout=240)

    # Вывод итоговых результатов
    print (str(result.get_counts('Circuit')))
```

```
#####
# Linux :main()
# windows if __name__ == '__main__':
#     main()
```

Рассмотрим, что происходит в листинге 4.3.

- В строках 2–5 импортируются нужные библиотеки: `sys` (системная), `qiskit` (квантовые классы), `logging` (для отладки) и `QuantumProgram` — базового класса для всех программ.
- Затем в строке 11 создается `QuantumProgram`. Это точка доступа ко всем операциям.
- Для создания списка кубитов воспользуйтесь системным вызовом квантовой программы `create_quantum_register(NAME, SIZE)`, где `NAME` — это название списка регистров, а `SIZE` — количество кубитов. В данном случае оно равно 1 (строка 14).
- Для каждого кубита создайте классический регистр для проведения измерений с помощью системного вызова `create_classical_register(NAME, SIZE)`.
- Затем создайте схему, добавив системный вызов `create_circuit(NAME, QUANTUM_SET, CLASSIC_SET)`, где `NAME` — название схемы, `QUANTUM_SET` — список кубитов, а `CLASSIC_SET` — список классических регистров. Схема — это логический блок, содержащий все кубиты и классические регистры (строка 20).
- При желании включите отладку с помощью системного вызова `enable_logs(LEVEL)`, где `LEVEL` может иметь одно из значений `logging.DEBUG`, `logging.INFO` и т. д. (обычные журналы).
- Затем пропустите кубиты через квантовые вентили и выполните их измерения, чтобы получить результаты. В данном случае мы применяем вентиль Паули X, который изменяет изначальное состояние кубита с  $|0\rangle$  на  $|1\rangle$  (строки 25–29).
- Наконец, скомпилируйте программу и запустите моделирующее или реальное устройство. В данном случае выполняем запуск на локальном моделирующем устройстве на Python (`local_qasm_simulator`) (строки 37–41).

Разработчики под Windows, будьте внимательны! Вам нужно обернуть свою программу в функцию `main` и затем вызвать ее:

```
if __name__ == '__main__':  
    main()
```

В Windows так придется сделать, потому что QISKit выполняет программу с помощью асинхронных задач (исполнителей), а при запуске такой задачи подпроцесс сначала выполняет основной модуль. Таким образом, вам нужно защитить основной код, чтобы избежать рекурсивного создания подпроцессов. Я выяснил это на собственном горьком опыте, когда мои программы правильно работали в CentOS, но в Windows выдавали ошибку:

RuntimeError:

An attempt has been made to start a new process before the current process has finished its bootstrapping phase.

This probably means that you are not using `fork` to start your child processes and you have forgotten to use the proper idiom in the `main` module:

```
if __name__ == '__main__':  
    freeze_support()  
    ...
```

The "`freeze_support()`" line can be omitted if the program is not going to be frozen to produce an executable.

Это может создать затруднения для новичков в разработке на Python. Теперь запустите программу, чтобы увидеть вывод:

```
INFO:qiskit._jobprocessor:<qiskit._result.Result object at  
0x00000000D99F470>  
{'1': 1024}
```

Результатом является документ в формате JSON `{'1': 1024}`, где 1 соответствует результату измерения кубита (помните, что мы использовали вентиль X для инверсии бита), а 1024 — количество итераций с таким результатом. Вероятность получения данного результата рассчитывается как отношение количества итераций с ним (1024) к общему количеству итераций программы (1024). В данном случае  $P = 1024 / 1024 = 1$ .

**ПРИМЕЧАНИЕ**

Квантовые компьютеры — это вероятностные машины. Всем измерениям сопутствует вероятность получения определенного результата.

Листинг 4.3 может также быть описан эквивалентной квантовой схемой, быстро составленной в Composer в IBM Q Experience (рис. 4.4).

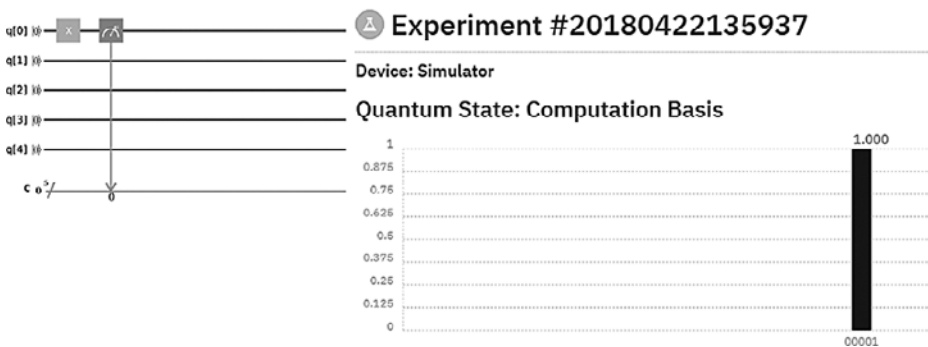


Рис. 4.4. Эксперимент в Composer для листинга 4.3

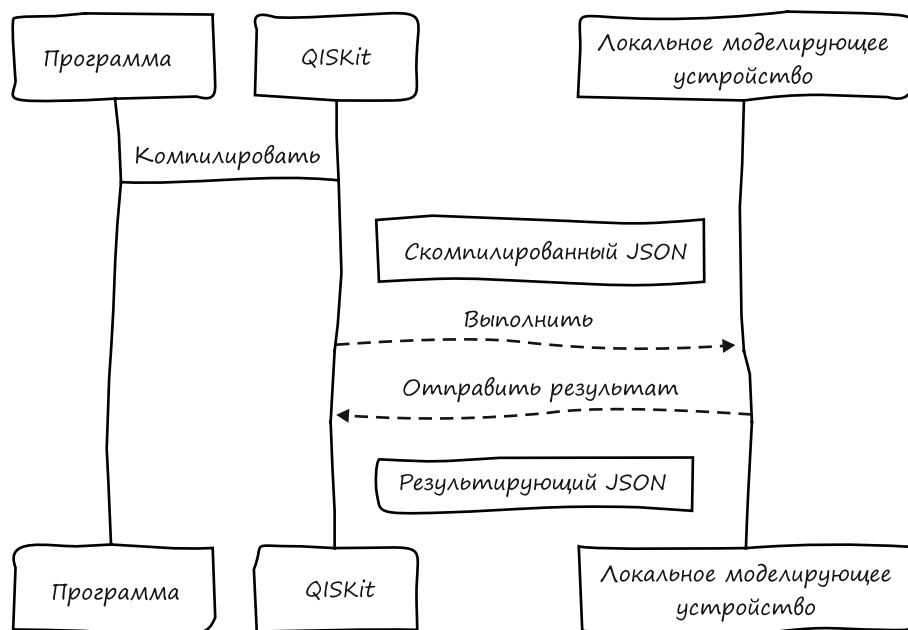
Здесь показаны квантовая схема для листинга 4.3, результат эксперимента и сопутствующая ему вероятность. Как видите, схема очень проста: в Composer нужно перетащить вентиль X в положение над нулевым кубитом, а затем выполнить измерение на этом же кубите. Composer покажется вам замечательным инструментом для построения относительно простых схем, их выполнения и визуализации результатов! Теперь давайте заглянем во внутренние компоненты SDK, чтобы увидеть, как обрабатывается этот код.

## Внутренние компоненты SDK: компиляция схемы и QASM

На рис. 4.5 показаны внутренние процессы, происходящие при запуске вашей программы.

- QISKit компилирует схему (схемы) программы в документ в формате JSON, который будет отправлен на локальное моделирующее устройство.

- Управляющее устройство анализирует этот документ, запускает схему и возвращает документ в формате JSON без четкой структуры (скрытый от разработчика).
- QISKit обортывает результирующий JSON-документ в объект, доступный для основной программы. Например, вызов `result.get_counts('Circuit')` извлекает численные данные из этого документа.



**Рис. 4.5.** Схема последовательности действий между программой, QISKit и локальным моделирующим устройством

## Компиляция схемы

В листинге 4.4 показан формат скомпилированной программы перед отправкой на моделирующее устройство. В документе приводятся:

- идентификатор выполнения;
- заголовок с информацией о моделирующем устройстве, включая название, количество кредитов, использованных при выполнении, количество запусков;

О раздел схем, содержащий массив объектов. Каждая схема содержит:

- название;
- заголовок (конфигурацию) с такой информацией, как карта связей кубитов, базовые (физические) вентили, начальное число времени выполнения и т. д.;
- раздел скомпилированной схемы с заголовком, содержащим информацию о кубитах и классических регистрах, массивом операторов (или вентилей), примененных к схеме, и их параметрами.

**Листинг 4.4.** Формат скомпилированной программы из листинга 4.3

```
{
  "id": "aA46vJHgkKQko3u5L1QqbUDk31sY2m",
  "config": {
    "max_credits": 10,
    "backend": "local_qasm_simulator",
    "shots": 1024
  },
  "circuits": [{
    "name": "Circuit",
    "config": {
      "coupling_map": "None",
      "layout": "None",
      "basis_gates": "u1,u2,u3,cx,id",
      "seed": "None"
    },
    "compiled_circuit": {
      "operations": [{
        "name": "u3",
        "params": [3.141592653589793, 0.0, 3.141592653589793],
        "texparams": ["\\pi", "0", "\\pi"],
        "qubits": [0]
      }, {
        "name": "measure",
        "qubits": [0],
        "clbits": [0]
      }
    ],
    "header": {
      "number_of_qubits": 1,
      "qubit_labels": [
        ["qr", 0]
      ],
    },
  }
}
```



```
        "number_of_clbits": 1,  
        "clbit_labels": [  
            ["cr", 1]  
        ]  
    }  
},  
"compiled_circuit_qasm": "OPENQASM 2.0;\ninclude \"qelib1.inc\";  
\nqreg qr[1];\ncreg cr[1];\n\nu3(3.14159265358979,0,3.14159265358979)  
qr[0];\nmeasure qr[0] -> cr[0];\n"
```

Для того чтобы отобразить схему, скомпилированную в вашей программе, распечатайте результат шага компиляции с помощью следующей команды:

```
qobj = qp.compile(circuits, backend)  
print(str(qobj))
```

---

### ПРИМЕЧАНИЕ

Формат компиляции непрозрачен для программиста и предназначен не для прямого доступа, а через SDK API. Причина в том, что его формат может меняться от версии к версии. Однако всегда хорошо понимать, что происходит за кулисами.

---

## Результаты выполнения

Это ответный документ от локального моделирующего устройства к QISKit. Формат документа показан в листинге 4.5.

Наиболее значимая информация, содержащаяся в нем:

- статус запуска, время выполнения, название моделирующего устройства и т. д.;
- результирующие данные. Их можно получить внутри вашей программы с помощью вызова `print (str(result.get_counts('Circuit')))`.

**Листинг 4.5.** Документ с результатами от моделирующего устройства

```
{
  "backend": "local_qiskit_simulator",
  "id": "aA46vJHgnKQko3u5L1QqbUDk31sY2m",
  "result": [{
    "data": {
      "counts": {
        "1": 1024
      },
      "time_taken": 0.0780002
    },
    "name": "Circuit",
    "seed": 123,
    "shots": 1024,
    "status": "DONE",
    "success": true,
    "threads_shot": 4
  ]},
  "simulator": "qubit",
  "status": "COMPLETED",
  "success": true,
  "time_taken": 0.0780002
}
```

Получить документ с результатами немного сложнее, потому что это объект без четкой структуры, недоступный для пользовательской программы. Тем не менее вы можете сохранить скомпилированную схему из предыдущего раздела и вручную передать ее в моделирующее устройство, чтобы получить результат, показанный в листинге 4.5. Однако это задание оставлено для самостоятельного выполнения. Важно помнить, что документ с результатами, как и формат компиляции, непрозрачен для программиста. Причина в том, что их форматы могут меняться со временем. Тем не менее всегда полезно понимать, что происходит за кадром.

---

**ПРИМЕЧАНИЕ**

Разработчикам моделирующих устройств будет полезно разбираться в форматах компиляции и результатов. Например, вы можете сохранить форматы компиляции и результатов для примера схемы, исправить ошибку в моделирующем устройстве на C++, передать схему и сравнить результаты. Таким образом, ваше моделирующее устройство можно легко интегрировать с SDK, чтобы остальные пользователи могли его опробовать.

---

## Код ассемблера

Скомпилированная схема из листинга 4.4 включает раздел, который содержит трансляцию программы на квантовый ассемблер (QASM), как показано в следующем отрывке:

```
OPENQASM 2.0;
include "qelib1.inc";
qreg qr[1];
creg cr[1];
x qr[0];
measure qr[0] -> cr[0];
```

---

### ПРИМЕЧАНИЕ

QASM полезен только тогда, когда запуск происходит на удаленном моделирующем устройстве, предоставляемом IBM Q Experience.

---

## Локальные моделирующие устройства QISKit

Доступ к реальным квантовым устройствам в IBM Q Experience ограничен кредитами, количество которых уменьшается по мере использования. Таким образом, не стоит запускать совсем простые программы, такие как в листинге 4.3. Для этой цели QISKit предоставляет множество моделирующих устройств, способных удовлетворить все потребности в тестировании. В табл. 4.1 приведен список локальных и удаленных моделирующих устройств, доступных через QISKit и IBM Q Experience на момент написания этой книги.

**Таблица 4.1.** Список локальных и удаленных моделирующих устройств для IBM Q Experience

Название	Описание
local_qasm_simulator	Моделирующее устройство на Python, поставляемое по умолчанию в комплекте с QISKit. Надо отметить, что оно очень медленное
local_clifford_simulator, также известный как local_qiskit_simulator	Высокопроизводительное моделирующее устройство, написанное на C++, с реалистичной эмуляцией шума и ошибок

Продолжение ➞

**Таблица 4.1** (продолжение)

Название	Описание
ibmqx_qasm_simulator	Высокопроизводительное 24-кубитное удаленное моделирующее устройство на QASM, предоставляемое Q Experience. Это удаленное моделирующее устройство по умолчанию
ibmqx_hqc_qasm_simulator	Тридцатидвухкубитное сверхмощное параллельное моделирующее устройство, предоставляемое Q Experience. Это резервная замена для удаленного моделирующего устройства по умолчанию

В качестве простого упражнения получите список моделирующих и реальных устройств от IBM Q Experience, вставив в браузер следующий URL для REST API: [https://quantumexperience.ng.bluemix.net/api/Backends?access\\_token=ACCESS\\_TOKEN](https://quantumexperience.ng.bluemix.net/api/Backends?access_token=ACCESS_TOKEN).

Конечно, вам нужен токен доступа, который можно легко получить с помощью API удаленного доступа, рассмотренного в главе 3. Далее запустим нашу программу на других локальных моделирующих устройствах, включая удаленное моделирующее устройство от IBM Q Experience. Наконец, посмотрим, какое моделирующее устройство самое быстрое.

### **Запуск на локальном моделирующем устройстве, написанном на C++**

QISKit по умолчанию использует моделирующее устройство на чистом Python (`local_qasm_simulator`). Однако вы можете воспользоваться и быстродействующим моделирующим устройством на C++ с реалистичным уровнем шума и ошибок, изменив название серверного ПО в своей программе на `local_clifford_simulator` или `local_qiskit_simulator` (строка 35 в листинге 4.3). Но прежде, чем задействовать его, следует учесть некоторые предостережения.

- *Пользователям Linux.* Данное моделирующее устройство работает на основе стандарта C++11, требующего gcc 5.3 или более поздней версии. Фактически это моделирующее устройство так и не было скомпилировано в моих системах CentOS 6 и 7 (вы же можете взять Windows).

- *Пользователям Windows.* Python использует утилиту CMake для компиляции моделирующего устройства на лету. В общем, источник по умолчанию не предоставляет решение Visual Studio для сборки в Windows. Тем не менее я подготовил одно решение и исправил пару ошибок, с которыми столкнулся в Windows 7.

---

### СОВЕТ

Бинарный файл для 64-разрядной Windows для моделирующего устройства на C++ можно найти в исходниках книги по адресу `Ch04\qiskit-simulator\qiskit-simulator\x64\Debug`. Программа Visual Studio 2017 также предоставляется, если вы хотите скомпилировать его самостоятельно. Убедитесь, что вы скопировали все файлы из этой папки в `HOME\Lib\site-packages\qiskit\backends`, если их там нет.

---

## Запуск на удаленном моделирующем устройстве

Для запуска на удаленном моделирующем устройстве, предоставляемом IBM Q Experience, нужно немного изменить листинг 4.3. Давайте посмотрим как.

Первое, что нам потребуется, — дескриптор конфигурации IBM Q Experience с параметрами выполнения, приведенными в следующем отрывке:

```
APItoken = 'YOU-API-TOKEN'
config = {
    'url': 'https://quantumexperience.ng.bluemix.net/api',
    # Следующий код понадобится только пользователям IBM Q
    'hub': 'MY_HUB',
    'group': 'MY_GROUP',
    'project': 'MY_PROJECT'
}
```

---

### СОВЕТ

Предыдущий код должен храниться в отдельном файле (`Qconfig.py`) в той же папке, что и основная программа. Получите свой токен API из веб-консоли IBM Q Experience, как показано в главе 3, и вставьте его в код. Обратите внимание, что хаб, группу и проект указывают только корпоративные клиенты.

---

Затем импортируйте упомянутый ранее дескриптор в основную программу:

```
# Конфигурация Q Experience
import Qconfig

# Main sub
def main():
```

Наконец, замените исполняющее серверное ПО на удаленное моделирующее устройство.

1. Измените название серверного ПО на `ibmq_qasm_simulator`.
2. Сообщите квантовой программе, что ей нужно использовать IBM Q Experience, установив параметры API с помощью системного вызова `qr.set_api(Qconfig.APIToken, Qconfig.config['url'])`, где `APIToken` и `URL` — значения, взятые из дескриптора конфигурации.
3. Выполните в IBM Q Experience посредством системного вызова `result = qr.execute(circuits, backend, shots=512, max_credits=3)`. Обратите внимание на то, что мы не компилируем и не запускаем схему, как раньше. Поэтому вы должны удалить вызовы `qobj = qr.compile(circuit, backend)` и `result = qr.run(qobj, wait = 2, timeout = 240)`.

Изменения показаны в следующей команде. Убедитесь, что вы удалили результаты предыдущих компиляций и запустили системные вызовы, иначе программа не будет выполнена:

```
backend = 'ibmqx_qasm_simulator'

# Группа исполняемых схем
circuits = ['Circuit']
# установите APIToken и url для API Q Experience
qr.set_api(Qconfig.APIToken, Qconfig.config['url'])

result = qr.execute(circuits, backend, shots=512, max_credits=3, wait=10,
timeout=240)
```

Наконец, запустите на выполнение и протестируйте. Должен получиться вывод, похожий на следующий:

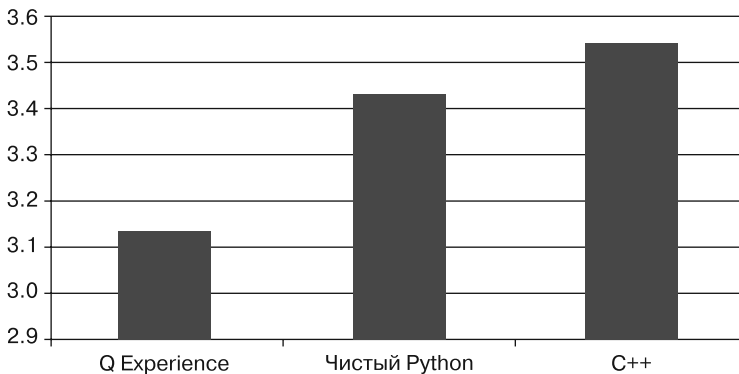
```
DEBUG:qiskit.backends._qeremote:Running on remote backend ibmq_qasm_
simulator
with job id: 3677ff592e5e5a6fd31a569b0b4faf92
```

```
INFO:qiskit._jobprocessor:<qiskit._result.Result object at  
0x0000000004A35160>  
{'1': 512}
```

Теперь соберем все вместе и посмотрим, какое моделирующее устройство самое быстрое. Я делаю ставку на C++.

### Наиболее быстро моделирующее устройство по результатам сравнения времени выполнения

Я собрал статистику о времени выполнения для всех моделирующих устройств на машине с 64-битным процессором под управлением Windows 7. Невероятно, но самым быстрым оказалось удаленное моделирующее устройство IBM Q Experience, за которым следовали чистый Python и мой личный фаворит C++ (рис. 4.6).



**Рис. 4.6.** Время выполнения для моделирующих устройств QISKit

Несмотря на то что вызов проходит через сеть, удаленному моделирующему устройству IBM Q Experience удастся превзойти другие. Меня поразило то, что интерпретируемое моделирующее устройство на Python может быть быстрее реализации машинного кода. Вероятно, это связано с тем, что вызов машинного кода использует асинхронные задачи для порождения процесса моделирующего устройства на C++ и таким образом приводит к такому замедлению, что код на Python превосходит его. Теперь, когда вы узнали, как запустить программу на моделирующем устройстве, сделаем это на реальном.

## Запуск на реальном квантовом устройстве

Изменим программу из предыдущего раздела, чтобы усложнить схему. Листинг 4.6 показывает пример схемы, которая выполняет серию вращений на первом кубите квантового компьютера. Вращения демонстрируют использование физических затворов реального квантового процессора `ibmqx4` `u1`, `u2` и `u3` для поворота кубита вокруг осей  $X$ ,  $Y$  и  $Z$  сферы Блоха на  $\theta$ ,  $\phi$  или  $\lambda$  градусов.

### ПРИМЕЧАНИЕ

Сфера Блоха — это геометрическое представление кубита, где верхняя часть оси  $Z$  отвечает базисному состоянию  $|0\rangle$ , а нижняя —  $|1\rangle$ . Поворот вокруг данной оси представляет собой вероятность того, что кубит коллапсирует в определенном направлении при выполнении измерения (рис. 4.7).

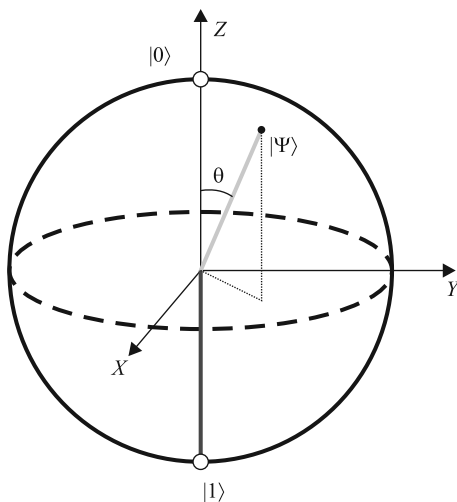


Рис. 4.7. Представление кубита на сфере Блоха

Физические вентили, известные также как базовые вентили, важны, потому что на их основе строятся более сложные логические элементы. Таким образом, в листинге 4.6 выполняются следующие шаги.



1. Выделяются пять кубитов и пять классических измерительных регистров, соответствующих пяти кубитам, доступным в процессоре `ibmqx4` в Q Experience (строки 17–20).
2. Затем на первом кубите выполняется последовательность вращений с использованием базисных элементов `u1`, `u2` и `u3` (строки 29–34).
3. Наконец, на кубите выполняется измерение, а результат сохраняется в классическом регистре.
4. Перед выполнением в качестве серверного ПО устанавливается `ibmqx4` (пятикубитный процессор — строка 42), а токен аутентификации и URL-адрес API задаются через `set_api(Qconfig.APIToken, Qconfig.config['url'])`.
5. Для выполнения на реальном квантовом устройстве используйте системный вызов `execute QuantumProgram(NAMES, BACKEND, shots = SHOTS, max_credits = CREDITS, timeout = TIMEOUT)`, где:
  - `NAMES` — список названий схем;
  - `SHOTS` — количество итераций, выполненных в схеме. Чем оно больше, тем выше точность;
  - `CREDITS` — максимальное количество баллов, которое вы хотите потратить из своего банка кредитов на выполнение (15 — заданное по умолчанию начальное количество). Обратите внимание, что чем больше снимков вы сделаете, тем больше кредитов будет вычтено из банка. Следите за этим, чтобы не исчерпать кредиты;
  - `TIMEOUT` — тайм-аут чтения из удаленной конечной точки.

---

### ПРИМЕЧАНИЕ

Квантовые программы или эксперименты на Python, выполненные на реальном устройстве, не записываются в разделе `Composer-Scores` IBM Q Experience. Это связано с тем, что Python неявно использует `Jobs` из REST API, который вместо этого помещает эксперимент в очередь выполнения. Если хотите записать то, что выполнили, в `Composer`, можете использовать веб-консоль или REST API, как показано в следующем разделе.

---

**Листинг 4.6.** Пример схемы № 2

```
import sys,time,math
import qiskit
import logging
from qiskit import QuantumProgram

# Конфигурация Q Experience
import Qconfig

# Main sub
def main():

    # Создать программу
    qp = QuantumProgram()

    # Создать один кубит
    quantum_r = qp.create_quantum_register("qr", 5)

    # Создать один классический регистр
    classical_r = qp.create_classical_register("cr", 5)

    # Создать схему
    circuit = qp.create_circuit("Circuit", [quantum_r], [classical_r])

    # Включить ведение журналов
    qp.enable_logs(logging.DEBUG);

    # Первый физический вентиль:  $u1(\lambda)$  к кубиту 0
    circuit.u2(-4 * math.pi/3, 2 * math.pi, quantum_r[0])
    circuit.u2(-3 * math.pi/2, 2 * math.pi, quantum_r[0])
    circuit.u3(-math.pi, 0, -math.pi, quantum_r[0])
    circuit.u3(-math.pi, 0, -math.pi/2, quantum_r[0])
    circuit.u2(math.pi, -math.pi/2, quantum_r[0])
    circuit.u3(-math.pi, 0, -math.pi/2, quantum_r[0])

    # Измерительный вентиль, приводящий кубит 0 к классическому биту 0
    circuit.measure(quantum_r[0], classical_r[0])
    circuit.measure(quantum_r[1], classical_r[1])
    circuit.measure(quantum_r[2], classical_r[2])

    # Серверное ПО
    backend = 'ibmqx4'

    # Набор исполняемых схем
    circuits = ['Circuit']

    # Установить APIToken и URL-адрес API Q Experience
    qp.set_api(Qconfig.APIToken, Qconfig.config['url'])
```

```

result = qp.execute(circuits, backend, shots=512, max_credits=3,
timeout=240)

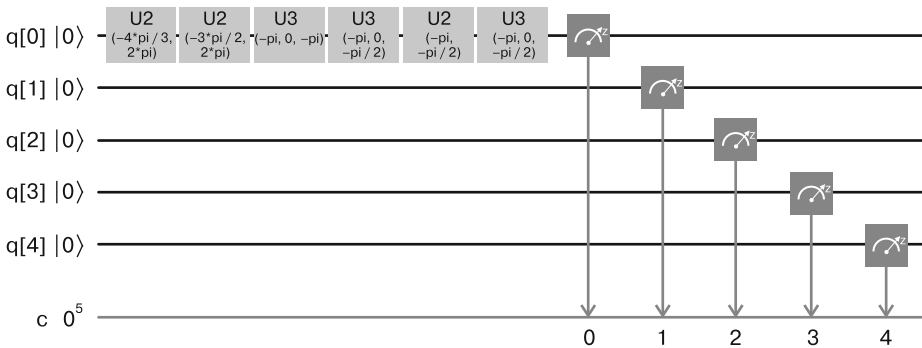
# Вывод полученных числовых значений
print ("Job id=" + str(result.get_job_id()) + " Status:" +
result.get_status())

#####
if __name__ == '__main__':
    start_time = time.time()
    main()
    print("--- %s seconds ---" % (time.time() - start_time))

```

## Квантовая схема для Composer

Программу из листинга 4.6 также можно создать в Composer от IBM Q Experience с помощью удобного пользовательского интерфейса с поддержкой перетаскивания объектов мышью. Просто перетащите вентиль на гистограмму кубита (рис. 4.8), установите параметры для вентилья и, наконец, сохраните и запустите на моделирующем или реальном устройстве.



**Рис. 4.8.** Схема для листинга 4.6, созданная в Composer от Q Experience

Тем, кто предпочитает ассемблер, Composer позволяет копировать и вставлять код непосредственно в консоль в режиме ассемблера (рис. 4.9). Он даже проанализирует любые синтаксические ошибки в вашем коде и покажет строки, в которых они содержатся.

Есть несколько способов выполнить ваш эксперимент в IBM Q Experience. Один из самых интересных — воспользоваться их прекрасным REST API.

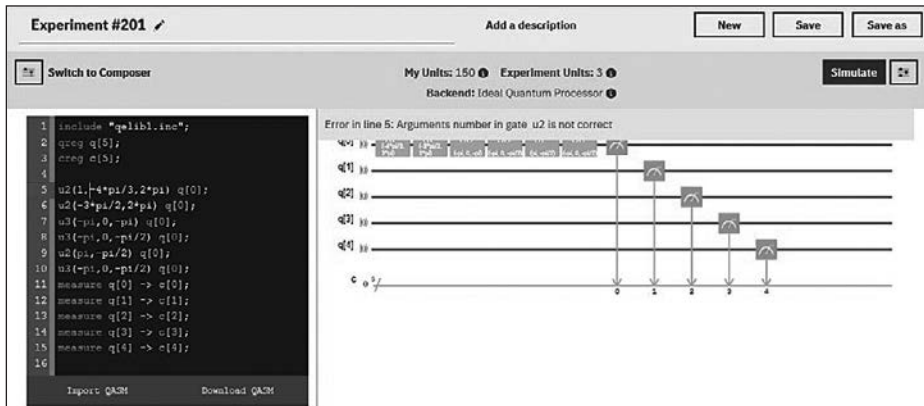


Рис. 4.9. Composer в режиме ассемблера для схемы, показанной на рис. 4.8

## Выполнение с помощью вашего любимого клиента REST

Это один из самых интересных способов взаимодействия с Q Experience. Используя простые REST-запросы, вы можете делать практически все, что делаете в Python или Composer:

- получить список устройств серверного ПО;
- получить аппаратные или калибровочные параметры для реальных устройств;
- получить информацию об очереди выполнения задания;
- получить статус задания или эксперимента;
- добавить или отменить задания;
- выполнить эксперимент и записать его в разделе Composer под названием Scores.

### ПРИМЕЧАНИЕ

REST API позволяет использовать любой язык, чтобы создать собственный интерфейс для взаимодействия с Q Experience (даже браузер). Этот API подробно описан в главе 3.

Существует два способа отправки экспериментов с использованием REST: через API заданий и выполнения. Посмотрим, как это делается.

## Запуск через API заданий

Вы можете применить любимый браузерный REST-клиент для отправки эксперимента, приведенного в листинге 4.6. Например, с помощью Chrome YARC (еще один клиент REST) создайте HTTP-запрос POST к конечной точке: [https://quantumexperience.ng.bluemix.net/api/jobs?access\\_token=ACCESS\\_TOKEN](https://quantumexperience.ng.bluemix.net/api/jobs?access_token=ACCESS_TOKEN).

Самое сложное — получить токен доступа или ключ доступа. Для этого вы должны пройти аутентификацию, используя свой токен API или имя пользователя и пароль. Обратите внимание на то, что токен API не следует путать с токеном доступа. Чтобы получить токен доступа, вы должны выполнить запрос аутентификации (см. раздел «Удаленный доступ через REST API» главы 3).

---

### ПРИМЕЧАНИЕ

YARC от Chrome позволяет создавать запросы REST и сохранять их как избранные. Создайте запрос аутентификации для IBM Q Experience, как описано в главе 3, сохраните его как избранный и используйте каждый раз для получения токена доступа при тестировании других вызовов REST API.

---

Полезные данные запроса представляют собой документ в формате JSON, показанный в листинге 4.7. Формат описан в табл. 4.2.

**Таблица 4.2.** Формат запроса для API заданий

Ключ	Описание
qasms	Массив программ на ассемблере, записанных в одну строку и разделенных символами конца строки (\n)
Shots	Количество итераций вашего кода
backend	Описание серверного ПО. В данном случае это ibmqx4
maxCredits	Подсказка о том, сколько кредитов будет вычтено из баланса вашего счета

**Листинг 4.7.** HTTP-запрос для API заданий

```
{
  "qasms": [{
    "qasm": "\n\ninclude \"qelib1.inc\";\nqreg q[5];\ncreg c[5];\nu2
(-4*pi/3,2*pi) q[0];\nu2(-3*pi/2,2*pi) q[0];\nu3(-pi,0,-pi) q[0];
\nu3(-pi,0,-pi/2) q[0];\nu2(pi,-pi/2) q[0];\nu3(-pi,0,-pi/2) q[0];
\nmeasure q -> c;\n"
  }],
  "shots": 1024,
  "backend": {
    "name": "ibmqx4"
  },
  "maxCredits": 3
}
```

Получив токен доступа, скопируйте и вставьте полезные данные из листинга 4.7 в свой REST-клиент, отправьте и дождитесь ответа. Если все идет хорошо, вы должны увидеть ответ, похожий на приведенный в листинге 4.8.

**Листинг 4.8.** HTTP-ответ от Q Experience

```
{
  "qasms": [
    {
      "qasm": "\n\ninclude \"qelib1.inc\";\nqreg q[5];\ncreg c[5];
\nu2(-4*pi/3,2*pi) q[0];\nu2(-3*pi/2,2*pi) q[0];\n
nu3(-pi,0,-pi) q[0];\nu3(-pi,0,-pi/2) q[0];\nu2(pi,-pi/2)
q[0];\nu3(-pi,0,-pi/2) q[0];\nmeasure q -> c;\n",
      "status": "WORKING_IN_PROGRESS",
      "executionId": "e9d758c3480a54a6455f72c84c5cc2a6"
    }
  ],
  "shots": 1024,
  "backend": {
    "id": "c16c5ddebbf8922a7e2a0f5a89cac478",
    "name": "ibmqx4"
  },
  "status": "RUNNING",
  "maxCredits": 3,
  "usedCredits": 3,
  "creationDate": "2018-04-24T00:12:07.847Z",
  "deleted": false,
  "id": "33d58594fcb7204e4d2ccdb65cd3c88c",
  "userId": "ef072577bd26831c59ddb212467821db"
}
```

Частично формат ответа описан в табл. 4.3.

**Таблица 4.3.** Формат ответа для API заданий

Ключ	Описание
qasms	Массив объектов, который содержит: <ul style="list-style-type: none"><li>• отправленный код;</li><li>• статус времени выполнения — <code>WORKING_IN_PROGRESS</code>, <code>COMPLETED</code> или <code>FAILED</code>;</li><li>• идентификатор выполнения для кода</li></ul>
shots	Количество итераций эксперимента
backend	Объект, содержащий такую информацию о серверном ПО, как название и идентификатор
status	Общий статус задания — <code>RUNNING</code> , <code>COMPLETED</code> или <code>FAILED</code>
maxCredits	Максимальное количество кредитов, используемое для этого запуска
usedCredits	Реальное количество кредитов, потраченных на запуск
creationDate	Дата создания задания
deleted	Истинно, если был отправлен запрос на удаление задания, иначе ложно. (Отмененные или удаленные задания удаляются из очереди не сразу, а через некоторое время)
id	Идентификатор задания
userId	Идентификатор хозяина

#### ПРИМЕЧАНИЕ

API заданий (а также выполнения) не документированы и не предназначены для прямого доступа в данный момент. Таким образом, со временем формат ответа может меняться. Возможно, в будущем это изменится и REST API станет частью официального SDK. Между тем наши с вами результаты могут различаться.

## Запуск через API выполнения

Основное различие между API выполнения и заданий заключается в том, что первый регистрирует эксперимент в Composer. Чтобы увидеть, как это происходит, создайте HTTP-запрос POST к конечной точке: [https://quantumexperience.ng.bluemix.net/api/codes/execute?access\\_token=TOKEN&shots=1&seed=SEED&deviceRunType=ibmqx4](https://quantumexperience.ng.bluemix.net/api/codes/execute?access_token=TOKEN&shots=1&seed=SEED&deviceRunType=ibmqx4).

Параметры запроса:

- `access_token` — ваш токен доступа;
- `shots` — количество итераций эксперимента;
- `seed` — случайное начальное число для выполнения (требуется только при запуске на моделирующем устройстве);
- `deviceRunType` — название устройства, на котором будет проводиться эксперимент.

Полезные данные запроса приведены в листинге 4.9. Каждый эксперимент должен иметь название. Тип кода — QASM2, ассемблерный код должен быть записан в одну строку, разделенную символами конца строки (`\n`).

### Листинг 4.9. Полезные данные HTTP-запроса API выполнения

```
{
  "name": "Experiment #20180410193125",
  "codeType": "QASM2",
  "qasm": "\n\ninclude \"qelib1.inc\";\nqreg q[5];\ncreg c[5];\nu2
(-4*pi/3,2*pi) q[0];\nu2(-3*pi/2,2*pi) q[0];\nu3(-pi,0,-pi) q[0];\nu3
(-pi,0,-pi/2) q[0];\nu2(pi,-pi/2) q[0];\nu3(-pi,0,-pi/2) q[0];
\nmeasure q -> c;\n"
}
```

Отправьте запрос, используя свой REST-клиент, и дождитесь результата. В листинге 4.10 приведен краткий формат ответа для эксперимента.

---

### СОВЕТ

Избавьте себя от головной боли. Всегда проверяйте, подключено ли устройство к сети и записан ли код на `qasm` в одну строку, включающую символы конца строки (`\n`), иначе у вас будет много проблем. Дважды и трижды проверьте это, или ваш запрос в большинстве случаев не будет успешно выполнен.

---



**Листинг 4.10.** Формат ответа для API выполнения

```

{
  "startDate": "2018-04-24T22:31:23.555Z",
  "modificationDate": 1524609083555,
  "typeCredits": "plan",
  "status": {
    "id": "WORKING_IN_PROGRESS"
  },
  "deviceRunType": "real",
  "ip": {
    "ip": "172.58.152.206",
    "country": "United States",
    "continent": "North America"
  },
  "shots": 1,
  "paramsCustomize": {},
  "deleted": false,
  "userDeleted": false,
  "id": "1203b1158e6ae537e8b770cb8049a6ae",
  "codeId": "e0f5c573eef75581cf16bce4187ecab8",
  "userId": "ef072577bd26831c59ddb212467821db",
  "infoQueue": {
    "status": "PENDING_IN_QUEUE",
    "position": 108
  },
  "code": {
    "type": "Algorithm",
    "active": true,
    "versionId": 1,
    "idCode": "e86d38c389f4449e62756922a1aa5729",
    "name": "Experiment #201",
    "jsonQASM": {
      "gateDefinitions": [],
      "topology": "3b8e671a5a3b56899e6e601e6a3816a1",
      "playground": [
        {
          "name": "q",
          "line": 0,
          "gates": [
            ""
          ]
        },
        {
          "name": "q",
          "line": 4,
          "gates": [
            {
              "name": "measure",
              "qasm": "measure",

```

```

        "position": 10,
        "measureCreg": {
            "line": 5,
            "bit": 4
        }
    }
]
},
{
    "name": "c",
    "line": 0
}
],
"numberGates": 7,
"hasMeasures": true,
"numberColumns": 11,
"include": "include \"qelib1.inc\";"
},
"qasm": "\n\ninclude \"qelib1.inc\";\nqreg q[5];\ncreg c[5];\nu2
(-4*pi/3,2*pi) q[0];\nu2(-3*pi/2,2*pi) q[0];\nu3(-pi,0,-pi) q[0];
\nu3 (-pi,0,-pi/2) q[0];\nu2(pi,-pi/2) q[0];\nu3(-pi,0,-pi/2)
q[0];\nmeasure q -> c;\n",
"codeType": "QASM2",
"creationDate": "2018-04-24T22:31:22.561Z",
"deleted": false,
"orderDate": 1524609083391,
"userDeleted": false,
"isPublic": false,
"id": "e0f5c573eef75581cf16bce4187ecab8",
"userId": "ef072577bd26831c59ddb212467821db"
}
}

```

В этом ответе содержится много информации, причем большая часть данных хорошо понятна. Тем не менее в табл. 4.4 описаны наиболее важные значения.

**Таблица 4.4.** Информация, возвращаемая для API выполнения

Ключ	Описание
status	Статус выполнения. Может принимать одно из следующих значений: WORKING_IN_PROGRESS, COMPLETED или FAILED
deviceRunType	Устройство, на котором будет запущен эксперимент: real (для реальных устройств) или simulator (для моделирующих устройств)

Ключ	Описание
infoQueue	Информация об очереди выполнения, содержащая: <ul style="list-style-type: none"><li>• статус — PENDING_IN_QUEUE;</li><li>• позицию в очереди</li></ul>
code	Очень подробное описание эксперимента, включающее: <ul style="list-style-type: none"><li>• квантовые вентили, параметры, позицию и многое другое;</li><li>• код ассемблера;</li><li>• разную информацию: название, тип, статус, версию и т. д.</li></ul>

---

### СОВЕТ

После получения ответа войдите в консоль IBM Q Experience. Название эксперимента должно отображаться в разделе Quantum Scores в Composer.

---

## Квантовый ассемблер: мощь, скрытая за кулисами

Вы, скорее всего, поняли, что происходит за кулисами, когда эксперимент выполняется в Composer или REST-клиенте. Схема транслируется в квантовый ассемблер (QASM), а затем выполняется на реальном или моделирующем устройстве. Квантовый ассемблер является промежуточным представлением кода высокого уровня Python и результатом сотрудничества между IBM Q Experience и сообществом разработчиков ПО с открытым исходным кодом.

---

### ПРИМЕЧАНИЕ

QASM основан на классическом аналоге, который стал своего рода утраченным искусством. Тем не менее он не столь пугающий, как его собрат. Фактически в его основе действительно лежит грамматика классического ассемблера.

---

Формально жизненный цикл вашей программы на Python или схемы в Q Experience можно описать как нечто среднее между квантовой и классической частями вычислений со следующими шагами.

○ *Компиляция.* Это автономный шаг, выполняемый на классическом компьютере. Когда работает схема, написанная на Python или составленная в Composer, классический компилятор переводит высокоуровневое представление (например, на Python) в промежуточное представление на QASM. Этот шаг имеет следующие характеристики:

- конкретные параметры задачи еще неизвестны;
- не требуется взаимодействие с квантовым компьютером;
- можно скомпилировать классические процедуры в объектную программу и выполнить первоначальную оптимизацию. Например, программа на Python, приведенная в листинге 4.6, и соответствующая схема, составленная в Composer, транслированы в ассемблер, показанный в листинге 4.11.

**Листинг 4.11.** Код QASM для программы на Python из листинга 4.6

```
include "qelib1.inc";
qreg qr[5];
creg cr[5];
u2(-4.18879020478639,6.28318530717959) qr[0];
u2(-4.71238898038469,6.28318530717959) qr[0];
u3(-3.14159265358979,0,-3.14159265358979) qr[0];
u3(-3.14159265358979,0,-1.57079632679490) qr[0];
u3(3.14159265358979,-1.57079632679490) qr[0];
u3(-3.14159265358979,0,-1.57079632679490) qr[0];
measure qr[0] -> cr[0];
measure qr[1] -> cr[1];
measure qr[2] -> cr[2];
```

○ *Генерация схемы.* Код QASM из предыдущего шага передается на фазу генерации цепи. Этот шаг выполняется на классическом компьютере, где известны конкретные параметры задачи и может произойти определенное взаимодействие с квантовым компьютером. Шаг имеет следующие характеристики:

- это онлайн-фаза (происходит на квантовом компьютере);
- выход представляет собой набор квантовых цепей или квантовых базовых блоков с соответствующими классическими инструкциями

управления и классическим объектным кодом, необходимым во время выполнения.

- *Выполнение.* Этот шаг производится на физическом квантовом компьютере. Вход представляет собой набор квантовых схем, выраженных с использованием промежуточного представления квантовых схем. Они выполняются на контроллере низкого уровня, а выход представляет собой набор результатов измерений, возвращаемых из контроллера высокого уровня.
- *Постобработка.* Выполняется на классическом компьютере, будет получен набор обработанных результатов измерений. Выходные данные являются окончательным результатом квантовых вычислений (рис. 4.10).

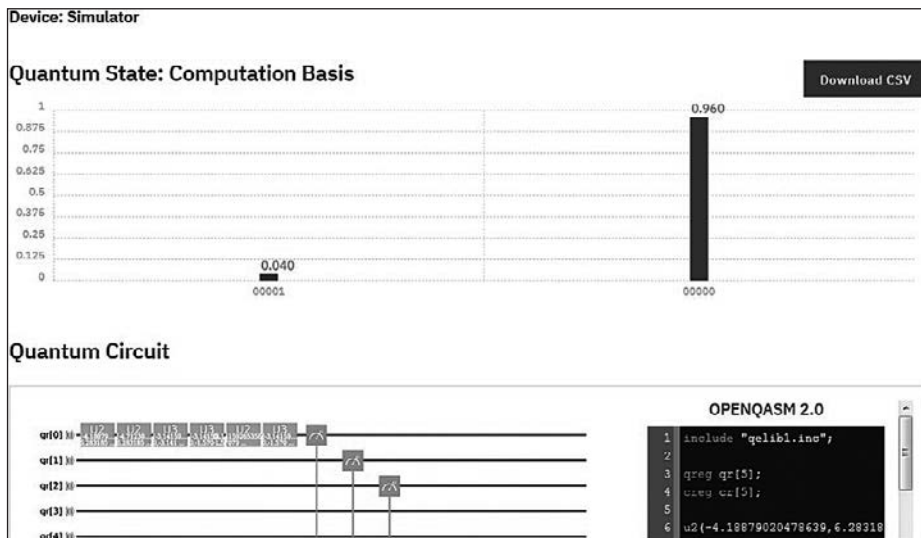


Рис. 4.10. Результаты постобработки для жизненного цикла схемы из листинга 4.6

В общем, синтаксис квантового ассемблера не так страшен по сравнению с его классическим аналогом. На самом деле программировать напрямую на квантовом ассемблере оказывается проще и быстрее, чем на Python. Далее представлен набор простых приемов, которые можно использовать, если вы решите написать код непосредственно на QASM.

- Всегда начинайте с включения заголовка `qelib1.inc`. Он содержит аппаратные примитивы Q Experience (квантовые вентили). Логические элементы, представленные в этой библиотеке, описаны в табл. 4.5 для однокубитных и в табл. 4.6 для многокубитных вентилях.

**Таблица 4.5.** Однокубитные вентили, представляемые квантовым ассемблером

Название	Описание
<code>u3(theta,phi,lambda)</code>	Однокубитный трехпараметрический двухимпульсный вентиль
<code>u2(phi,lambda)</code>	Однокубитный двухпараметрический одноимпульсный вентиль
<code>u1(lambda)</code>	Однопараметрический однокубитный вентиль
<code>Id</code>	Эквивалент матрицы тождественного преобразования или $u(0,0,0)$
<code>X</code>	Паули X или $\sigma_x$ или инверсия битов
<code>Y</code>	Паули Y или $\sigma_y$
<code>Z</code>	Паули Z или $\sigma_z$
<code>rx(theta)</code>	Поворот вокруг оси X на $\theta$ градусов
<code>ry(theta)</code>	Поворот вокруг оси Y на $\theta$ градусов
<code>rz(Phi)</code>	Поворот вокруг оси Z на $\theta$ градусов
<code>H</code>	Адамара — переводит одиночный кубит в суперпозицию состояний
<code>S</code>	Квадратный корень из Z — вентиль фазового сдвига $\sqrt{Z}$
<code>Sdg</code>	$S^\dagger$ — эрмитово сопряжение S. Алгебраически это определено как комплексное сопряжение транспонированной матрицы $\sqrt{Z}$
<code>T</code>	Вентиль фазового сдвига $\sqrt{S}$
<code>Tdg</code>	$T^\dagger$ — эрмитово сопряжение $\sqrt{S}$

**Таблица 4.6.** Многокубитные вентили, представляемые квантовым ассемблером

Название	Описание
<code>cx c,t</code>	Управляемое НЕ (CNOT) — инвертирует состояние второго кубита (t), только если управляющий кубит (c) имеет значение 1. Используется для запутывания кубитов

Название	Описание
cz a,b	Управляемый фазовый сдвиг — применяет фазовый сдвиг, только если управляющий кубит (a) имеет значение 1
cuy a,b	Управляемый Y — применяет Паули Y, только если управляющий кубит (a) имеет значение 1
ch a,b	Управляемый H — переводит кубит (b) в суперпозицию, только если управляющий кубит (a) имеет значение 1
csx a,b,c	Трехкубитный вентиль Тоффли — инвертирует состояние кубита c, только если управляющие кубиты a и b имеют значение 1

- Объявить регистр (массив) кубитов просто. Например, объявление регистра, состоящего из пяти кубитов: `qreg qr[5]`. (**Примечание:** все инструкции разделяются точкой с запятой.)
- Чтобы объявить регистр, состоящий из пяти классических битов, используйте `creg cr[5]`.
- Чтобы применить вентиль к определенному кубиту, просто введите название вентиля и управляемый кубит. Например, чтобы перевести первый кубит в суперпозицию (для квантового генератора случайных чисел), используйте `hq[0]`.
- Последним шагом в вашей программе всегда должно быть измерение кубита. Например, чтобы измерить кубит, который мы перевели в состояние суперпозиции, и сохранить его в первом классическом регистре, возьмите `measure qr[0] -> cr[0]`.

Обратите внимание, что квантовые компьютеры являются вероятностными машинами, следовательно, невозможно узнать, в каком именно состоянии находится кубит (это запрещено квантовой механикой). Таким образом, мы получаем лишь вероятность того, что кубит находится в состоянии 0 или 1. Для простого квантового генератора случайных чисел на нулевом кубите `hq[0]`, упомянутом ранее, мы можем использовать вероятность состояния 1 в качестве случайного числа. Это можно представить в виде графика, выдаваемого Composer в IBM Q Experience, когда результаты собираются после выполнения кода ассемблера (см. рис. 4.9).

Вы сделали первый шаг по карьерной лестнице в качестве квантового программиста, работающего в облаке. Используя высокоуровневый SDK для Python и мощный механизм квантового ассемблера, можно проводить эксперименты на потрясающей платформе IBM Q Experience. Эти навыки окажутся полезными через несколько лет, когда квантовые компьютеры начнут появляться в центрах обработки данных. В следующей главе мы перейдем на новый уровень, рассмотрев набор алгоритмов, которые демонстрируют волшебство квантовой механики в вычислениях. Так что читайте дальше.



# 5

## **Запускаем движки: от квантовой генерации случайных чисел до телепортации с остановкой на сверхплотном кодировании**

В этой главе вы познакомитесь с тремя замечательными возможностями квантовых систем по обработке информации. Мы начнем с одной из самых простых процедур, исследуя принципиально случайную природу квантовой механики как источника истинной случайности. Далее в главе рассматриваются две взаимосвязанные процедуры, называемые сверхплотным кодированием и квантовой телепортацией. Изучая сверхплотное кодирование, вы узнаете, как можно отправить два классических бита информации с помощью одного кубита. Читая о квантовой телепортации — как квантовое состояние кубита можно воссоздать с помощью гибридной классическо-квантовой процедуры передачи информации. Все алгоритмы включают разработку схем для IBM Q Experience Composer, а также код на Python и QASM. Результаты разберем и проанализируем, так что начнем.

### **Квантовый генератор случайных чисел**

В этом разделе вы узнаете, как вероятностную природу квантового компьютера можно использовать для генерации случайных битов или чисел с помощью вентилля Адамара.

## Генератор случайных битов на основе вентиля Адамара

Вентиль Адамара — один из фундаментальных логических элементов квантовых информационных систем. Он применяется для перевода кубита в суперпозицию состояний. С алгебраической точки зрения описывается матрицей:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Чтобы лучше понять, каким образом данная матрица переводит кубит в суперпозицию, рассмотрим геометрическое представление одиночного кубита.

На рис. 5.1 базисные состояния кубита описываются скобочной нотацией, где  $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  и  $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . Вспомните из предыдущей главы, что кет-вектор — это просто единичный вектор (вектор, длина которого равна 1). Таким образом, общее состояние (или суперпозиция) определяется единичным вектором  $\psi = \alpha|0\rangle + \beta|1\rangle$ , где  $\alpha$  и  $\beta$  — комплексные коэффициенты. Применение вентиль  $H$  к базисным состояниям дает:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \left( \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{|0\rangle + |1\rangle}{\sqrt{2}},$$

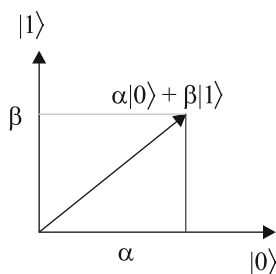
$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \left( \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

и полученное методом суперпозиции состояние  $\psi$ :

$$\psi = \alpha|0\rangle + \beta|1\rangle \rightarrow \alpha \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) + \beta \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \frac{\alpha + \beta}{\sqrt{2}} |0\rangle + \frac{\alpha - \beta}{\sqrt{2}} |1\rangle.$$

В общем, вентиль Адамара расширяет диапазон состояний, возможных для квантовой схемы. Это важно, потому что расширение со-

стояний позволяет искать быстрые способы решения, что ускоряет вычисления.



**Рис. 5.1.** Геометрическое представление общего (полученного методом суперпозиции) состояния  $\psi$  кубита

#### ПРИМЕЧАНИЕ

Согласно принципам квантовой механики мы не можем с уверенностью предсказать значения коэффициентов  $\alpha$  и  $\beta$  в предыдущих базисных состояниях, даже учитывая полное знание законов физики или начальных условий, в которых находится частица. Лучшее, что мы можем сделать, — это рассчитать вероятность.

С учетом этого реализация схемы генератора случайных битов упрощена настолько, насколько это возможно. В IBM Q Experience Composer создайте схему с вентилем Адамара для первого кубита, а затем выполните измерение в базисном состоянии (рис. 5.2).

Пожалуй, не стоит запускать ее на реальном устройстве, так как процесс может оказаться длительным (помните, что выполнение запланировано и может занять некоторое время в зависимости от количества заданий в очереди выполнения). К тому же каждое выполнение на реальном устройстве уменьшает количество ваших кредитов. Запустите схему на моделирующем устройстве, чтобы получить результат немедленно (рис. 5.3). Обратите внимание на то, что появление каждого результата

(0 или 1) имеет равную вероятность  $1/2$ , поэтому мы можем создавать случайные данные: если вероятность для результата  $1 > 1/2$ , получим 1, в противном случае — 0.

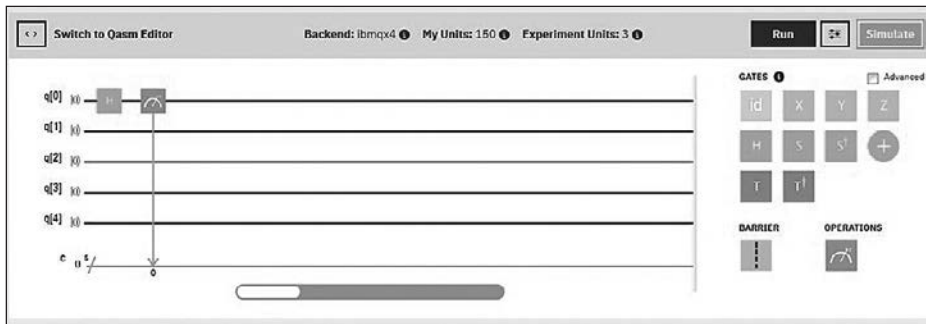


Рис. 5.2. Схема для генератора случайных битов

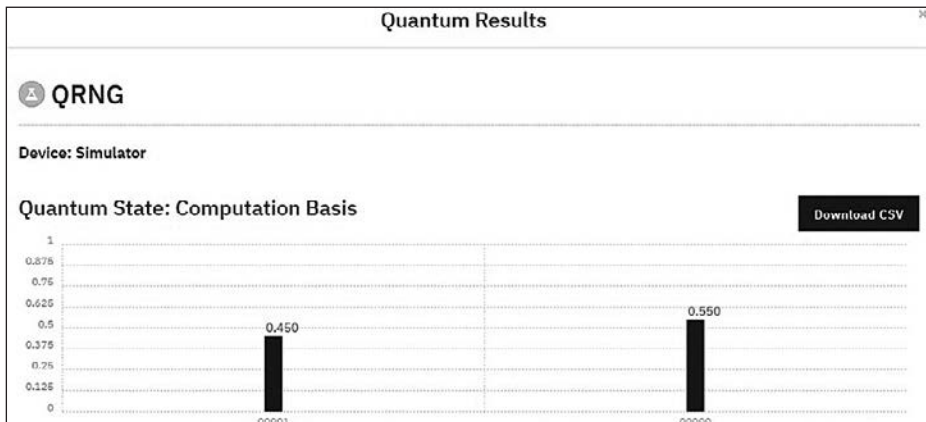


Рис. 5.3. Результаты выполнения для схемы, приведенной на рис. 5.2

Конечно, это очень неэффективный способ генерации случайных битов. Лучше было бы написать сценарий на Python с помощью QISKit для программного создания схемы в этом задании.

В листинге 5.1 показан простой сценарий для генерации  $n$  случайных чисел с помощью  $x$  кубитов, где число битов равно  $2^x$ . По умолчанию сценарий генерирует десять восьмибитовых случайных чисел, используя

три кубита, то есть  $n = 10$  и  $x = 3$ , при этом  $2^3 = 8$ . Рассмотрим сказанное подробнее.

- В 12-й строке определяется функция `qrng` для создания схемы с применением  $n$  кубитов.
- С помощью `QISKitAPI` в строках 15–21 создается `QuantumProgram` с  $n$  кубитами и  $n$  классическими регистрами для хранения измерений.
- Ко всем кубитам применяется вентиль Адамара, затем выполняется измерение для каждого, и, наконец, результат сохраняется в  $n$ -м классическом регистре (строки 30–35).
- Схема компилируется для запуска на удаленном моделирующем устройстве от Q Experience с помощью системного вызова `set_api(API-TOKEN, URL)`. Обратите внимание, что вам понадобится дескриптор конфигурации с токеном API и URL-адресом конечной точки. Схема запускается, и полученные в результате числовые значения накапливаются (строки 40–51).
- Наконец, чтобы сгенерировать случайные биты, посмотрите на итоговые числа. Например, получены результаты `{ '100': 133, '101': 134, '011': 131, '110': 125, '001': 109, '111': 128, '010': 138, '000': 126 }`. Для каждого результата, если значение больше средней вероятности, вы получаете 1, в противном случае — 0. Средняя вероятность рассчитывается делением количества запусков (в данном случае 1024) на количество результатов (для  $2^x$ , где  $x$  — количество кубитов (по умолчанию 3), получим  $1024 / 8 = 128$ ). Таким образом, для предыдущих результатов получается следующее.

133	1	
134	1	
131	1	11100010 = 226
125	0	
109	0	
128	0	
138	1	
126	0	

**Листинг 5.1.** Квантовая программа, позволяющая генерировать  $n$  случайных  $2^x$ -битовых чисел

```
#####
import sys,time
import qiskit
import logging
from qiskit import QuantumProgram

# конфигурация Q Experience
sys.path.append('../Config/')
import Qconfig

# Генерация случайного  $2^{**n}$ -битового числа, где  $n$  – количество кубитов
def qrng(n):

    # Создать программу
    qp = QuantumProgram()

    # Создать  $n$  кубитов
    quantum_r = qp.create_quantum_register("qr", n)

    # Создать  $n$  классических регистров
    classical_r = qp.create_classical_register("cr", n)

    # Создать схему
    circuit = qp.create_circuit("QRNG", [quantum_r], [classical_r])

    # Включить ведение журналов
    qp.enable_logs(logging.DEBUG);

    # Вентиль Адамара для всех кубитов
    for i in range(n):
        circuit.h(quantum_r[i])

    # Измерить  $n$ -й кубит и сохранить результат в  $n$ -м классическом бите
    for i in range(n):
        circuit.measure(quantum_r[i], classical_r[i])

    # Серверное моделирующее устройство
    backend = 'ibmq_qasm_simulator'

    # Набор исполняемых схем
    circuits = ['QRNG']

    # Скомпилируйте вашу программу: ASM print(qp.get_qasm('Circuit')), JSON:
    print(str(qobj))
```

```
# Установите APIToken и URL-адрес API Q Experience
qp.set_api(Qconfig.APIToken, Qconfig.config['url'])
shots=1024
result = qp.execute(circuits, backend, shots=shots, max_credits=3,
timeout=240)

# Вывод полученных числовых значений
# counts={'100': 133, '101': 134, '011': 131, '110': 125, '001': 109,
# '111': 128, '010': 138, '000': 126}
counts = result.get_counts('QRNG')
bits = ""
for v in counts.values():
    if v > shots/(2**n) :
        bits += "1"
    else:
        bits += "0"

return int(bits, 2)

#####
if __name__ == '__main__':
    start_time = time.time()
    numbers = []

    # Генерация 100 восьмибитовых случайных чисел
    size = 10
    qubits = 3 # bits = 2**qubits

    for i in range(size):
        n = qrng(qubits)
        numbers.append(n)

    print ("list=" + str(numbers))
    print("--- %s seconds ---" % (time.time() - start_time))
```

---

### ВНИМАНИЕ

Перед выполнением любой программы убедитесь, что конфигурация задана правильно, включая корректные токен API и URL конечной точки. Это основной источник неприятностей. Помните, что программа не будет выполняться, если вы допустите промах на этом крайне важном шаге.

---

Квантовая схема для листинга 5.1 показана на рис. 5.4. В схеме используются три кубита для генерации восьмибитовых случайных чисел в диапазоне между 0 и 255.

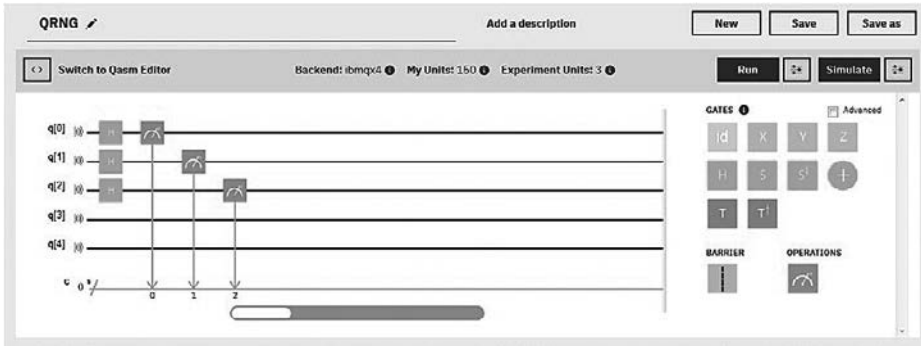


Рис. 5.4. Схема в Q Experience для программы из листинга 5.1

Накопим данные после нескольких запусков и протестируем результаты.

## Тестирование результатов на случайность

Linux предоставляет удобную программу `ent` (сокращение от `entropy` — «энтропия»), предназначенную для проверки последовательности псевдослучайных чисел. Мы можем воспользоваться ею для проверки чисел, сгенерированных в предыдущем разделе.

---

### ПРИМЕЧАНИЕ

Пользователям Windows: бинарный файл для Windows 32 доступен для скачивания на сайте проекта. Он включен также в исходники для этой главы, находится в `Workspace\Ch05\ent.exe`.

---

Итак, я собрал около 200 случайных восьмибитовых чисел, сгенерированных с помощью программы из листинга 5.1.



С использованием `ent` данную последовательность можно протестировать вводом команды `ent [infile]`, как показано далее:

```
C:\Workspace\Ch05>ent qrnd-stdout.txt
Entropy = 3.122803 bits per byte.
Optimum compression would reduce the size of this 805 byte file by 60
percent.
```

Chi square distribution for 805 samples is 29149.54, and randomly would exceed this value less than 99.9 percent of the times.

Arithmetic mean value of data bytes is 46.1503 (127.5 = random).  
 Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).  
 Serial correlation coefficient is -0.356331 (totally uncorrelated = 0.0).

Считается, что критерий хи-квадрат определяет качество случайной последовательности. Если процентное распределение хи-квадрат меньше 1 % или больше 99 %, то последовательность недостаточно хороша. Мой вывод дает 99,9 %, что указывает на то, что случайность чисел низкая. Вероятно, это связано с тем, что я использовал дистанционное моделирующее устройство. Видимо, оно основано на генераторе случайных чисел от UNIX (довольно низкого качества), применяемом по умолчанию. Посмотрите, будет ли ваша последовательность лучше. В табл. 5.1 сравниваются результаты, полученные от различных детерминированных и квантовых источников (предоставлены разработчиками `ent`).

**Таблица 5.1.** Результаты тестирования на случайность для различных источников

Источник	Критерий хи-квадрат
<code>rand()</code> от UNIX	99,9 % для 500 000 образцов (плохо)
Генератор UNIX, улучшенный Park & Miller	97,53 % для 500 000 образцов (чуть лучше)
HotBits: случайные числа на основе радиоактивного полураспада	40,98 % для 500 000 образцов (наилучший результат)

Таблица ясно показывает, что `rand()` от UNIX не следует доверять генерацию случайных чисел. Если требуется получить большое количество действительно случайных чисел (например, для генерации ключей шифрования), используйте квантовый источник, такой как HotBits. В общем, цель этого раздела заключается в том, чтобы получить первоначальное

представление о предмете с помощью простой квантовой схемы для генерации случайных чисел. В следующем разделе мы перейдем на другой уровень, познакомившись с причудливым протоколом квантовой передачи данных под названием «сверхплотное кодирование».

## Сверхплотное кодирование

Сверхплотное кодирование (super dense coding, SDC) — это протокол передачи данных, который демонстрирует замечательные возможности квантовой системы по обработке информации. Формально SDC представляет собой простую процедуру передачи двух классических битов информации другой стороне с помощью одного кубита. Этот протокол проиллюстрирован на рис. 5.5.

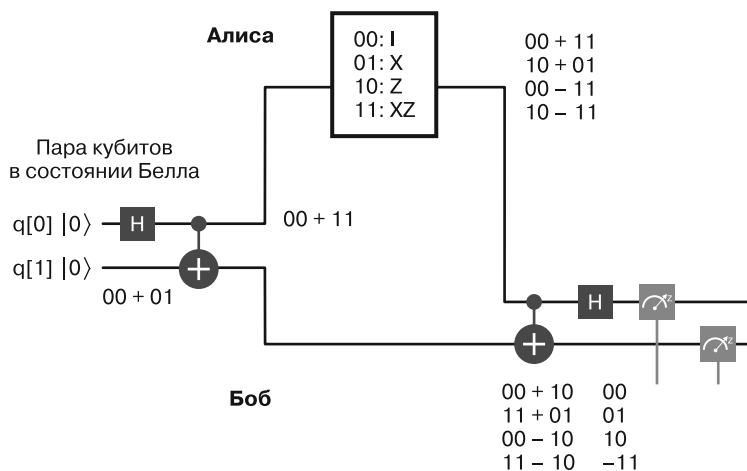


Рис. 5.5. Протокол сверхплотного кодирования

1. Процесс начинается с того, что третья сторона (Чарли) генерирует так называемую пару кубитов в состоянии Белла. Чарли начинает с двух кубитов в базисном состоянии  $|0\rangle$ , применяя вентиль Адамара к первому кубиту, чтобы создать суперпозицию. Затем задействует вентиль CNOT, используя первый кубит в качестве управляющего (точка), а второй — в качестве управляемого (+). Это дает состояния, приведенные в табл. 5.2.

**Таблица 5.2.** Пара кубитов в состоянии Белла

Вентиль	Результирующее состояние	Подробности
H	$ 00\rangle \rightarrow  00\rangle +  10\rangle$	Когда вентиль Адамара применяется к первому кубиту, он переходит в состояние суперпозиции. Таким образом, мы получаем состояние $00 + 10$ , где второй кубит остается как 0. Обратите внимание на то, что в матрице Адамара для простоты был опущен квадратный корень из 2
CNOT	$ 00\rangle +  10\rangle \rightarrow  00\rangle +  11\rangle$	Вентиль CNOT запутывает оба кубита. В частности, он инвертирует управляемый кубит (+), если управляющий (.) имеет значение 1. Таким образом мы инвертируем второй кубит, если первый имеет значение 1, получая $00 + 11$

2. На втором этапе процесса первый кубит отправляется Алисе, а второй — Бобу. Обратите внимание, что Алиса и Боб могут быть значительно удалены друг от друга. Цель протокола состоит в том, чтобы Алиса отправила два классических бита информации Бобу, используя свой кубит. Но прежде, чем она это сделает, ей нужно применить набор квантовых правил (или вентилях) к своему кубиту в зависимости от двух битов информации, которые она хочет отправить (табл. 5.3).

**Таблица 5.3.** Правила сверхплотного кодирования

Правила	Результирующие состояния
00: I (единичный вентиль)	$I(00 + 11) = 00 + 11$
01: X	$X(00 + 11) = 10 + 01$
10: Z	$Z(00 + 11) = 00 - 11$
11: ZX	$ZX(00 + 11) = 10 - 11$

3. Таким образом, если она отправляет 00, то ничего не делает со своим кубитом (применяет единичный вентиль (identity)). Если отправляет 01, то задействует вентиль X (или инверсию битов). Чтобы получить 10, она применяет вентиль Z. Обратите внимание, что последний меняет знак (или фазу) на противоположный, если кубит имеет значение 1. Тогда  $Z|0\rangle = |0\rangle$ ,  $Z|1\rangle = -|1\rangle$ . Наконец, если она отправляет 11, то применяет вентили XZ к своему кубиту. Алиса отправляет свой кубит Бобу в качестве последнего шага данного процесса.

4. Боб получает кубит Алисы (нулевой) и использует свой кубит, чтобы запустить в обратном направлении процесс состояния Белла, созданного Чарли. То есть он применяет к первому кубиту вентиль CNOT, затем вентиль Адамара и, наконец, выполняет измерение на двух кубитах, чтобы получить два классических бита, закодированных в кубите Алисы (табл. 5.4).

**Таблица 5.4.** Состояния кубитов после восстановления

Вентиль	Результирующее состояние	Подробности
CNOT	00 + 10 11 + 01 00 – 10 11 – 10	Начинаем с состояний Алисы на втором шаге: 00 + 11 10 + 01 00 – 11 10 – 11  Вентиль CNOT инвертирует второй кубит, если первый имеет значение 1, давая в результате состояния, приведенные во втором столбце
H	00 01 10 –11	Применение вентилей Адамара к первому кубиту в последней строке дает результаты, приведенные во втором столбце. Когда Боб выполняет измерения в вычислительных базисных состояниях, он получает четыре возможных результата с вероятностью 1 каждый. Эти результаты соответствуют тому, что Алиса хотела отправить на втором шаге в первом столбце. Обратите внимание, что последний результат со знаком минус. Тем не менее, поскольку вероятность рассчитывается как квадрат амплитуды, –1 становится 1, что корректно

Сведем все это воедино в схеме, составленной в Composer от IBM Q Experience.

## Схема в Composer

На рис. 5.6 показаны схема для сверхплотного кодирования, а также код в Composer.

- Создание схемы начинается с пары кубитов в состоянии Белла, то есть `qubit[0]` переводится в суперпозицию (с помощью вентилей Адамара) и запутывается с `qubit[1]` с использованием вентилей CNOT.

- Два следующих вентиля представляют правила кодирования Алисы. Помните, что она применяет единичный вентиль (ничего не выполняется) для кодирования битов 00, X — для 01, Z — для 10 и ZX — для 11. В данном конкретном случае кодированию подвергаются биты 11. Это показано на рис. 5.6 слева от разделительного знака. Учтите, что разделитель станет препятствовать выполнению до тех пор, пока все вентили не будут использованы обоими кубитами.
- Справа от разделительного знака находится протокол Боба. По сути, он выполняет в обратном порядке те же операции, что и Алиса. Он применяет к кубитам сначала вентили CNOT, а затем Адамара. Наконец, на обоих кубитах производится измерение, чтобы извлечь два закодированных классических бита.

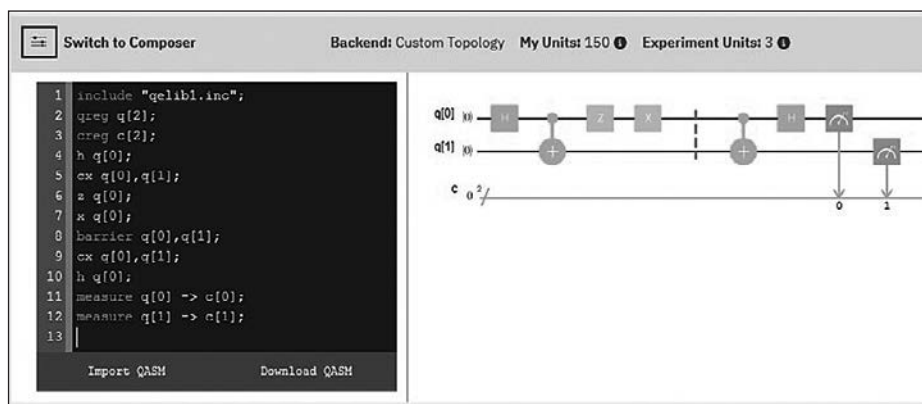


Рис. 5.6. Схема для сверхплотного кодирования для Q Experience

Запустите предыдущую схему на моделирующем устройстве. Результат должен быть гистограммой с вероятностью, очень близкой к 1 или равной ей для результирующего состояния 11. Он должен соответствовать тому, который будет получен в следующем разделе с использованием сценария Python.

## Удаленный запуск с использованием Python

В листинге 5.2 приведен сценарий Python для схемы, изображенной на рис. 5.6.

- В строках 17–19 создаются два кубита и два классических регистра для хранения результатов.
- Затем создается сверхплотная схема с запутанной парой кубитов в состоянии Белла (строки 22–24).
- Алиса кодирует 11, применяя вентили ZX. По своему усмотрению прокомментируйте любую из этих строк для кодирования другой пары, а затем убедитесь, что результат соответствует схеме кодирования Алисы (строки 32–35).
- Боб выполняет в обратном порядке операции Алисы и измеряет кубиты (строки 38–41).
- Наконец, схема выполняется на удаленном моделирующем устройстве (`ibmq_qasm_simulator`) и результаты отображаются в виде графиков, поддерживаемых в Python.

**Листинг 5.2.** Сценарий Python для сверхплотного кодирования

```
import sys,time,math

# Импорт QISKit
from qiskit import QuantumCircuit, QuantumProgram

sys.path.append('../Config/')
import Qconfig

# Импорт основных средств вывода графиков
from qiskit.tools.visualization import plot_histogram

def main():
    # Первоначальная настройка квантовой программы
    Q_program = QuantumProgram()
    Q_program.register (Qconfig.APIToken, Qconfig.config["url"])

    # Создание регистров
    q = Q_program.create_quantum_register("q", 2)
    c = Q_program.create_classical_register("c", 2)

    # Квантовая схема для создания запутанного состояния
    superdense = Q_program.create_circuit("superdense", [q], [c])
    superdense.h(q[0])
    superdense.cx(q[0], q[1])

    # Для 00 ничего не делать
    # Для 10 применить вентиль X
```

```

# superdense.x(q[0])
# Для 01 применить вентиль Z
# superdense.z(q[0])
# Алиса: Для 11 применить вентили ZX
superdense.z(q[0])
superdense.x(q[0])
superdense.barrier()

# Боб
superdense.cx(q[0], q[1])
superdense.h(q[0])
superdense.measure(q[0], c[0])
superdense.measure(q[1], c[1])

circuits = ["superdense"]
print(Q_program.get_qasms(circuits)[0])

backend = "ibmq_qasm_simulator" #квантовое устройство ibmqx2
shots = 1024 # количество запусков эксперимента

result = Q_program.execute(circuits, backend=backend, shots=shots,
max_credits=3, timeout=240)

print("Counts:" + str(result.get_counts("superdense")))

plot_histogram(result.get_counts("superdense"))

#####
# main
if __name__ == '__main__':
    start_time = time.time()
    main()
    print("--- %s seconds ---" % (time.time() - start_time))

```

Рассмотрим результаты единичного запуска программы из листинга 5.2 в следующем разделе.

## Результаты

Вот стандартный вывод для запуска программы из листинга 5.2:

```

C:\python36-64\python.exe p05-superdensecoding.py
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];

```

```
h q[0];
cx q[0],q[1];
z q[0];
x q[0];
barrier q[0],q[1];
cx q[0],q[1];
h q[0];
measure q[0] -> c[0];
measure q[1] -> c[1];

Counts:{'11': 1024}
--- 167.52969431877136 seconds ---
```

Сценарий выводит код ассемблера для схемы, а также количество появлений выходного значения: {'11': 1024} плюс время выполнения. Количество появлений результата используется для расчета вероятности данного выходного значения путем деления количества появлений результата (1024) на количество запусков (1024). Таким образом, вероятность равна 1 для результата 11, как показано на графике, который выводится на последнем шаге в листинге 5.2 (рис. 5.7). Обратите внимание, что при выполнении на моделирующем устройстве вероятность всегда будет равна 1, то есть `count = shot`. Но если вы работаете на реальном квантовом устройстве, из-за шума и ошибок, обусловленных окружающей средой, количество появлений результата будет меньше 1024, что даст вероятность меньше 1.

Таким образом, сверхплотное кодирование обеспечивает средства для кодирования двух классических битов в одном кубите. Стоит упомянуть: согласно квантовым вычислениям невозможно сохранить более одного классического бита на кубит, что, кажется, противоречит показанному в этом протоколе. На самом деле никакого противоречия нет. Протокол работает, потому что кубиты Алисы и Боба запутаны через пару в состоянии Белла. Это позволяет отправлять два классических бита в запутанном кубите Алисы. В общем, вы можете хранить максимум два классических бита на кубит при условии, что ваш кубит запутан с другим в паре в состоянии Белла.

В общих чертах этот протокол можно интерпретировать как набор модульных абстракций: модуль генератора пары в состоянии Белла для создания двух запутанных кубитов, за которым следует модуль кодера информации, применяющий правила Алисы для кодирования двух



классических битов информации. Наконец, модуль декодера извлекает классические биты из кубитов, предоставляемых парой в состоянии Белла так же, как и модулем кодера (своего рода инструмент квантовой архивации и распаковки архивов (zip/unzip), если хотите). Сверхплотное кодирование дает общую картину квантовой обработки информации и поможет вам понять тему следующего раздела — квантовую телепортацию.

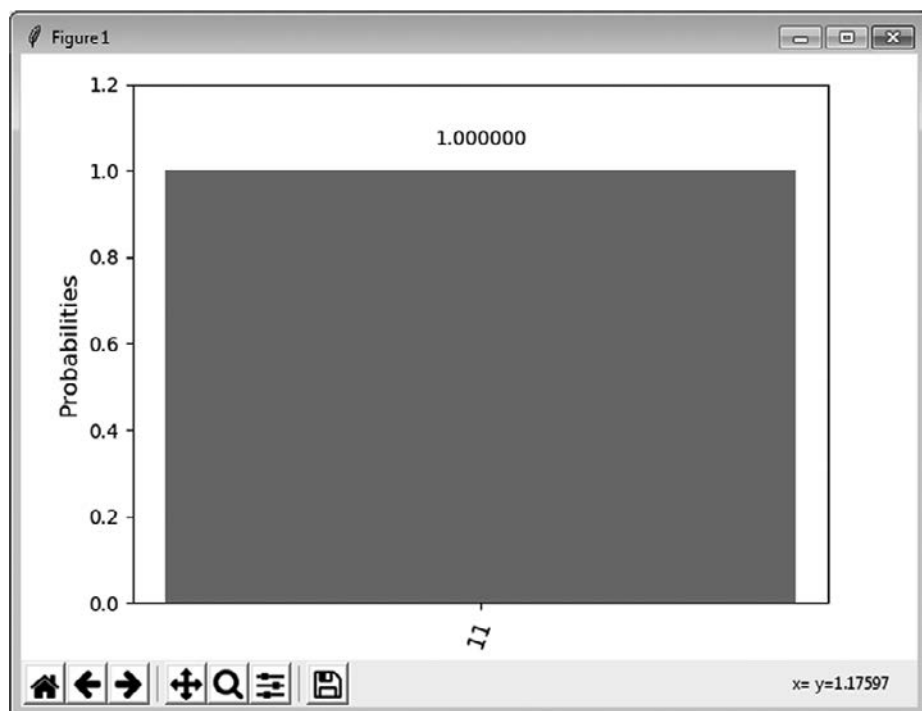


Рис. 5.7. Графическое отображение результатов для сверхплотного кодирования

#### ПРИМЕЧАНИЕ

Этот простой протокол был разработан в 1992 году физиком Чарльзом Беннетом, почти через 70 лет после открытия квантовой механики. Несмотря на относительную простоту, это отнюдь не стандартная процедура, и, если копнуть глубже, можно обнаружить много интересного.

## Квантовая телепортация

Квантовая телепортация — это процедура, тесно связанная со сверхплотным кодированием. Возможно, термин «телепортация» не совсем подходит, так как мы на самом деле ничего не телепортируем, по крайней мере в том смысле, в каком это делается в научной фантастике или «Звездном пути». Формально квантовая телепортация представляет собой процесс, с помощью которого состояние кубита  $\psi$  может передаваться из одного местоположения в другое с помощью классической связи и пары в состоянии Белла, обсуждавшейся в предыдущем разделе. Процедура в краткой форме представлена на рис. 5.8.

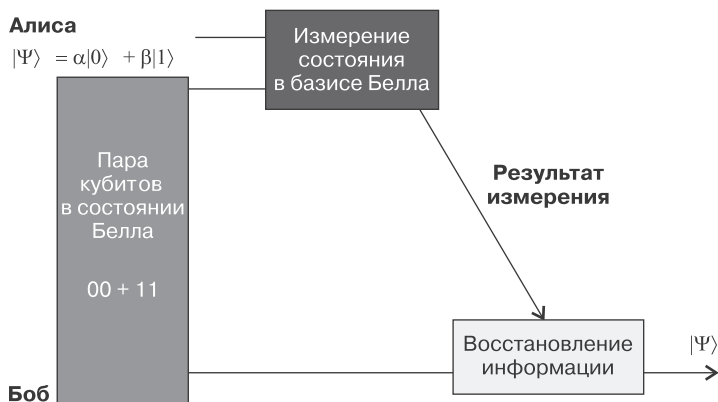


Рис. 5.8. Рабочий цикл квантовой телепортации

1. Алиса и Боб начинают с того, что разделяют пару запутанных кубитов в состоянии Белла. Один из них идет к Алисе, а другой — к Бобу, которые находятся в разных местах на расстоянии друг от друга. Представьте, что пара кубитов в состоянии Белла подготовлена третьей стороной (Чарли).
2. Алиса готовит свой кубит к телепортации в состоянии  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ . Затем измеряет состояния в базисе Белла своего кубита и запутанного кубита из пары в состоянии Белла, предоставленной Чарли. Затем отправляет результат измерения классическим способом Бобу.

3. На данном этапе для кубита Боба имеется апостериорное состояние как функция измерения, выполненного Алисой. Это ключ к пониманию процедуры (помните, что у обоих есть по запутанному кубиту). Таким образом, мы увидим, как Боб, применяя соответствующий квантовый вентиль, может восстановить исходное состояние, созданное Алисой.

Разберемся с этим, рассмотрев апостериорное состояние кубита Боба в момент измерения, выполненного Алисой перед операцией восстановления. Для этого запишем объединенные состояния трех кубитов, участвующих в процессе. Обратите внимание, что для простоты скобочная нотация не приводится.

Таким образом, если мы объединим кубит Алисы, учитывая его состояние  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ , с одним из запутанных кубитов, взятым из пары в состоянии Белла, предоставленной Чарли, то получим:

$$(\alpha|0\rangle + \beta|1\rangle)(|00\rangle + |11\rangle) = \alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle \quad (5.1)$$

Теперь нужно записать состояние первых двух кубитов с помощью состояний в базисе Белла.

$$\begin{aligned} |B0\rangle &= (|00\rangle + |11\rangle) / \sqrt{2} & |00\rangle &= (|B0\rangle - |B1\rangle) / \sqrt{2} \\ |B1\rangle &= (|10\rangle + |01\rangle) / \sqrt{2} & |01\rangle &= (|B1\rangle - |B3\rangle) / \sqrt{2} \\ |B2\rangle &= (|00\rangle - |11\rangle) / \sqrt{2} & |10\rangle &= (|B1\rangle - |B3\rangle) / \sqrt{2} \\ |B3\rangle &= (|10\rangle - |01\rangle) / \sqrt{2} & |11\rangle &= (|B0\rangle - |B2\rangle) / \sqrt{2} \end{aligned}$$

Выражение (5.1) приводится к виду

$$\begin{aligned} (\alpha|0\rangle + \beta|1\rangle)(|00\rangle + |11\rangle) &= |B0\rangle(\alpha|0\rangle + \beta|1\rangle) + |B1\rangle(\alpha|1\rangle + \beta|0\rangle) + \\ &+ |B2\rangle(\alpha|0\rangle - \beta|1\rangle) + |B3\rangle(-\alpha|1\rangle + \beta|0\rangle) \end{aligned} \quad (5.2)$$

Выражение (5.2) показывает состояния трех кубитов после того, как Алиса выполнит свои измерения. Боб знает, как восстановить состояние кубита Алисы  $\psi$ , рассмотрев апостериорное состояние кубита в выражении (5.2) (состояния в скобках). Более наглядно это показано в табл. 5.5.

**Таблица 5.5.** Восстановление информации при квантовой телепортации

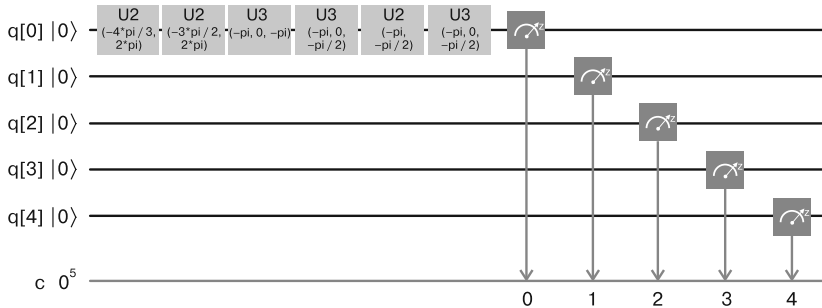
Состояние Белла	Апостериорное состояние	Операция восстановления информации на стороне Боба
B0	$\alpha 0 + \beta 1$	$\psi$
B1	$\alpha 1 + \beta 0$	$X\psi$
B2	$\alpha 0 - \beta 1$	$Z\psi$
B3	$-\alpha 1 + \beta 0$	$ZX\psi$

В целом протокол квантовой телепортации предоставляет средства для восстановления состояния  $\psi$  любого кубита путем разделения запутанной пары в состоянии Белла между двумя удаленными сторонами. Отсюда и название «телепортация». Теперь давайте создадим схему для этого протокола, запустим ее на моделирующем устройстве и, наконец, рассмотрим результаты.

## Схема в Composer

На рис. 5.9 показаны схема в Composer, а также результаты выполнения (только для моделирующего устройства, на этот раз версии для реального устройства нет) для протокола квантовой телепортации.

- Вентиль слева от разделительного символа (пунктирная линия) представляет пару кубитов в состоянии Белла, подготовленную третьей стороной (Чарли): первый и второй кубиты.
- Алиса готовит свой кубит (0) к заданному состоянию  $\psi$ . Фактическое значение  $\psi$  не имеет значения, так как оно будет восстановлено Бобом на заключительной стадии процесса. Алиса получает qubit[1] от Чарли, а qubit[2] отправляется к Бобу.
- Алиса выполняет измерение своих кубитов [0, 1] (показано справа от пунктирной линии) и классическим способом отправляет результаты Бобу.
- Боб применяет к своему кубиту (2) правила восстановления, упомянутые в предыдущем разделе, в зависимости от результатов, отправленных Алисой. Наконец, после измерения qubit[2] Боб восстанавливает состояние, изначально созданное Алисой. Все это стало возможным благодаря тому, что Алиса и Боб имеют запутанную пару кубитов, которая заставляет все это работать.



### Teleportation

Device: Simulator

Quantum State: Computation Basis

Download CSV

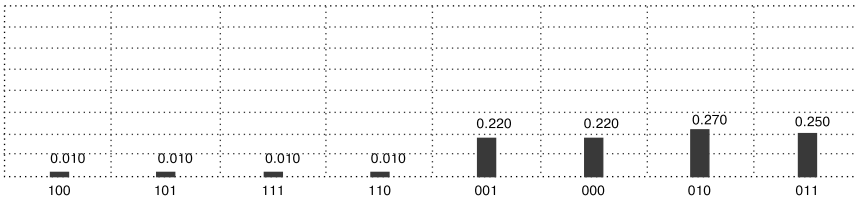


Рис. 5.9. Схема для квантовой телепортации в Composer

Конечно, результаты выполнения, показанные на рис. 5.9, нужно обработать, чтобы убедиться, что полученное Бобом значение  $\psi$  соответствует созданному Алисой. Лучший способ сделать это — воспользоваться сценарием Python. В следующем разделе мы запустим ту же самую схему удаленно и посмотрим на результаты, чтобы убедиться, что протокол работает.

## Удаленный запуск с помощью Python

В этом разделе мы используем Python для удаленного запуска протокола квантовой телепортации на моделирующем устройстве. Обратите внимание, что на данный момент квантовую телепортацию нельзя запустить на реальном квантовом устройстве в IBM Q Experience. Это связано с тем, что аппаратное обеспечение не поддерживает вентили физического вращения, необходимые Алисе для создания своего состояния  $\psi$ . В связи с этим мы будем применять удаленное моделирующее устройство — локальное

моделирующее устройство на Python тоже подойдет. В листинге 5.3 протокол показан в действии. В частности:

- созданы три кубита для совместного использования обеими сторонами, Алисой и Бобом, а также три классических регистра (c0, c1, c2) для хранения результатов Алисы (строки 20–23);
- Чарли подготавливает пару кубитов в состоянии Белла, применяя вентиль Адамара (H), за которым следует контролируемое НЕ (CNOT) на первом и втором кубитах (строки 27–29);
- Алиса подготавливает свое состояние  $\psi$  в нулевом кубите, выполняя поворот вокруг оси Y на  $\pi/4$  радиан (строка 33);
- Алиса теперь запутывает свой кубит(0) с кубитом(1) из пары в состоянии Белла, переданным ей. Затем выполняет измерение на обоих кубитах и сохраняет результаты в нулевом и первом классических регистрах (строки 37–43);
- теперь очередь Боба — он применяет вентили Z или X к своему кубиту (2) в зависимости от результатов, отправленных Алисой. Если значение нулевого классического регистра соответствует 1, то применяет вентиль Z. Если первый классический регистр имеет значение 1, то применяет вентиль X. Затем он измеряет свой кубит и сохраняет результат во втором классическом регистре (строки 50–53);
- программа выполняется на удаленном моделирующем устройстве (ibmq\_qasm\_simulator), и результаты накапливаются для вывода и проверки (строки 62–84).

---

#### ПРИМЕЧАНИЕ

Код этой программы включен в исходники для данной книги и размещен в файле `Workspace\Ch05\p05-teleport.py`.

---

#### Листинг 5.3. Сценарий Python для квантовой телепортации

```
import sys,time,math
import numpy as np

# Импорт QISKit
from qiskit import QuantumCircuit, QuantumProgram
```

```

# Конфигурация Q Experience
sys.path.append('../Config/')
import Qconfig

# Импорт основных средств вывода графиков
from qiskit.tools.visualization import plot_histogram

def main():
    # Первоначальная настройка квантовой программы
    Q_program = QuantumProgram()
    Q_program.register(Qconfig.APIToken, Qconfig.config["url"])

    # Создание регистров
    q = Q_program.create_quantum_register('q', 3)
    c0 = Q_program.create_classical_register('c0', 1)
    c1 = Q_program.create_classical_register('c1', 1)
    c2 = Q_program.create_classical_register('c2', 1)

    # Квантовая схема для создания разделенного запутанного состояния
    # (пара кубитов в состоянии Белла)
    teleport = Q_program.create_circuit('teleport', [q], [c0, c1, c2])
    teleport.h(q[1])
    teleport.cx(q[1], q[2])

    # Алиса подготавливает свое квантовое состояние перед телепортацией
    #  $\psi = a|0\rangle + b|1\rangle$  где  $a = \cos(\theta/2)$ ,  $b = \sin(\theta/2)$ ,
    #  $\theta = \pi/4$ 
    teleport.ry(np.pi/4, q[0])

    # Алиса применяет вентиль CNOT к двум своим квантовым состояниям,
    # а затем вентиль H, чтобы перепутать их
    teleport.cx(q[0], q[1])
    teleport.h(q[0])
    teleport.barrier()

    # Алиса измеряет два своих квантовых состояния:
    teleport.measure(q[0], c0[0])
    teleport.measure(q[1], c1[0])

    circuits = ['teleport']
    print(Q_program.get_qasms(circuits)[0])

    ##### Боб в зависимости от результата применяет вентили X или Z
    # или оба этих вентиля к своему состоянию
    teleport.z(q[2]).c_if(c0, 1)
    teleport.x(q[2]).c_if(c1, 1)

    teleport.measure(q[2], c2[0])

    # Вывод кода ассемблера
    circuits = ['teleport']
    print(Q_program.get_qasms(circuits)[0])

```

```

# Запуск на моделирующем устройстве
# (реальные устройства еще не поддерживают эту процедуру)
#backend = "local_qasm_simulator"
backend = "ibmq_qasm_simulator"
shots = 1024 # количество запусков эксперимента

result = Q_program.execute(circuits, backend=backend, shots=shots
    , max_credits=3, timeout=240)

print("Counts:" + str(result.get_counts("teleport")))

# РЕЗУЛЬТАТЫ
# Измерения Алисы:
data = result.get_counts('teleport')
alice = {}
alice['00'] = data['0 0 0'] + data['1 0 0']
alice['10'] = data['0 1 0'] + data['1 1 0']
alice['01'] = data['0 0 1'] + data['1 0 1']
alice['11'] = data['0 1 1'] + data['1 1 1']
plot_histogram(alice)

#Боб
bob = {}
bob['0'] = data['0 0 0'] + data['0 1 0'] + data['0 0 1'] + data['0 1 1']
bob['1'] = data['1 0 0'] + data['1 1 0'] + data['1 0 1'] + data['1 1 1']
plot_histogram(bob)

#####
# main
if __name__ == '__main__':
    start_time = time.time()
    main()
    print("--- %s seconds ---" % (time.time() - start_time))

```

Для проверки нужно собрать статистику появления результатов, полученных Алисой и Бобом и возвращаемых моделирующим устройством. Вывести график результатов — лучший способ убедиться в том, что состояние Алисы  $\psi$  было восстановлено Бобом. Вот пример того, что возвращает моделирующее устройство:

```

{'1 0 0': 37, '1 0 1': 45, '1 1 1': 43, '0 1 1': 215, '0 0 1': 200,
 '0 0 0': 206, '0 1 0': 230, '1 1 0': 48}

```

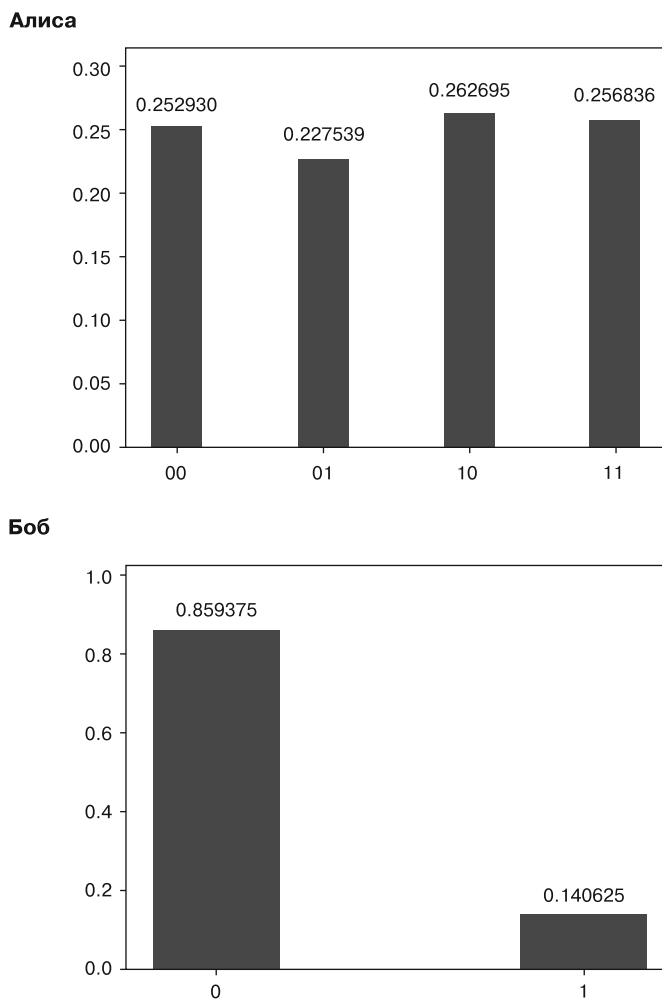
В этой строке в формате JSON слева приведены результаты для трех кубитов в обратном порядке. Например, в первом результате 1 0 0: В (1) А (0) А (0) для Алисы (А) и Боба (В). Справа приведено количество появлений данного результата. Помните, что вероятность этого выходного значения



(используется для построения графиков) рассчитывается делением количества его появлений на общее количество запусков (1024). Таким образом:

$$P(1\ 0\ 0) = 37/1024 = 0,036$$

Гистограммы для результатов Алисы и Боба, полученных при выполнении программы из листинга 5.3, показаны на рис. 5.10.



**Рис. 5.10.** Результирующие вероятности для измерений, выполненных Алисой и Бобом

Что же все это значит? И как мы узнаем, что состояние  $\psi$  было восстановлено Бобом? Рассмотрим результаты подробнее.

## Результаты

Чтобы интерпретировать результаты, сначала посмотрим, как рассчитываются вероятности по числовым значениям, полученным для листинга 5.3:

```
{'1 0 0': 37, '1 0 1': 45, '1 1 1': 43, '0 1 1': 215, '0 0 1': 200,
'0 0 0': 206, '0 1 0': 230, '1 1 0': 48}
```

Используя эти числа, мы можем рассчитать вероятности появления результатов для Алисы и Боба, показанные на рис. 5.10 (табл. 5.6).

**Таблица 5.6.** Результирующие вероятности для эксперимента по квантовой телепортации

Строка	Алиса	Боб	Результат	Счетчик появлений результата	Вероятность	Алиса	Суммарная вероятность
0	Алиса(00)	Боб(0)	0 0 0	206	0,201171875	0 0	0,237304688
1	Алиса(01)	Боб(0)	0 0 1	200	0,195312500	1 0	0,239257813
2	Алиса(10)	Боб(0)	0 1 0	230	0,224609375	0 1	0,271484375
3	Алиса(11)	Боб(0)	0 1 1	215	0,209960938	1 1	0,251953125
4	Алиса(00)	Боб(1)	1 0 0	37	0,036132813		
5	Алиса(01)	Боб(1)	1 0 1	45	0,043945313	<b>Боб</b>	
6	Алиса(10)	Боб(1)	1 1 0	48	0,046875000	0	0,831054688
7	Алиса(11)	Боб(1)	1 1 1	43	0,041992188	1	0,168945313

Как показано в табл. 5.6, для вычисления полной вероятности результата Алисы 00 нужно суммировать вероятности в нулевой и четвертой строках. Таким образом,  $P(A00) = 0,201 + 0,036 = 0,237$ . То же самое правило применяется к данным Боба. Например,  $P(B0) = 0,20 + 0,19 + 0,22 + 0,20 = 0,83$  (вероятности в строках с нулевой по третью). Это то, что показано с правой стороны для всех результатов Алисы и Боба. Таким образом, сценарий в листинге 5.3 обрабатывает данные перед построением графиков результатов, приведенных на рис. 5.10. Но что все это означает и как мы узнаем,

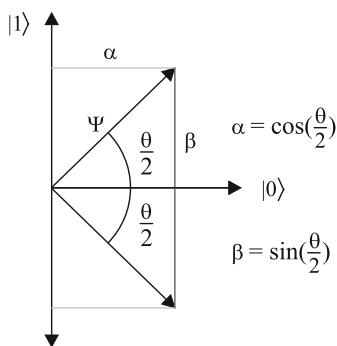
что Боб восстановил  $\psi$  Алисы? Взглянем на полную вероятность, полученную Бобом для его кубита:

Боб	
0	$0,20 + 0,19 + 0,22 + 0,20 = 0,83$
1	$0,036 + 0,043 + 0,046 + 0,041 = 0,168$

Согласно принципам квантовой механики вероятность  $\psi$  определяется по формуле  $P(\psi) = |\psi|^2$ . Таким образом, плотность вероятности — это квадрат модуля  $\psi$ . Теперь вспомним, что Алиса подготовила  $\psi$  как

$$\psi = RY(\theta), \text{ где } \theta = \frac{\pi}{4}.$$

То есть Алиса применила к своему кубиту поворот на  $\pi/4$  радиан вокруг оси  $Y$ . Чтобы сделать это более наглядным, представим состояние  $\psi$  геометрически (рис. 5.11).



**Рис. 5.11.** Полученное в результате суперпозиции состояние для  $\psi$  Алисы

Помните, что полученное в результате суперпозиции состояние  $\psi$  описывается комплексными коэффициентами  $\alpha$  и  $\beta$  как:

$$\psi = \alpha |0\rangle + \beta |1\rangle.$$

$$\text{Вероятность } (0) = |\alpha|^2, \text{ вероятность } (1) = |\beta|^2.$$

Но, исходя из рис. 5.11, данные коэффициенты можно представить как  $\alpha = \cos(\theta/2)$  и  $\beta = \sin(\theta/2)$ . Тогда, наконец, если  $\theta = \pi/4$ , то:

$$\bigcirc |\alpha|^2 = |\cos(\pi/8)|^2 = 0,85;$$

$$\bigcirc |\beta|^2 = |\sin(\pi/8)|^2 = 0,14.$$

Это соответствует результатам Боба, показанным на графике, созданном программой для телепортации из листинга 5.3 (рис. 5.12). Это большой успех!

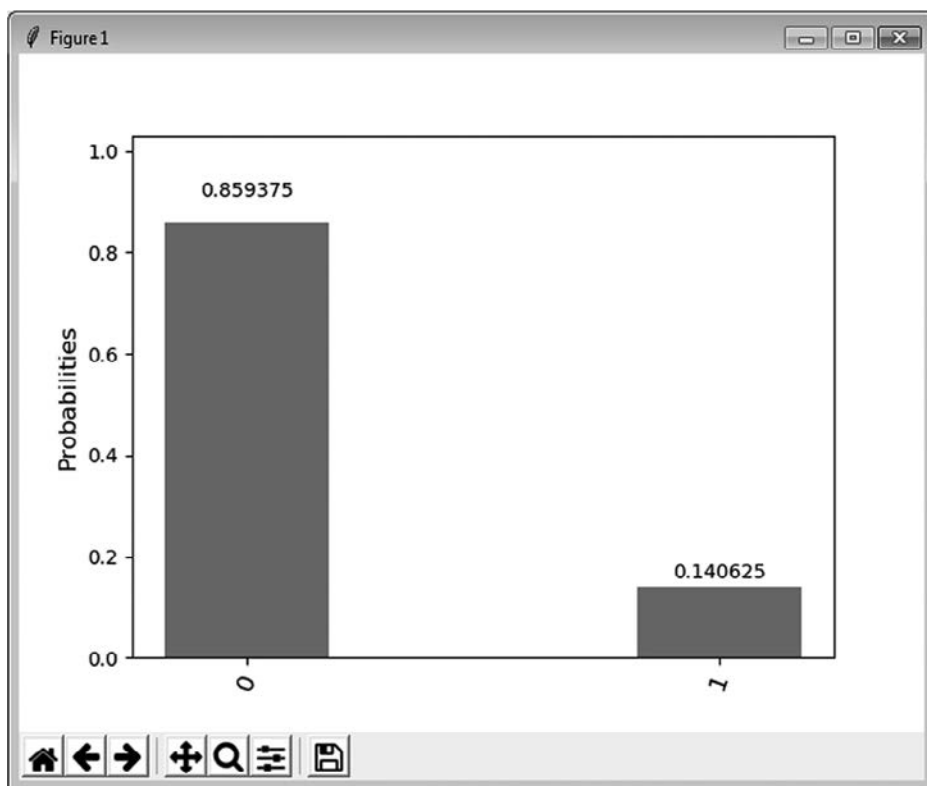


Рис. 5.12. Результаты телепортации для Боба

Вы сделали первый шаг к пониманию замечательных возможностей квантовых систем обработки информации. Мы начали с простой процедуры, где используется источник истинной случайности, свойственной кван-

товой механике, для генерации случайных чисел. Кроме того, разобрали два странных протокола — сверхплотное кодирование для кодирования классической информации и квантовую телепортацию для восстановления состояния кубита удаленной стороной. Эти протоколы были описаны с применением схем для IBM Q Experience, а также сценариев Python для удаленного выполнения на моделирующем или реальном квантовом устройстве. Чтобы вам было проще понять все происходящее, мы собирали результаты и объясняли их.

В следующей главе рассмотрим более легкую тему из области квантовых вычислений, а точнее, создадим простую игру с применением квантовых вентилях. Это необходимая передышка перед тем, как перейти к трудному для понимания материалу в последующих главах.

# 6

## Развлекаемся квантовыми играми

Здесь вы узнаете, как реализовать простую игру на квантовом компьютере. Для этой цели мы воспользуемся классической игрой Quantum Battleship (квантовый «Морской бой»), поставляемой вместе с учебником по QISKit на Python. В первой части рассматривается механика игры, включая:

- использование кубитов для представления позиций кораблей на игровом поле;
- расчет возможного ущерба с применением квантовой программы для запуска на локальном, удаленном моделирующем или реальном квантовом устройстве;
- способ поворота вокруг оси  $X$  одного кубита с помощью частичного квантового вентиля НЕ.

И мы не остановимся на этом. Во второй части главы выйдем на следующий уровень. Игра будет значительно модифицирована за счет реализации облачной версии Quantum Battleship со следующими характеристиками.

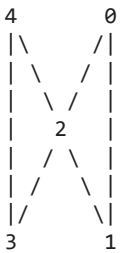
- Интерфейс пользователя с браузерной поддержкой и интерактивными игровыми полями для размещения кораблей и бомб. Механика игры остается прежней.

- Сценарий на основе CGI, использующий веб-сервер Apache для получения игровых событий и отправки их в квантовую программу.
- Модифицированная версия исходной квантовой программы для выполнения частичных поворотов при помощи вентиля НЕ на кубитах для расчета ущерба, нанесенного кораблям. Большая часть исходного кода остается неизменной.

Вы узнаете, как исходный код можно сделать модульным, чтобы повторно применять для другой версии Quantum Battleship в облаке. Приступим!

## Quantum Battleship с изюминкой

В этом разделе рассмотрим игру под названием *Quantum Battleship*, распространяемую вместе с руководством по QISKit. В программе используются пять кубитов для представления игрового поля, где каждый игрок размещает три корабля. Затем каждого игрока просят разместить бомбу в позиции 0–5. Ущерб для каждого корабля рассчитывается с помощью квантовой программы, которая применяет однобитовый двухимпульсный вентиль:  $U_3(\theta, \phi, \lambda)$ . Он называется частичным вентилем НЕ и выполняет поворот вокруг осей  $X$ ,  $Y$  или  $Z$  на  $\theta$ ,  $\phi$  или  $\lambda$  радиан.



В данном случае ущерб, нанесенный кораблю, рассчитывается путем выполнения серии частичных поворотов вокруг оси  $X$  ( $\theta$ ) с учетом количества бомб для этой позиции. Если урон для позиции или корабля превышает 95 %, корабль уничтожен, а после разгрома всего флота одного из игроков объявляется победитель и игра заканчивается. Это самый

обычный «Морской бой», в который мы все играли в детстве, но с квантовым компьютером или моделирующим устройством на заднем плане.

---

#### ПРИМЕЧАНИЕ

Игру написал Джеймс Вуттон из Базельского университета. Она внесена в учебник по QISKit на Python. Модифицированная версия оригинального кода Вуттона доступна в исходниках для этой книги по адресу `Workspace\Ch06\battleship\BattleShip.py` (за исключением ненужного текста с причудливым форматированием).

---

Запустим программу и рассмотрим механику игры.

## Инструкции по настройке

Запустите на выполнение программу `BattleShip.py` из исходников к этой книге, как описано далее.

1. Для CentOS 6 или 7 или любой ОС, подобной Fedora, активируйте виртуальную среду Python. Это необходимо, только если у вас есть несколько версий Python, например 2.7 и 3.6. Помните, что вы должны использовать версию 3.5 или более позднюю. Инструкции по настройке виртуальной среды Python приведены в главе 3.
2. Скопируйте сценарий из `Workspace\Ch06\battleship\BattleShip.py` и файл конфигурации `Qconfig.py` из исходников книги в свое рабочее пространство и запустите его на выполнение, как показано в следующем фрагменте кода:

```
# Активация виртуальной среды Python3 на $HOME/qiskit/qiskit
$ source $HOME/qiskit/qiskit/bin/activate
$ python BattleShip.py
##### Quantum Battle Ship #####
Do you want to play on the real device? (y/n) n
```

Рассмотрим работу программы.



## Инициализация

В листинге 6.1 демонстрируется инициализация сценария. Она начинается с выполнения основных задач Python.

- Загружаются системные библиотеки `sys` и `QuantumProgram`, необходимые для всех операций QISKit.
- Проверяется то, что вы используете Python 3.5 или более позднюю версию.
- Программа спрашивает, хотите ли вы задействовать моделирующее устройство или настоящий квантовый компьютер. Затем устанавливается количество запусков на выполнение, по умолчанию 1024.

### Листинг 6.1. Инициализация сценария

```
#####
# Quantum Battleship из учебника @
# https://github.com/QISKit/qiskit-tutorial
##### import sys
# проверка версии PYTHON; поддерживаются только версии > 3.5
if sys.version_info < (3,5):
    raise Exception('Please use Python version 3.5 or greater.')

from qiskit import QuantumProgram
import Qconfig
import getpass, random, numpy, math

## 1. Выбор серверного ПО: моделирующее устройство IBM
(ibmqx_qasm_simulator) или реальный чип ibmqx2

d = input("Do you want to play on the real device? (y/n)\n").upper()
if (d=="Y"):
    device = 'ibmqx2'
else:
    device = 'ibmqx_qasm_simulator'

# Обратите внимание, что устройством может быть 'ibmqx_qasm_simulator',
# 'ibmqx2' или 'local_qasm_simulator'
# Установка количества запусков
shots = 1024
```

**СОВЕТ**

Чтобы запустить квантовую программу на реальном устройстве, вы должны поместить файл конфигурации (Qconfig.py) в то же место, что и основной сценарий. Конфигурация содержит требуемый токен API и конечную точку IBM Q Experience:

```
APIToken = 'YOUR API TOKEN'
config = {
    'url': 'https://quantumexperience.ng.bluemix.net/api',
}
```

---

Теперь расставим несколько кораблей на игровом поле.

## Размещение кораблей на игровом поле

В программе используется элементарный текстовый интерфейс для всего пользовательского ввода. В листинге 6.2 показана логика ввода кораблей для каждого игрока. Нажмите **Enter**, чтобы начать, и введите позиции трех кораблей для каждого игрока (нумерация позиций с нуля).

- Сценарий может сам выбрать случайные позиции либо игрок должен ввести позиции для трех кораблей.
- Позиции хранятся в двумерном списке `shipPos`, где `shipPos[0]` содержит позиции первого игрока, а `shipPos[1]` — второго. Помните, что для одного игрока доступно только три корабля.

### Листинг 6.2. Размещение кораблей на игровом поле

```
##### 2. Размещение кораблей на игровом поле игроком
randPlace = input("> Press Enter to start placing ships...\n").upper()

# В переменной ship[X][Y] будет храниться позиция Y-го корабля игрока X+1
shipPos = [ [-1]*3 for _ in range(2)]

# Проход в цикле по обоим игрокам и трем кораблям для каждого
for player in [0,1]:
```

```

# Если выбор производится не игроком, а случайным образом
if ((randPlace=="r")|(randPlace=="R")):
    randPos = random.sample(range(5), 3)
    for ship in [0,1,2]:
        shipPos[player][ship] = randPos[ship]
else:
    for ship in [0,1,2]:

# Запрос позиции для каждого корабля
choosing = True
while (choosing):

    # Получить ввод игрока
    position = getpass.getpass("Player " + str(player+1)
        + ", choose a position for ship " + str(ship+1) +
        " (0-4)\ n" )

    # Проверка корректности ввода и запрос на повторный ввод
    if position.isdigit(): # Ответ должен быть целым числом
        position = int(position)
        # и находиться в диапазоне от 0 до 4
        if (position in [0,1,2,3,4]) and (not position in
            shipPos[player]):
            shipPos[player][ship] = position
            choosing = False
            print ("\n")
        elif position in shipPos[player]:
            print ("\nYou already have a ship there. Try
                again.\n")
        else:
            print ("\nThat's not a valid position. Try again.\n")
    else:
        print ("\nThat's not a valid position. Try again.\n")

```

Далее показан стандартный вывод, очень простой, но пока этого достаточно:

```

Do you want to play on the real device? (y/n) n
Player 1, choose a position for ship 1 (0, 1, 2, 3 or 4)
0
Player 1, choose a position for ship 2 (0, 1, 2, 3 or 4)
1
Player 1, choose a position for ship 3 (0, 1, 2, 3 or 4)
2
Player 2, choose a position for ship 1 (0, 1, 2, 3 or 4)
0

```

Player 2, choose a position for ship 2 (0, 1, 2, 3 or 4)

1

Player 2, choose a position for ship 3 (0, 1, 2, 3 or 4)

2

Интересные процессы происходят в основном цикле. Рассмотрим его.

## Основной цикл и результаты

В основном цикле выполняются следующие задания.

- Обоих игроков просят поместить по одной бомбе в позицию [0–4]. Количество бомб хранится в двумерном списке из пяти элементов (два игрока, пять счетчиков бомб). Обратите внимание, что игрок может бомбить одну и ту же позицию несколько раз; таким образом, если первый игрок наносит удар по нулевой позиции дважды, то `bombs = [[2,0,0,0,0], [0,0,0,0,0]]`.
- Создается `QuantumProgram` для хранения пяти кубитов (по одному на каждую позицию на доске) и пяти классических регистров, чтобы хранить результаты измерений.
- Если позиция бомбы совпадает с позицией корабля противника (из списка `shipPos`), ущерб рассчитывается выполнением одного поворота вокруг оси  $X$  по количеству бомб с использованием частичного однокубитного вентиля `HE`: `gridScript.u3(1/(ship+1) * math.pi, 0.0, 0.0, q[position])`. Обратите внимание, что эффективность бомбы также зависит от того, какой корабль бомбят (0, 1, 2).
- Для завершения схемы на кубите выполняется измерение для позиции, а результат сохраняется в соответствующем классическом регистре: `gridScript.measure(q[position], c[position])`.
- Затем программа выполняется на целевом устройстве, а результаты сохраняются в двумерном списке `grid`. Например, если нулевая позиция первого игрока разбомблена, то `grid = [[1,0,0,0,0], [0,0,0,0,0]]`. Далее показано, как это происходит:

```
results = Q_program.execute(["gridScript"], backend=device,
                             shots=shots)
grid[player] = results.get_counts("gridScript")
```

- Результаты проверяются на наличие ошибок. Если все верно, то процент ущерба рассчитывается в промежутке  $[0, 1]$ , когда в списке `grid` содержится значение 1 для этой позиции. Проценты сохраняются в двумерном списке повреждений. Таким образом, ущерб `[[0, 95, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]` указывает на то, что корабль первого игрока на нулевой позиции был уничтожен.
- Наконец, игрокам выводятся результаты. Процесс повторяется до тех пор, пока все корабли не будут уничтожены и не будет объявлен победитель (листинг 6.3).

### Листинг 6.3. Основной цикл игры «Морской бой»

```
?      100 %
| \    / |
| \    / |
|  ?  /  |
| /    \ |
| /    \ |
|  ?  \  |
| \    / |
| \    / |
?      100 %
```

```
##### 3. Основной цикл
# Каждая итерация начинается с опроса игроков о том, где
# на игровом поле противника должна быть размещена бомба
# Квантовый компьютер подсчитывает эффект от нанесенных ударов
# и предоставляет результаты
# Игра продолжается, пока все корабли одного из игроков
# не будут уничтожены
game = True
# В переменной bombs[X][Y] хранится количество раз, когда позиция Y
# была разбомблена игроком X+1
bomb = [ [0]*5 for _ in range(2)] # все значения инициализированы нулями

# В переменной grid[player] будут храниться результаты для каждого игрока
grid = [{},{}]

while (game):
    input("> Press Enter to place some bombs...\n")

    # Обоих игроков спрашивают, где они хотят разместить бомбы
    for player in range(2):

        print("\n\nIt's now Player " + str(player+1) + "'s turn.\n")

        # Вопрос повторяется, пока не будет получено корректное значение
        choosing = True
        while (choosing):
```

```

# Получить ввод пользователя
position = input("Choose a position to bomb (0, 1, 2, 3
or 4)\n")

# Проверка корректности ввода
# Запрос на повторный ввод, если предыдущий был некорректным
if position.isdigit(): # ответ должен быть целым числом
    position = int(position)
    if position in range(5):
        bomb[player][position] = bomb[player][position] + 1
        choosing = False
        print ("\n")
    else:
        print("\nThat's not a valid position. Try again.\n")
else:
    print("\nThat's not a valid position. Try again.\n")

# Теперь мы создаем и запускаем на выполнение квантовую программу
# для каждого игрока
for player in range(2):
    if device=='ibmqx2':
        print("\nUsing a quantum computer for Player " + str(player+1)
+ "'s ships.\n")
    else:
        print("\nUsing the simulator for Player " + str(player+1) +
"'s ships.\n")

# Теперь настройка квантовой программы (QASM) для моделирования
# сетки для данного игрока

Q_program = QuantumProgram()
# Установка APIToken и url-адреса API
Q_program.set_api(Qconfig.APIToken, Qconfig.config["url"])
# Объявление регистров пяти кубитов
q = Q_program.create_quantum_register("q", 5)
# Объявление регистров пяти классических битов для хранения
# результатов измерений
c = Q_program.create_classical_register("c", 5)
# Создание схемы
gridScript = Q_program.create_circuit("gridScript", [q], [c])

# Добавление бомб (противника)
for position in range(5):
    # Добавить столько бомб, сколько было помещено в данную позицию
    for n in range( bomb[(player+1)%2][position] ):
        # Эффективность бомбы
        # (квантовой операции, которую мы применим)
        # зависит от вида корабля
        for ship in [0,1,2]:

```

```

        if ( position == shipPos[player][ship] ):
            frac = 1/(ship+1)
            # Добавление этой части вентиля НЕ в QASM
            gridScript.u3(frac * math.pi, 0.0, 0.0, q[position])

# Наконец, выполнение измерений
for position in range(5):
    gridScript.measure(q[position], c[position])

# чтобы увидеть, какие действия должен выполнить квантовый
# компьютер, мы можем вывести на печать файл с QASM
# Эта строка обычно закомментирована
#print( Q_program.get_qasm("gridScript") )

# Скомпилировать и запустить на выполнение QASM
results = Q_program.execute(["gridScript"], backend=device,
shots=shots)

# Извлечение данных
grid[player] = results.get_counts("gridScript")

# Данные при желании можно проверить
# Эти строки обычно закомментированы
#print( grid[0] )
#print( grid[1] )

# Если один из запусков потерпел неудачу, сообщить об этом игрокам
# и начать раунд заново
if ( ( 'Error' in grid[0].values() ) or ( 'Error' in grid[1].
Values() ) ):

    print("\nThe process timed out. Try this round again.\n")

else:

    # Рассмотрим ущерб во всех кубитах (даже там, где нет кораблей)
    # Для каждого кубита каждого игрока будет сохранено значение
    # вероятности 1
    damage = [ [0]*5 for _ in range(2)]

    # Для этого мы пройдем в цикле по всем пяти битовым строкам
    # каждого из игроков
    for player in range(2):
        for bitString in grid[player].keys():
            # А затем по всем позициям
            for position in range(5):
                # Если в строке для данной позиции находится значение 1,
                # то распределение вероятности добавляется в ущерб
                # Помните, что бит нулевой позиции крайний справа

```

```

        # и соответствует bitString[4]
        if (bitString[4-position]=="1"):
            damage[player][position] += grid[player]
            [bitString]/shots
# Вывод результатов игрокам
for player in [0,1]:

    input("\nPress Enter to see the results for Player
    " + str(player+1) + "'s ships...\n")

# Отчет о существенном ущербе по кубитам для кораблей;
# в идеале это будет ненулевой ущерб,
# таким образом, мы выбираем пороговое значение 5 %
display = [" ? "]*5
# Перебрать в цикле кубиты для кораблей
for position in shipPos[player]:
    # Если ущерб довольно велик, отобразить его
    if ( damage[player][position] > 0.1 ):
        if (damage[player][position]>0.9):
            display[position] = "100 %"
        else:
            display[position] = str(int( 100*damage[player]
            [position] )) + "% "

print("Here is the percentage damage for ships that have been
bombed.\n")
print(display[ 4 ] + " " + display[ 0 ])
print(" | \      / |")
print(" | \    / |")
print(" | \ / |")
print(" | " + display[ 2 ] + " |")
print(" | / \ |")
print(" | /    \ |")
print(" | /      \ |")
print(display[ 3 ] + " " + display[ 1 ])
print("\n")
print("Ships with 95 % damage or more have been destroyed\n")

print("\n")

# Если все корабли одного игрока уничтожены, игра окончена
# В идеале это предполагает 100%-ный ущерб, но мы остановимся
# на 90 % опять же по причине шума
if (damage[player][ shipPos[player][0] ]>.9) and
(damage[player][ shipPos[player][1] ]>.9)
and (damage[player][ shipPos[player][2] ]>.9):
    print ("***All Player " + str(player+1) + "'s ships have
    been destroyed!***\n\n")
    game = False

```



```

if (game is False):
    print("")
    print("=====GAME OVER=====")
    print("")

```

Обратите внимание, что если ущерб превысит 90 %, то корабль будет помечен как уничтоженный. В листинге 6.4 показаны результаты одного игрового взаимодействия.

#### **Листинг 6.4.** Стандартный вывод для одного игрового взаимодействия

```
> Press Enter to place some bombs...
```

```
It's now Player 1's turn.
Choose a position to bomb (0, 1, 2, 3 or 4)
```

```
0
```

```
It's now Player 2's turn.
Choose a position to bomb (0, 1, 2, 3 or 4)
```

```
0
```

```
We'll now get the simulator to see what happens to Player 1's ships.
We'll now get the simulator to see what happens to Player 2's ships.
```

```
Press Enter to see the results for Player 1's ships...
Here is the percentage damage for ships that have been bombed.
```

```

?          100 %
| \       / |
|  \     /  |
|   \   /   |
|    \ /    |
|     / \    |
|    /  \   |
|   /    \  |
|  /      \ |
| /        \|
?          ?

```

```
Ships with 95% damage or more have been destroyed
```

```
Press Enter to see the results for Player 2's ships...
Here is the percentage damage for ships that have been bombed.
```

```

?          100 %
| \       / |
|  \     /  |
|   \   /   |
|    \ /    |
|     / \    |
|    /  \   |
|   /    \  |
|  /      \ |
| /        \|
?          ?

```

```
Ships with 95 % damage or more have been destroyed
```

Таким образом, основной цикл продолжается до тех пор, пока не будет объявлен победитель. Итак, вы узнали, как можно реализовать простую игру, чтобы использовать квантовый компьютер для выполнения несложных расчетов нанесенных повреждений путем поворотов кубита вокруг оси  $X$ . Эта версия довольно примитивна, но интересна. Однако мы можем сделать лучше. В следующем разделе усовершенствуем дизайн игры.

## Cloud Battleship: модификация удаленного доступа

Действительно, здорово иметь возможность играть в «Морской бой» на квантовом компьютере через простой текстовый интерфейс, но гораздо круче играть в эту игру в браузере в облаке. В этом разделе мы модифицируем Quantum Battleship и улучшим ее дизайн, в чем игра, несомненно, нуждается (рис. 6.1).

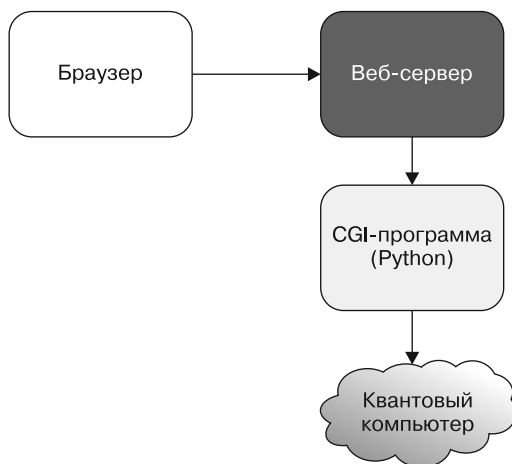


Рис. 6.1. Структура Quantum Battleship в облаке

Идея заключается в следующем.

- Откажитесь от скучного текстового интерфейса в пользу HTML-страницы, которую можно развернуть в облаке.

- Используйте общедоступный шлюзовой интерфейс (Common Gateway Interface, CGI) веб-сервера Apache для развертывания квантовой логики в сценарии.
- Дайте игроку возможность выбрать, на чем выполнять расчеты: на локальном или дистанционном моделирующем устройстве либо на реальном квантовом компьютере.

Реализуем это в серии упражнений, описанных в следующих разделах.

## Упражнение 6.1. Разделение интерфейса пользователя и логики игры

Один из основных принципов объектно-ориентированного программирования таков: никогда не смешивайте представление (интерфейс пользователя) и бизнес-логику. Спроектированные таким образом компоненты можно скомпилировать и повторно использовать где угодно. В случае с Battleship нам нужно выполнить следующее.

- Удалить или закомментировать первый раздел сценария, в котором считывается положение кораблей для каждого игрока (хороший кусок кода), стараясь не затронуть какие-либо структуры данных или переменные.
- Удалить или закомментировать все операторы вывода на печать и ввода с клавиатуры.
- Удалить основной цикл игры, в котором постоянно указывается позиция для бомбардировки. Сценарий должен завершиться после того, как он использует данные из HTTP-запроса. В нем не может быть бесконечных циклов, иначе запрос зависнет.
- Добавить в сценарий поддержку CGI в Python, чтобы из HTTP-запроса можно было прочитать данные, включая:
  - позиции кораблей для каждого игрока;
  - позиции и количество бомб для каждого игрока;
  - устройство для запуска квантовых вычислений.

Сценарий должен возвращать отчет о повреждениях (желательно в формате JSON) через HTTP-ответ, чтобы браузер отображал его в JavaScript.

Обратите внимание, что мы будем повторно использовать большую часть кода: структуры данных, локальные переменные и квантовую логику. В этом заключается причина комментирования всех операторов для ввода данных и вывода на печать. Решение этого (как и всех остальных) упражнения приведено в конце данного раздела.

## Упражнение 6.2. Создание веб-интерфейса для игрового поля

Создайте графический интерфейс пользователя для HTML по аналогии с текстовым, и используйте AJAX для асинхронной отправки запросов в CGI-сценарий. Верните результаты нанесенного ущерба и, наконец, обновите игровые поля. Улучшенный дизайн можно увидеть на рис. 6.2.

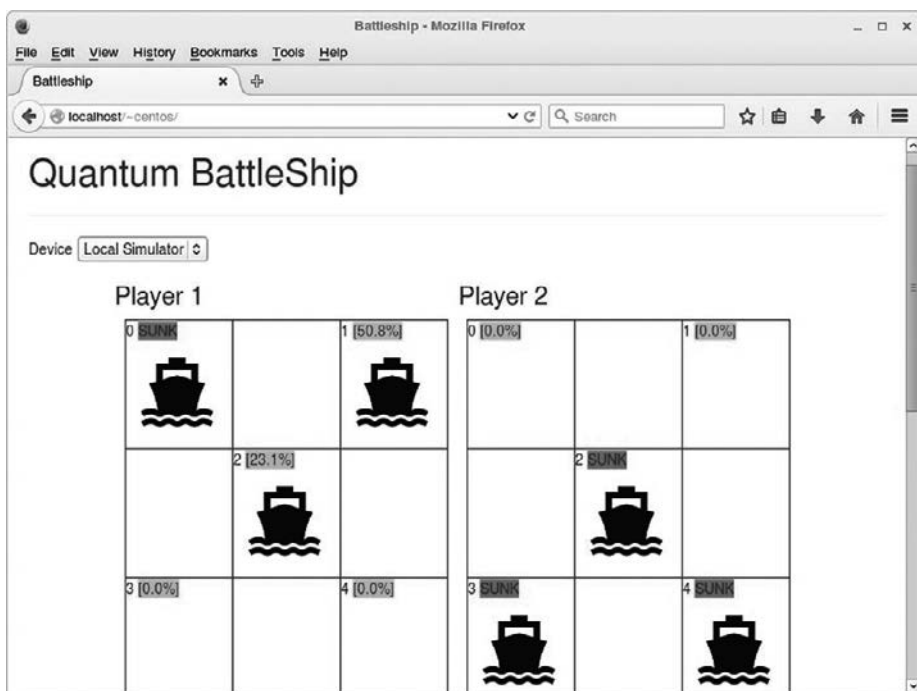


Рис. 6.2. Интерфейс пользователя для новой версии Quantum Battleship

- В HTML-файле будет четыре игровых поля, в каждом  $3 \times 3$  ячейки. Верхние игровые поля используются для того, чтобы разместить три корабля в пяти местоположениях, соответствующих кубитам. Их реализуем в виде флажков (`<INPUT TYPE = "checkbox">`). Мы воспользуемся CSS для замены кнопки-переключателя изображением, поэтому вместо нее при щелчке на ячейке будет переключаться изображение корабля.
- Нижние игровые поля позволят игрокам размещать бомбы в пяти местах при помощи все того же CSS, что и в предыдущем абзаце, но они будут реализованы как `<INPUT TYPE = "radio">`, так что в каждом местоположении может быть размещено несколько бомб.
- Несмотря на то что размер игрового поля  $3 \times 3$ , для пользовательского ввода доступны только пять местоположений, соответствующих каждому кубиту в квантовой программе.
- Для каждого местоположения корабля будут отображаться номер кубита и возвращаемый серверным ПО процент ущерба с цветной подсветкой.
- Механика игры точно такая же, как и в текстовой версии.
- Оба игрока размещают на игровых полях по три корабля, а затем каждый по очереди размещает бомбу и нажимает кнопку **Submit** (Отправить). CGI-сценарий Python получит запрос через AJAX, запустит квантовую программу, созданную в упражнении 6.1, и вернет результат для ущерба, который будет отображен в JavaScript.
- Обратите внимание, что все игровое состояние, массивы, переменные и другие данные хранятся в клиентском HTML, поэтому мы должны использовать AJAX для асинхронной отправки запроса, иначе данные будут потеряны при каждой отправке игроком. Обновлений страницы не будет.

На рис. 6.3 показаны нижние игровые поля  $3 \times 3$  ячейки, отображающие номера кубитов, количество щелчков на бомбе и изображения переключателей, выводимые с использованием CSS. Когда каждый игрок размещает три корабля и выбирает место для бомбардировки, он нажимает кнопку **Submit** (Отправить) для отправки запроса на сервер AJAX. Кнопка сброса также доступна, чтобы игру можно было в любой момент перезапустить.

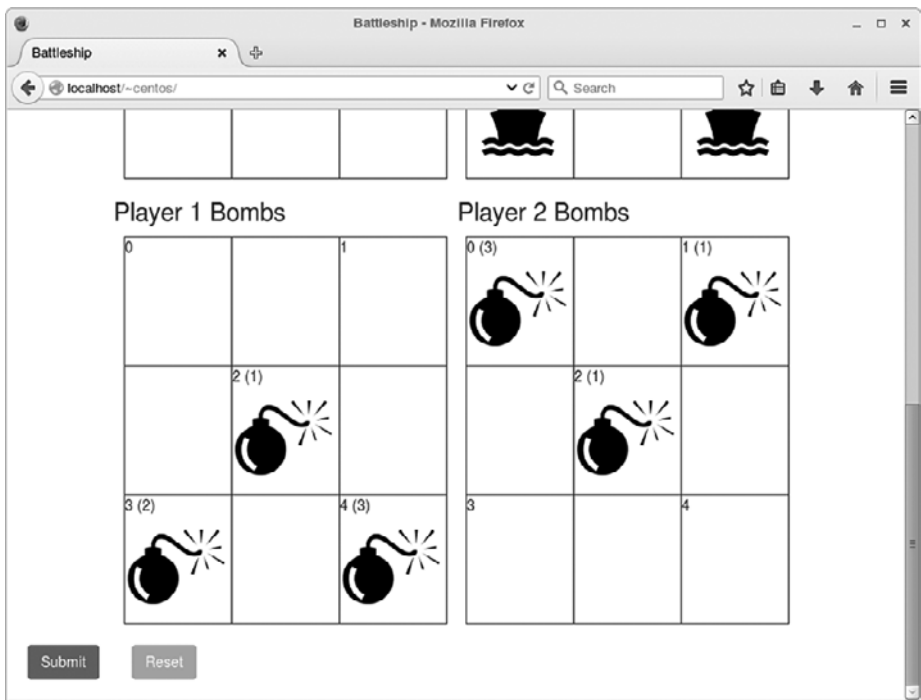


Рис. 6.3. Игровые поля с бомбами в Quantum Battleship

Учтите, что все состояние сохраняется в клиенте (браузере). Никакие данные не будут храниться в сценарии Python, поскольку HTTP является протоколом запроса-ответа без сохранения состояния. Это означает, что, когда запрос получен, программа выполняется веб-сервером, ответ печатается в буфере вывода запроса и программа завершается. Как и для предыдущего упражнения, решение находится в конце этого раздела.

### Упражнение 6.3. Развертывание и устранение неполадок на сервере Apache

Теперь, когда все на месте, пришло время для развертывания на веб-сервере. Я буду использовать сервер Apache под названием HTTPD в CentOS 6, но он должен работать для любой версии CentOS, Fedora или

Red Hat (возможно, для любого текущего дистрибутива Linux с HTTPD). Имейте в виду, что каждый вариант имеет свои особенности при настройке системного ПО. Например, CentOS фокусируется на стабильности и безопасности, что доставляет мне много головной боли при настройке HTTPD и Python.

## Решение 6.1. Программа на Python, позволяющая повторное использование

В этом разделе представлен CGI-скрипт Python, который получает HTTPD-запрос от браузера и отвечает строкой в формате JSON, содержащей отчет о нанесенном ущербе и другую информацию. Первая часть программы осталась практически неизменной, за исключением того, что теперь ввод нужно проанализировать из HTTP-запроса с применением библиотеки `cgi` в Python (листинг 6.5).

### Листинг 6.5. Инициализация модульной версии Quantum Battleship

```
import sys
from qiskit import QuantumProgram
import Qconfig
import getpass, random, numpy, math

import cgi
import cgitb

# Разрешите относительные зависимости, если вы копируете QISKit
# из репозитория Git и используете как глобальный
sys.path.append('../..//qiskit-sdk-py/')

# Отладка
cgitb.enable(display=0, logdir=".")

# В переменной ship[X][Y] будет храниться позиция Y-го корабля игрока X + 1
# все переменные инициализированы -1 (значение для невозможной позиции)
shipPos = [ [-1]*3 for _ in range(2)]

# В переменной bombs[X][Y] будет храниться количество раз, когда позицию Y
# бомбил игрок X+1
bomb = [ [0]*5 for _ in range(2)] # все значения равны 0
```

В листинге 6.5 показана первая часть сценария. В шестой и седьмой строках импортируются библиотеки Python: `cgi` и `cgitb` (CGI Toolbox),

которые используются для чтения данных из HTTP-запросов и отладки CGI-программ соответственно.

---

### ПРИМЕЧАНИЕ

Строки в следующем фрагменте кода активируют специальный обработчик исключений, который будет отображать подробные отчеты в браузере в случае возникновения ошибки.

---

```
import cgitb
cgitb.enable()
```

Имейте в виду, что при возникновении ошибки мы не можем показать внутреннюю информацию программы, так как клиент ожидает ответа в формате JSON. Вместо этого мы должны сохранять отчеты об ошибках в текущем рабочем каталоге при помощи кода, аналогичного следующему:

```
cgitb.enable (display = 0, logdir = ".")
```

Этот код избавит от проблем во время разработки, поскольку любое исключение будет сохранено в аккуратном HTML-документе в текущем рабочем каталоге.

Формат документа показан в разделе «Устранение ошибок» данной главы.

В листинге 6.5 также показаны структуры данных, используемые для хранения игрового состояния. Они те же самые, что и в старой версии:

- `shipPos` — двумерный список, в котором хранятся позиции для трех кораблей на игрока, инициализированные `-1`. Таким образом, `shipPos = [[-1, -1, -1], [-1, -1, -1]]`;
- `bomb` — двумерный список, в котором хранится количество бомб на позицию для каждого игрока. Инициализирован нулями: `bomb = [[0,0,0,0,0], [0,0,0,0,0]]`.

Обратите внимание, что одну и ту же позицию можно бомбить несколько раз, поэтому необходимо хранить количество. Этот список будет использоваться для расчета ущерба, причиненного кораблю.

Затем сценарий считывает игровые данные из HTTP-запроса (листинг 6.6).



**Листинг 6.6.** Чтение данных из HTTP-запроса

```
# CGI-анализ HTTP-запроса
form = cgi.FieldStorage()

ships1 = form["ships1"].value
ships2 = form["ships2"].value
bombs1 = form["bombs1"].value
bombs2 = form["bombs2"].value

# 'local_qasm_simulator', 'ibmqx_qasm_simulator'
device = str(form["device"].value)

shipPos[0] = list(map(int, ships1.split(","))) # [0,1,2]
shipPos[1] = list(map(int, ships2.split(","))) # [0,1,2]

bomb[0] = list(map(int, bombs1.split(",")))
bomb[1] = list(map(int, bombs2.split(",")))

stdout = "Ship Pos: " + str(shipPos) + " Bomb counts: " + str(bomb) + "<br>"
```

Чтобы прочитать данные из HTTP-запроса, используйте `form = cgi.FieldStorage()`. Этот вызов CGI возвращает словарь или хеш-карту пар «ключ — значение», с помощью которой извлекаются параметры строки запроса. В данном конкретном случае ожидаются следующие значения:

- `ships1` — трехэлементный массив данных в формате JSON с позициями корабля первого игрока;
- `ships2` — трехэлементный массив данных в формате JSON с позициями корабля второго игрока;
- `bombs1` — пятиэлементный массив данных в формате JSON со счетчиками бомб первого игрока;
- `bombs2` — пятиэлементный массив данных в формате JSON со счетчиками бомб второго игрока;
- `device` — устройство, на котором будет выполняться квантовая программа. Это может быть:
  - `local_qasm_simulator` — локальное моделирующее устройство, упакованное вместе с QISKit;
  - `ibmq_qasm_simulator` — удаленное моделирующее устройство, предоставляемое IBM;
  - `ibmqx2` — пятикубитный квантовый процессор, предоставляемый IBM Q Experience.

Самое замечательное в Python — это то, что данные в формате JSON, предоставляемые HTTP-запросом, в два счета могут быть отображены в поддерживаемые им коллекции:

```
shipPos[0] = list(map(int, ships1.split(",")))  
bomb[0] = list(map(int, bombs1.split(",")))
```

---

### ПРИМЕЧАНИЕ

В Python системный вызов `split(SEPARATOR)` используется для создания списка элементов типа `String`. Но нам нужен список целых чисел. Для его получения мы применяем системный вызов `map(DATA-TYPE, LIST)`. Обратите внимание, что в Python 3 вызов `map` возвращает хеш-таблицу (словарь), поэтому мы должны использовать системный вызов `list` для преобразования в нужный нам список целых чисел. Это здорово, потому что сценарий может повторно задействовать старые структуры данных и получится сохранять большую часть квантовой логики нетронутой.

---

Последняя строка листинга 6.6 — это просто строковый буфер стандартного вывода, который будет возвращен браузеру для отладки. Наконец, в листинге 6.7 показана внутренняя логика сценария, которая по большей части остается нетронутой.

### Листинг 6.7. Основной раздел квантового сценария

```
# В переменной grid[player] будут храниться результаты для поля каждого  
# игрока  
grid = [{}, {}]  
  
# Теперь мы создаем и запускаем на выполнение квантовые программы,  
# которые реализуют на поле каждого игрока следующее  
for player in range(2):  
  
    # Настройка квантовой программы (QASM) для моделирования сетки  
    # данного игрока  
  
    Q_program = QuantumProgram()  
    Q_program.set_api(Qconfig.APIToken, Qconfig.config["url"])  
  
    # Объявление регистра из пяти кубитов  
    q = Q_program.create_quantum_register("q", 5)  
    # Объявление регистра из пяти классических битов для хранения
```

```

# результатов измерений
c = Q_program.create_classical_register("c", 5)
# Создание схемы
gridScript = Q_program.create_circuit("gridScript", [q], [c])

# Добавление бомб (установленных противником)
for position in range(5):
    # Добавляется столько бомб, сколько было помещено в данную позицию
    for n in range( bomb[(player+1)%2][position] ):
        # Эффективность бомбы
        # (квантовой операции, которая будет применена)
        # зависит от типа корабля
        for ship in [0,1,2]:
            if ( position == shipPos[player][ship] ):
                frac = 1/(ship+1)
                # Добавление этой части вентили НЕ в QASM
                gridScript.u3(frac * math.pi, 0.0, 0.0, q[position])
# Наконец, выполнить измерения
for position in range(5):
    gridScript.measure(q[position], c[position])

# Чтобы увидеть, какие действия должен выполнить квантовый
# компьютер, мы можем вывести на печать файл с QASM
# Эта строка обычно закомментирована
#print( Q_program.get_qasm("gridScript") )

# Скомпилировать и запустить на выполнение QASM
results = Q_program.execute(["gridScript"], backend=device,
shots=shots)

# Извлечение данных
grid[player] = results.get_counts("gridScript")

# Если один из запусков был неудачным, сообщить об этом игрокам
# и начать раунд заново
if ( ( 'Error' in grid[0].values() ) or ( 'Error' in grid[1].
Values() ) ):

    stdout += "The process timed out. Try this round again.<br>"

else:

    # Рассмотрим ущерб во всех кубитах (даже там, где нет кораблей)
    damage = [ [0]*5 for _ in range(2)]

    # Для этого мы пройдем в цикле по всем пяти битовым строкам
    # каждого из игроков
    for player in range(2):
        for bitString in grid[player].keys():
            # а затем по всем позициям

```

```

    for position in range(5):
        # Если в строке для данной позиции находится значение 1,
        # то распределение вероятности добавляется в ущерб
        # Помните, что бит нулевой позиции – крайний справа
        # и соответствует bitString[4]
        if (bitString[4-position]=="1"):
            damage[player][position] += grid[player][bitString]/
            shots

    stdout += "Damage: " + str(damage) + "<br>"

```

В основной раздел первоначального сценария было внесено несколько небольших изменений.

- Все операторы вывода на печать были удалены. Вместо этого используется стандартный выходной строковый буфер для возврата информации клиенту. Это сделано потому, что при выводе в Python информация будет выгружаться непосредственно в HTTP-ответ, что испортит формат JSON, который мы должны вернуть обратно (JavaScript ожидает правильный JSON от AJAX). Обратите внимание, что это необязательный, но полезный шаг, предназначенный для возврата отладочной информации клиенту. В общем, вы можете обойти его, просто закомментировав все операторы вывода на печать (конечно, если произойдет ошибка, трудно будет разобраться, что пошло не так).
- Все операторы пользовательского ввода (считывание положения бомбы, нажатие клавиши Enter для продолжения и др.) были удалены. Помните, что позиции кораблей и счетчики бомб отображаются в коллекции из HTTP-запроса.
- Первоначальный сценарий использует бесконечный цикл while для чтения позиций бомбы. Этот цикл удален. Если этого не сделать, сценарий будет работать вечно и зависнет HTTP-запрос.

Наконец, сценарий возвращает документ в формате JSON в браузер для обновления интерфейса пользователя (листинг 6.8).

#### Листинг 6.8. Отправка ответа в браузер

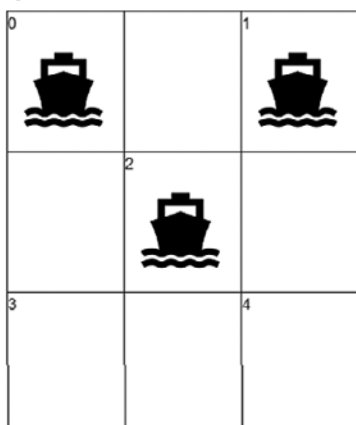
```

# Ответ
print ("Content-type: application/json\n\n")
print ("{"status": 200, "message": \"" + stdout + "\", "damage\": " +
str(damage) + "}")

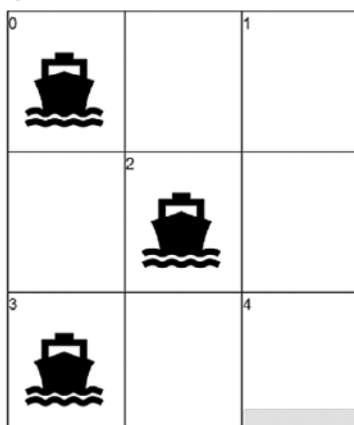
```



Player 1

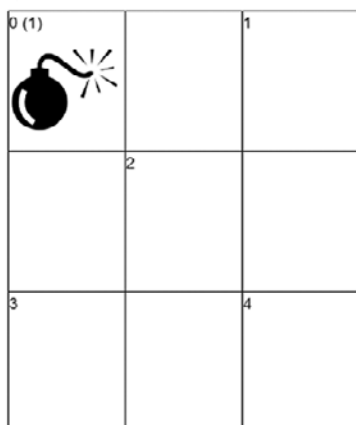


Player 2



Bomb ready. Click Submit

Player 1 Bombs



Player 2 Bombs

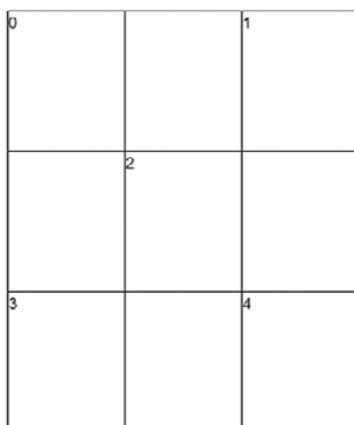


Рис. 6.4. Интерфейс пользователя Quantum Battleship

```

<script type="text/javascript"> table(1, 's')</script>
</td>
<td>
  <div><h3>Player 2</h3></div>
  <script type="text/javascript"> table(2, 's')</script>
</td>
</tr>
<tr>
<td>
  <div><h3>Player 1 Bombs</h3></div>

```

```

        <script type="text/javascript"> table(1, 'b')</script>
    </td>
    <td>
        <div><h3>Player 2 Bombs</h3></div>
        <script type="text/javascript"> table(2, 'b')</script>
    </td>
</tr>
</table>
</form>

```

Рисование игровых полей  $3 \times 3$  ячейки происходит динамически при помощи системного вызова `document.write()` (листинг 6.10).

**Листинг 6.10.** Динамическое рисование таблиц с помощью `document.write()`

```

// type: 's' (ship) = checkbox, 'b' (bomb) = radio
function table (player, type) {
    var d = document;
    var html = '<table border="1">\n';
    var qubit = 0;

    for ( var i = 0 ; i < 3 ; i ++ ) {
        html += '<tr>';

        for ( var j = 0 ; j < 3 ; j ++ ) {
            if ( ( i + j ) % 2 == 0 ) {
                var id = 'p' + player + type + qubit++;


                // checkbox = ship, radio = bomb
                var itype = type == 's' ? 'checkbox' : 'radio';
                var extra = type == 'b' ? ' onclick="cell_click_bomb(this)"' : ' onclick="return cell_click_ship(this)";'

                // <TD> SHIP-INDEX DAMAGE IMAGE </TD>
                html += '<td>' + (qubit - 1)
                    + ' <span id="' + type + player + (qubit - 1) + '">'
                    + '</span>'
                    + '<input id="' + id + '" name="' + id + '" type="' + itype + '" ' + extra + '>'
                    + '<label for="' + id + '" class="ship">&nbsp;</label></td>'
            } else {
                html += '<td>&nbsp;</td>';
            }
        }
        html += '</tr>\n';
    }
    html += '</table>'; d.write(html);
}

```

В табл. 6.1 приведены основные особенности интерфейса пользователя.

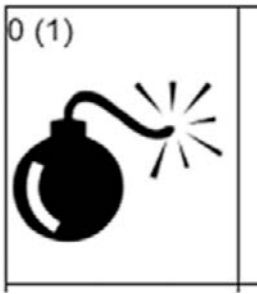
**Таблица 6.1.** Интерфейс пользователя облачной версии Battleship, приемы и хитрости

<p>Мы скрываем флажки и переключатели при помощи таблиц стилей. Селекторы в строках 1 и 2 используют псевдокласс отрицания, чтобы скрыть правило от старых браузеров. Строки с 3-й по 5-ю задают ширину, поля и отступы для точного позиционирования альтернативной графики. В строке 6 устанавливается нулевое значение для непрозрачности, что делает стандартный интерфейс пользователя невидимым</p>	<pre>input[type=checkbox]:not(old), input[type=radio]:not(old){   width : 104px;   margin : 0;   padding : 0;   opacity : 0; }</pre>
<p>Каждая ячейка в таблице кораблей отображает:</p> <ul style="list-style-type: none"><li>• номер кубита;</li><li>• элемент span для отображения процента нанесенного ущерба;</li><li>• &lt;INPUT type = "checkbox"&gt;, модифицированный для использования изображения 100 × 100 пикселей вместо обычного элемента управления</li></ul> 	<pre>input[type=checkbox]:not(old) + label {   display : inline-block;   margin-left : -104px;   padding-left : 104px;   background : url('img/ship.png')   no-repeat 0 0;   line-height : 100px; }</pre> <p>Мы размещаем метку и выводим изображение с неустановленным флажком. В строке 2 для метки устанавливается отображение в виде элемента inline-block, что позволяет в строке 6 задать высоту в соответствии с размерами альтернативной графики и центрировать текст по вертикали. В строке 3 используется отрицательное значение для поля (margin), чтобы покрыть пространство, в котором будет отображаться стандартный интерфейс пользователя, а в строке 4 применяется отступ, чтобы восстановить правильное положение текста метки. Используемое здесь значение 104 пиксела равно ширине изображения плюс некоторый дополнительный отступ, чтобы текст метки не был расположен слишком близко к изображению. В строке 5 показано изображение с неустановленным флажком в поле перед текстом метки</p>



Каждая ячейка в таблице бомб содержит:

- номер кубита;
- элемент `span` для отображения количества бомб в данной позиции;
- `<INPUT type="radio">` модифицированный для использования изображения  $100 \times 100$  пикселей вместо обычного элемента управления



Стиль, который применяется для форматирования бомб, приведен далее:

```
input[type=radio]:not(old) + label{
    display : inline-block;
    margin-left : -104px;
    padding-left : 104px;
    background : url('img/bomb.png')
    no-repeat 0 0;
    line-height : 100px;
}
```

Затем отображаются выбранные картинки, для которых установлены флажки и переключатели:

```
input[type=checkbox]:not(old):checked + label{
    background-position : 0 -100px;
}
input[type=radio]:not(old):checked + label{
    background-position : 0 -100px;
}
```

Поскольку мы объединили изображения для различных состояний в одно, приведенные ранее правила изменяют положение фона, чтобы показать соответствующее изображение

Прекрасные библиотеки jQuery, Bootstrap и Bootstrap-Growl используются для отображения сообщений и отладки информации в консоли JS:

```
<script type="text/javascript"
src="js/log.js"></script>
<script type="text/javascript"
src="js/jquery.js"></script>
<script type="text/javascript"
src="js/bootstrap.js"></script>
<script type="text/javascript"
src="js/bootstrap-growl.js">
</script>
<script type="text/javascript"
src="js/notify.js"></script>
```

Чтобы HTML-код стал красивым, применяется типичная для проектирования GUI библиотека Bootstrap. Сообщения отображаются на экране с помощью библиотеки Bootstrap-Growl из JS:

```
notify('Bomb ready. Click Submit', info);
```



## Правила игры и проверка данных

Поскольку HTTP — это протокол без сохранения состояния, все структуры данных и логика проверки должны быть перенесены на сторону клиента. Например:

- игрокам нельзя разрешать размещать на доске более трех кораблей;
- запрещено менять судно после установки бомбы;
- бомбы нельзя устанавливать до того, как все игроки разместят свои корабли;
- для отслеживания щелчков кнопкой мыши используется глобальный массив счетчиков бомб `var BOMBS = [[0,0,0,0,0], [0,0,0,0,0]]`. Он соответствует аналогу в Python `bomb = [[0]*5 for _ in range(2)]`.

Эти правила можно применить, добавив обратный вызов при щелчке на ячейке корабля или бомбы (листинг 6.11).

**Листинг 6.11.** Установление правил игры с помощью обратных вызовов в исходном коде книги в `index.html`

```
// Вызывается при щелчке на ячейке корабля
function cell_click_ship ( obj ) {
    var id = obj.id;
    var player = parseInt(id.charAt(1));
    var qubit = parseInt(id.charAt(3));
    var json = countShipsBombs();

    LOGD('Cell Clicked ' + id + ' Counts:' + JSON.stringify(json));
    if ( json.ships[0] > 3 || json.ships[1] > 3 ) {
        return error('All Players must place only 3 ships.');
```

}

```
// Запрет на изменение кораблей после установки бомб
if ( json.bombs[0] > 0 || json.bombs[1] > 0 ) {
    return error('No ship changes after bombs are placed.');
```

}

```
} return true;
}
```

// Вызывается при щелчке на ячейке бомбы

```
function cell_click_bomb ( obj ) {
    var id = obj.id; // Для бомб: p[PLAYER]b[QUBIT]
    var player = parseInt(id.charAt(1));
    var qubit = parseInt(id.charAt(3));
```

```

// Проверка: { 'ships': [s1, s2], 'bombs': [b1, b2]}
var json = countShipsBombs();
LOGD('Bomb Clicked ' + id + ' Counts:' + JSON.stringify(json));

if ( json.ships[0] < 3 || json.ships[1] < 3) {
    $('#' + id).attr('checked', false);
    return error('All Players must place 3 ships first.');
```

```

}
if ( mustSubmit) {
    return error('Bomb in place already. Click Submit.');
```

```

}

// Проверка очереди игрока. Ошибки?
var dif = (json.bombs[player - 1] + 1) - json.bombs[ 1 - (player - 1)];

if ( dif >= 2 ) {
    if ( BOMBS[player - 1 ][qubit] < 1 ) {
        $('#' + id).attr('checked', false);
    }
    return error("Not your turn. It's player " + ((1-(player-1)) + 1) );
}

// Подсчет бомб
BOMBS[player - 1 ][qubit]++;

// Присвоить счетчики: d[PLAYER][QUBIT]
$('#' + id + player + qubit).html("(" + BOMBS[player - 1 ][qubit] + ")");

// Бомбы на месте, нажать Подтвердить
notify('Bomb ready. Click Submit', 'info');
mustSubmit = true;
}

function error (msg) {
    notify(msg, 'danger');
    return false
}

```

Теперь данные могут быть отправлены на сервер.

## Конечная точка и формат ответа

Каждый запрос должен отправляться на веб-сервер асинхронно с использованием AJAX. Кроме того, строка запроса должна иметь определенный формат.

Для конечной точки `http://localhost/~centos/battleship/cgi-bin/qiskit-driver.sh` мы предполагаем, что:

- имя пользователя — `centos`;
- код был развернут в домашней папке пользователя `$HOME/centos/public_html/battleship`;
- Python 3 необходимо активировать с помощью сценария-обертки `qiskit-driver.sh`. Это требуется только в том случае, если на хосте имеется несколько версий Python (см. далее раздел «Запуск нескольких версий Python»).

Строка запроса должна содержать следующие значения:

- `ship1` — разделенный запятыми список из трех позиций для кораблей первого игрока;
- `ship2` — разделенный запятыми список из трех позиций для кораблей второго игрока;
- `bombs1` — разделенный запятыми список из пяти бомб для первого игрока;
- `bombs2` — разделенный запятыми список из пяти бомб для второго игрока;
- `device` — квантовое устройство, например `local_qasm_simulator`, `ibmq_qasm_simulator` или `ibmqx2`.

Вот пример полного запроса AJAX для запуска на удаленном моделирующем устройстве:

```
http://localhost/~centos/battleship/cgi-bin/qiskit-driver.sh?ships1=0,1,2&ships2=0,1,2&bombs1=0,1,0,0,0&bombs2=0,0,0,0,0&device=ibmqx_qasm_simulator
```

Когда игрок нажимает кнопку **Submit** (Отправить), позиции кораблей `ships1` и `ships2`, а также счетчики бомб `bombs1` и `bombs2` собираются из дерева DOM и глобального массива `BOMBS`. Конечная точка и строка запроса определены, и HTTP-запрос GET отправляется через AJAX (листинг 6.12).

**Листинг 6.12.** Отправка данных на сервер в файле index.html

```
function submit() {
    var frm = $('#frm1');
    var url = "cgi-bin/qiskit-driver.sh";

    // Формат данных: ships1=0,1,2&ships2=0,1,2&bombs1=0,1,0,0,0
    // &bombs2=0,0,0,0,0
    // Для кораблей указаны позиции по игрокам,
    // Для бомб указано количество для их позиций по игрокам
    // Корабли: три корабля на игрока, бомбы: пять счетчиков для позиций
    var data = "";
    var s1 = "";
    var s2 = "";

    for ( var i = 0 ; i < 5 ; i++) {
        if ( $('#p1s' + i).prop('checked') ) s1 += ',' + i;
        if ( $('#p2s' + i).prop('checked') ) s2 += ',' + i;
    }

    // Удаление первой точки
    if (s1.length > 0) s1 = s1.substring(1);
    if (s2.length > 0) s2 = s2.substring(1);

    // Строка запроса
    data = 'ships1=' + s1 + '&ships2=' + s2
        + '&bombs1=' + BOMBS[0].join(',') + '&bombs2=' + BOMBS[1].join(',')
        + '&device=' + $('#device').val();

    LOGD('Url:' + url + ' data=' + data);

    // https://api.jquery.com/jquery.get/
    $.get( url, data)
    .done(function (json) {
        handleResponse (json);
    })
    .fail(function() {
        LOGD( "error" );
        notify('Internal Server Error. Check logs.', 'danger');
    })
}
```

Если что-то пойдет не так, уведомление об ошибке будет отображаться на экране, в противном случае ожидаемый ответ в формате JSON будет отправлен обработчику. Посмотрим, как это происходит.

## Обработчик ответов

Задача обработчика ответа состоит в том, чтобы использовать ответ сервера и обновлять счетчики нанесенного ущерба, отображать сообщения об ошибках, если таковые имеются, или повторять этот процесс, пока не будет объявлен победитель. Процесс показан в листинге 6.13, но сначала рассмотрим имеющий большое значение формат ответа JSON:

```
{"status":200,"message":"OK","damage":[[0.475,0,0,0.70,0],  
[0.786,0.90,0,0,0.]]}
```

Самым важным ключом является `damage`. Он содержит двумерный массив, представляющий для каждого игрока позиции, в которых корабль поврежден. Значение ущерба находится в диапазоне от 0 до 1. Эти данные используются обработчиком ответа для обновления интерфейса пользователя.

### Листинг 6.13. Обработчик ответа в файле `index.html`

```
function handleResponse (json) {  
    LOGD("Got: " + JSON.stringify(json))  
    var damage = json.damage;  
    var d1 = damage[0]; // ущерб для P1  
    var d2 = damage[1]; // ущерб для P2  
  
    for ( var i = 0 ; i < 5 ; i++) {  
        var pct1 = (d1[i] * 100).toFixed(1);  
        var pct2 = (d2[i] * 100).toFixed(1);  
        var s1, c1, s2, c2;  
        if ( pct1 < 90 ) {  
            s1 = '[' + pct1 + '%]';  
            c1 = 'cyan';  
        }  
        else {  
            s1 = 'SUNK';  
            c1 = 'red';  
            notify('Player 1 Ship ' + i + ' sunk.', 'warning');  
        }  
        if ( pct2 < 90 ) {  
            s2 = '[' + pct2 + '%]';  
            c2 = 'cyan';  
        }  
        else {  
            s2 = 'SUNK';  
            c2 = 'red';  
            notify('Player 2 Ship ' + i + ' sunk.', 'warning');  
        }  
    }  
}
```

```

        //LOGD(i + ' s1=' + s1 + ' s2=' + s2 + ' d1=' + d1[i] +
        // ' d2=' + d2[i]);
        $('#s1' + i).html(s1).css('background-color', c1);
        $('#s2' + i).html(s2).css('background-color', c2);
    }

    // Результат игры: суммарный ущерб sum > 2.85 (0.95 * 3) = loss
    // https://www.w3schools.com/jsref/jsref_reduce.asp
    // array.reduce(function(total, currentValue, currentIndex, arr),
    // initialValue)
    var s1 = d1.reduce(function(total, currentValue, currentIndex, arr)
        { return total + currentValue}, 0);
    var s2 = d2.reduce(function(total, currentValue, currentIndex, arr)
        { return total + currentValue}, 0);
    var winner = 0;
    if ( s1 > 2.85) winner = 2;
    if ( s2 > 2.85) winner = 1;

    LOGD ("Results Damage sums s1:" + s1 + " s2:" + s2);
    if ( winner != 0) {
        notify ('** G.A.M.E O.V.E.R Player ' + winner + ' wins **',
            'success'); gameover = true;
    }

    // Разрешить отправку
    $("#btnSubmit").prop("disabled", false);
}

```

Обработчик извлекает массив для ущерба и проходит в цикле по всем позициям, переводя ущерб для каждого игрока в проценты.

Чтобы более эффектно продемонстрировать нанесенный урон, используется цветная подсветка. Для каждого затонувшего корабля на экран выводятся сообщения (рис. 6.5).

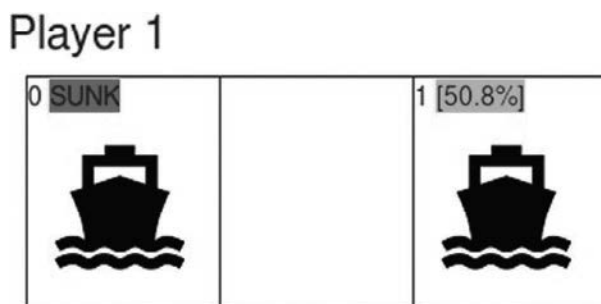


Рис. 6.5. Цветная подсветка поля для отображения ущерба

Если суммарный ущерб превышает предельное значение 90 % для всех кораблей игрока, объявляется победитель и игра заканчивается. Чтобы начать новую игру, нажмите кнопку **Reset** (Сбросить).

Чтобы запустить игру заново, мы просто снимаем все флажки и переключатели и обнуляем глобальный массив **BOMBS** (листинг 6.14).

**Листинг 6.14.** Перезагрузка игры в исходном коде книги в `index.html`

```
// Перезапуск игры вызывается при нажатии кнопки перезапуска
function reset_click () {
    if ( ! confirm("Are you sure?")) {
        return;
    }
    gameover = false;
    for ( var i = 0 ; i < 5 ; i++) {
        $('#p1s' + i).attr('checked', false);
        $('#p2s' + i).attr('checked', false);
        $('#p1b' + i).attr('checked', false);
        $('#p2b' + i).attr('checked', false);
        // элементы span с информацией
        $('#s1' + i).html("");
        $('#s2' + i).html("");
        $('#b1' + i).html("");
        $('#b2' + i).html("");
        BOMBS[0][i] = 0;
        BOMBS[1][i] = 0;
    }
}
```

Теперь пришло время для запуска, развертывания, тестирования и устранения неполадок, если это необходимо. Для разработки я использую CentOS 6, которая по умолчанию включает Python 2.7. Помните, что мы должны взять Python 3.5 или более поздней версии.

---

#### ПРИМЕЧАНИЕ

Листинги 6.9–6.12 находятся в исходниках книги в `Workspace\Ch06\battleship\index.html`, как и ресурсы, необходимые для развертывания игры в облаке.

---



## Запуск нескольких версий Python

В главе 3 объясняется, как установить и запустить Python 3.6 и Python 2.7 по отдельности. В данном конкретном случае сценарий-обертка используется для активации Python 3 на сервере CGI перед вызовом квантовой программы:

```
#!/bin/sh
# Домашняя папка
root=/home/centos
program=qbattleship.py

# Активация python 3
source $root/qiskit/qiskit/bin/activate

# Выполнение квантовой программы на python
python $program
```

В предыдущем сценарии просто активируется Python 3 и вызывается реальная квантовая программа qbattleship.py. Это необходимо, иначе веб-сервер будет использовать версию Python по умолчанию (2.7) и программа завершится с ошибкой, поскольку для QISKit требуется Python 3.5 или более поздней версии. Помните, что среда Python 3 была создана в домашней папке пользователя следующим образом:

```
$ mkdir -p $HOME/qiskit
$ cd $HOME/qiskit
$ python3.6 -m venv qiskit
Активация виртуальной среды:
$ source qiskit/bin/activate
```

Теперь наконец перейдем к развертыванию и тестированию. Надеюсь, что устранять ошибки не потребуется.

## Решение 6.3. Развертывание и тестирование

В этом разделе мы развернем игру на HTTPD-сервере Apache и посмотрим на нее в действии. Полный исходный код игры, включая все файлы поддержки, стили, изображения, оболочку CGI и квантовую программу, можно найти в исходниках книги, в разделе `Workspace\Ch06\battleship`. Расположение папок показано на рис. 6.6.

### ПРИМЕЧАНИЕ

В этом разделе предполагается, что вы уже установили HTTPD-сервер Apache в своей системе и служба по умолчанию настроена и работает правильно.

Если это не так, исправить ситуацию можно, обратившись к учебным пособиям. Например, для CentOS 7 мне нравится [www.liquidweb.com/kb/how-to-install-apache-on-centos-7/](http://www.liquidweb.com/kb/how-to-install-apache-on-centos-7/).

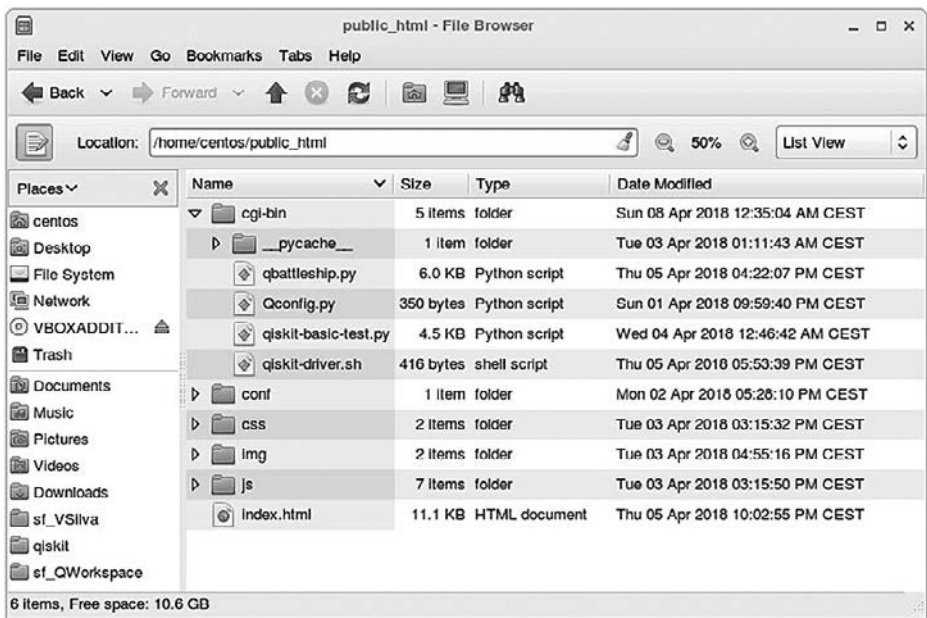


Рис. 6.6. Расположение папок для облачной Quantum Battleship

1. Создайте папку с названием `public_html` в своей домашней папке пользователя:

```
$ mkdir $HOME/public_html
```

2. Создайте папку `cgi-bin` в `public_html` для хранения CGI-сценариев Python:

```
$ mkdir $HOME/public_html/cgi-bin
```

3. Выполните конфигурацию сервера HTTPD, чтобы разрешить доступ как из `public_html`, так и из `public_html/cgi-bin` (листинг 6.15). Обратите внимание, что для `cgi-bin` требуется специальное разрешение для выполнения CGI-сценариев.
4. Если хотите использовать исходный код книги, скопируйте все файлы из `Workspace\Ch06\battleship` в `public_html/battleship`.
5. Убедитесь, что права доступа к файлам установлены корректно для папки `public_html` и всех вложенных папок и файлов. Это очень важно: если разрешения неправильные, браузер выдаст ответ «500 — Internal Server Error». Когда я выполнял тестирование на своем персональном компьютере с CentOS 6, основным источником головной боли было `$ chmod -R 755 public_html`.

**Листинг 6.15.** Конфигурация, разрешающая HTTP-запросы из папки `public_html` (CentOS 6/Apache HTTPD 2.2)

```
<IfModule mod_userdir.c>
    # UserDir disabled
    #
    # Чтобы разрешить запросам к /~user/ обращаться к папке public_html,
    # удалите строку "UserDir disabled" выше и взамен раскомментируйте
    # следующую строку:
    #
    UserDir public_html
</IfModule>

<Directory /home/*/public_html>
    AllowOverride FileInfo AuthConfig Limit
    Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
    +ExecCGI
    AddHandler cgi-script .cgi
    <Limit GET POST OPTIONS>
        Order allow,deny
        Allow from all
    </Limit>
</Directory>

<Directory "/home/*/public_html/cgi-bin">
    AllowOverride None
    Options ExecCGI
    SetHandler cgi-script
</Directory>
```

### СОВЕТ

Чтобы разрешить запросы из `public_html` (см. листинг 6.15), необходимо, чтобы в `httpd.conf` был включен модуль `userdir` для Apache (раскомментируйте `LoadModule userdir_module modules / mod_userdir.so`). Этот модуль может быть заблокирован по умолчанию.

---

Скопируйте сценарий из листинга 6.15 в системную папку `/etc/httpd/conf.d`. Она содержит файлы конфигурации, автоматически загружаемые HTTPD-сервером Apache при запуске. Теперь запустите сервер HTTPD в CentOS (обратите внимание: предполагается, что в вашей системе уже установлен HTTPD-сервер Apache):

```
$ sudo service httpd start    (CentOS 6)
$ sudo systemctl start httpd  (CentOS 7)
```

Наконец, в качестве финального аккорда запустите браузер и перейдите по URL-адресу `http://localhost/~centos/battleship/` (при условии, что имя пользователя — `centos`). Надеюсь, проблем не будет и вы сможете начать играть в Quantum Battleship в облаке. Но если что-то пойдет не так, изучите приведенный далее список проблем, с которыми я столкнулся при настройке.

## Устранение ошибок

Большинство проблем, с которыми я столкнулся, были связаны с правами доступа к файлам из-за моей застарелой привычки использовать старый добрый HTTPD-сервер Apache.

- Особенность HTTPD-сервера Apache заключается в следующем: чтобы разрешить запросы из домашней папки пользователя (см. листинг 6.15), необходимо, чтобы модуль `userdir` был включен в конфигурации демона `httpd.conf`. В зависимости от вашей ОС этот модуль может быть заблокирован по умолчанию. Пользователи HTTPD 2.4 должны учесть: листинг 6.15 предназначен для Apache v2.2, а v2.4 может потребоваться другой синтаксис.
- HTTP-статус 500 — внутренняя ошибка сервера в браузере: убедитесь, что права доступа к файлам для `public_html` и всех вложенных файлов

и папок установлены на 755. Вы можете диагностировать это, посмотрев файлы журналов HTTPD, расположенные по адресам:

```
/var/log/httpd/error_log  
/var/log/httpd/suexec.log
```

Например, вот фрагмент из `suexec.log`, в котором говорится, что мои разрешения некорректны:

```
$ tail -f /var/log/httpd/suexec.log  
[2018-04-02 17:03:45]: cannot get docroot information (/home/centos)  
[2018-04-02 17:10:13]: uid: (500/centos) gid: (500/centos) cmd: first.cgi  
[2018-04-02 17:10:13]: directory is writable by others: (/home/centos/  
public_html)
```

---

### ПРИМЕЧАНИЕ

suEXEC — это функция веб-сервера Apache. Он позволяет пользователям запускать приложения CGI и SSI от имени другого пользователя. В CentOS suEXEC записывает журналы в `/var/log/httpd/suexec.log`.

---

- Проблемное место SELinux — модуль безопасности ядра Linux, который обеспечивает механизм поддержки политик безопасности контроля доступа. В CentOS данная функция включена по умолчанию. Ее можно отключить временно в командной строке с помощью команды:

```
$ sudo setenforce 0
```

или постоянно, отредактировав файл `/etc/sysconfig/selinux` и установив для ключа `SELINUX` значение `disabled`:

```
$ sudo vi /etc/sysconfig/selinux  
SELINUX=disabled  
SELINUXTYPE=targeted
```

Обратите внимание, что SELinux может стать причиной проблем при вызове сценариев CGI или когда квантовая программа пытается выполнить код удаленно на моделирующем устройстве IBM либо на реальном устройстве.

- Ошибки Python: если в сценарии Python возникает какая-либо ошибка, обработчик исключений CGI перехватывает ее и выдает красивую

HTML-страницу в текущем рабочем каталоге (`cgi-bin`). На рис. 6.7 показан вывод ошибки превышения лимита времени при выполнении на реальном квантовом устройстве `ibmqx2`.

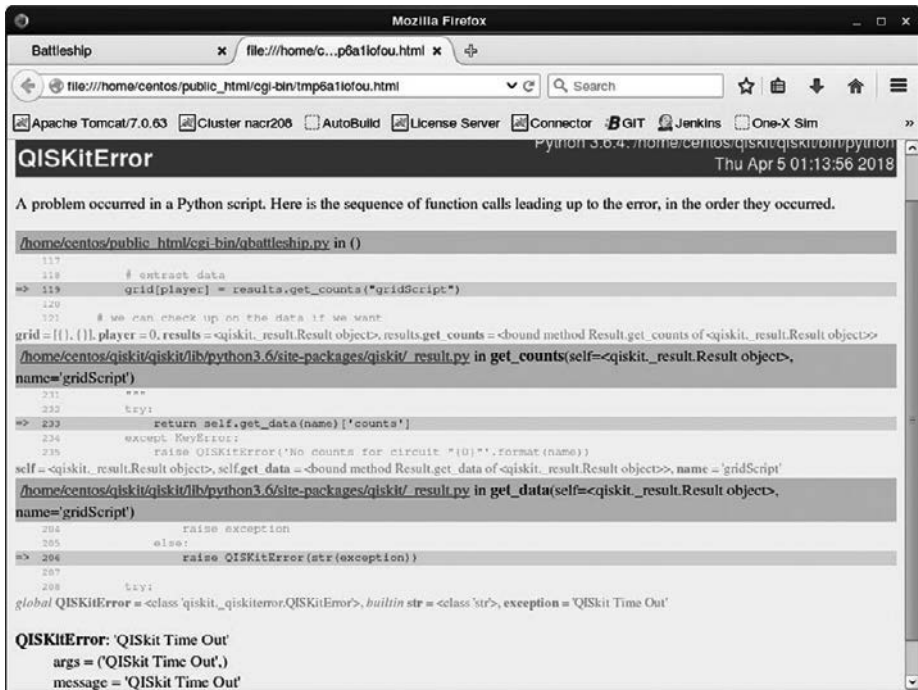


Рис. 6.7. Вывод ошибок в Python, созданный пакетом `cgi`

- Проблемы конфигурации API: если вы работаете на реальном квантовом устройстве, убедитесь, что конфигурация в `Qconfig.py` корректна (включая токен API для вашей учетной записи Q Experience), как показано в следующем фрагменте:

```
APIToken = 'YOUR-API-TOKEN'
config = {
    'url': 'https://quantumexperience.ng.bluemix.net/api',
}
```

Обратите внимание, что `Qconfig.py` должен находиться в том же месте, что и квантовая программа `qbattleship.py`, то есть в папке `cgi-bin`. Тем не менее в игру можно внести дополнительные улучшения. Обсудим их в следующем разделе.

## Дополнительные улучшения

В Cloud Battleship, рассмотренной в предыдущем разделе, можно внести некоторые улучшения — наверняка вы сами заметите это, поиграв какое-то время. Вот список моих идей.

- Интерфейс пользователя отображает игровые поля с кораблями и бомбами для обоих игроков. В настоящей игре «Морской бой» каждый игрок должен открыть собственное окно браузера, установить свои корабли и начать бомбардировку противника.
- Игровое состояние: корабль, позиции бомб и квантовое устройство хранятся на стороне клиента из-за того, что HTTP — это протокол без сохранения состояния. То есть приходит запрос, запускается программа на Python, и ответ отправляется обратно. После этого вся память стирается. Настоящая игра должна использовать серверный лобби-клиент игрока для размещения игрового состояния (например, с применением сервера приложения) и координации взаимодействия между окнами браузера.

## Улучшенная версия Cloud Battleship

Максимально улучшенная версия игры Cloud Quantum Battleship должна использовать два окна браузера для двух игроков с игровыми полями для размещения кораблей и бомб в каждом. Кроме того, HTTPD-сервер Apache нужно заменить сервером приложения, таким как Apache Tomcat, который способен хранить состояние игры. Схема показана на рис. 6.8.

- Элементарный лобби-клиент игры можно реализовать в виде веб-приложения Tomcat для хранения кораблей, позиций бомб и квантовых устройств.
- Веб-приложение может использовать средства времени выполнения операционной системы хоста, в данном случае API времени выполнения Java, чтобы вызвать квантовый сценарий Python, вернуть результаты нанесенного ущерба и отправить их каждому игроку.
- Чтобы избежать раздражающего постоянного обновления страниц, каждый браузер может подключаться через WebSocket к серверу приложений. Это позволит поддерживать постоянное соединение, а клиенты смогут быстро обмениваться текстовыми сообщениями JSON.

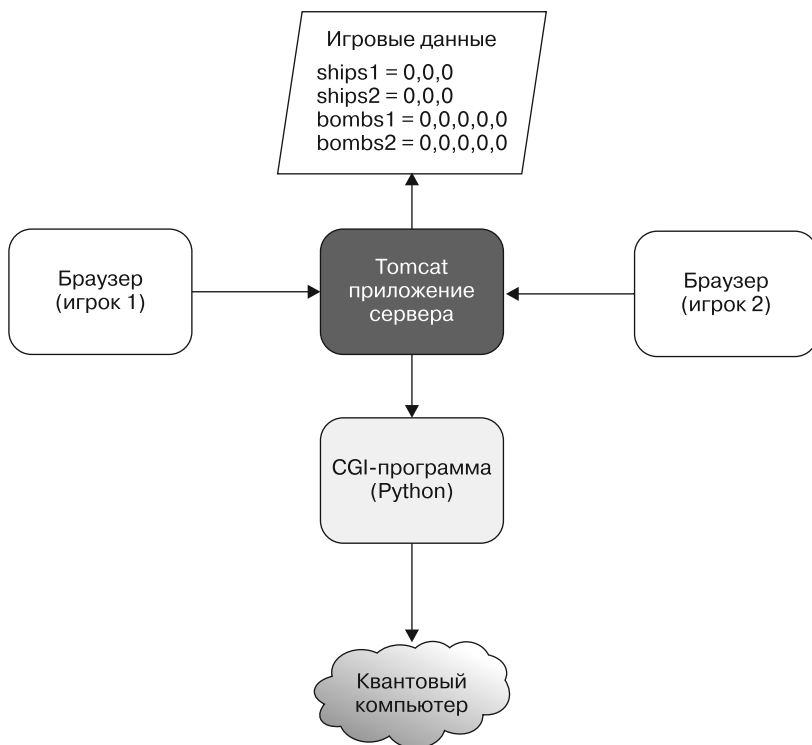


Рис. 6.8. Улучшенная версия Cloud Quantum Battleship

## Соединение через WebSocket

Для того чтобы соединение происходило через WebSocket вместо AJAX, веб-страницу интерфейса пользователя следует немного изменить, как показано в листинге 6.16.

---

### СОВЕТ

Проект Eclipse для этого раздела находится в исходниках книги в `Workspace\Ch06_BattleShip`. Учитывая, что некоторые детали этого веб-приложения сложно реализуются, рекомендую открыть рабочее пространство в своей IDE и дальше читать, параллельно работая с ним. Надеюсь, что вы хорошо умеете писать приложения с помощью Eclipse/Tomcat.

---



**Листинг 6.16.** Код клиента WebSocket на JavaScript в WebContent/js/websocket.js

```
// Конечная точка сервера WS (file: websocket.js)
var END_POINT = "ws://localhost:8080/BattleShip/WSBattleShip";

// Случайный ID, используемый для отслеживания клиента
var CLIENT_ID = Math.floor(Math.random() * 10000);

function WS_connect(host) {
    LOGD("WS Connect " + host);

    if ('WebSocket' in window) {
        this.socket = new WebSocket(host);
    } else if ('MozWebSocket' in window) {
        this.socket = new MozWebSocket(host);
    } else {
        LOGE('Error: WebSocket is not supported by this browser.');
```

return;

```
    }

    this.socket.onopen = function() {
        LOGD('WS Opened ' + host);
    };

    this.socket.onclose = function() {
        LOGD('WS Closed ' + host);
    };

    this.socket.onmessage = function(message) {
        // { status: 200 , message : '...' }
        LOGD('OnMessage: ' + message.data);
        var json = JSON.parse(message.data);

        if ( json.status >= 300 && json.status < 400 ) {
            // Предупреждение
            notify(json.message, 'warning');
        }
        if ( json.status >= 400 ) {
            // Ошибка
            notify(json.message, 'danger');
            return;
        }
        handleResponse (json);
    };
}

function WS_initialize () {
    var clientId = CLIENT_ID;
    var host      = END_POINT;
    this.url      = host + '?clientId=' + clientId;
```

```
        WS_connect(this.url);  
    };  
    function WS_send (text) {  
        this.socket.send(text);  
    };
```

На стороне клиента:

- все основные браузеры реализуют стандарт WebSocket, который используется для поддержки долговременного соединения с сервером. Для этого в строке 2 создана конечная точка `ws://localhost:8080/BattleShip/WS Battleship`. Обратите внимание, что этот параметр можно отправить в список конечных точек WebSocket как обычный URL. Таким образом, конечным URL для WS будет `ws://localhost:8080/BattleShip/WS Battleship?clientId=RANDOM-ID`, где для отслеживания каждого из игроков применяется случайный ID;
- WebSocket использует в JavaScript системный обратный вызов для получения следующих событий:
  - `socket.onopen` — вызывается при открытии сокета, в строке 23 показан обратный вызов для обработки этого события;
  - `socket.onclose` — вызывается при разрыве соединения, например, когда браузер был закрыт или обновился либо упал сервер;
  - `socket.onmessage` — это самая важная функция обратного вызова, которая выполняется при получении сообщений в формате JSON, отправленных Python, как это было с AJAX в предыдущей версии.

При загрузке страницы в браузере игрока клиент устанавливает соединение при помощи обратного вызова DOM `window.onload`:

```
function win_onload () {  
    WS_initialize ();  
}  
window.onload = win_onload;
```

На серверной стороне нам нужен сервер приложений с поддержкой WebSocket. К счастью, Tomcat в полной мере реализует стандарт WebSocket во всех операционных системах. В листинге 6.17 показана базовая реализация обработчика событий на стороне сервера WebSocket на Java.

**Листинг 6.17.** Структура обработчика событий на стороне сервера WebSocket (WSConnector.java)

```
@ServerEndpoint(value = "/WSBattleship")
public class WSConnector {
    // Установка соединений
    private static final List<WSConnectionDescriptor> connections =
        new CopyOnWriteArrayList<WSConnectionDescriptor>();

    // Данные из игры Player-ID => {name: 'Player-1', ships: "0,0,0,0,0,0",
    // bombs: "0,0,0,0,0,0 }
    private static final Map<String, JSONObject> data =
        new HashMap<String, JSONObject>();

    /** Id клиента для данного WS */
    String clientId;
    private String getSessionParameter (Session session, String key) {
        if ( ! session.getRequestParameterMap().containsKey(key)) {
            return null;
        }
        return session.getRequestParameterMap().get(key).get(0);
    }

    @OnOpen public void open(Session session) {
        clientId = getSessionParameter(session, "clientId");

        // Проверка id клиента на уникальность
        WSConnectionDescriptor conn = findConnection(clientId);

        if ( conn != null) {
            unicast(conn.session,
                WSMessages.createStatusMessage(400
                    , "Rejected duplicate session.").toString());
        }
        else {
            connections.add(new WSConnectionDescriptor(clientId, session));
        }
        dumpConnections("ONOPEN " + clientId );
    }

    @OnClose
    public void end() {
    }

    @OnMessage
    public void incoming(String message) {
        WSConnectionDescriptor d = findConnection(clientId);
        try {
            JSONObject root = new JSONObject(message);
        }
    }
}
```

```

        String name = root.getString("name");
        String action = root.optString("action");
        // перезапустить игру?
        if ( action.equalsIgnoreCase("reset")) {
            multicat(WSMessages.createStatusMessage(300
                , "Game reset by " + name).toString());
            data.clear();
            return;
        }

        // Валидация правил игры...
        // Выполнение скрипта Python
        linuxExecPython(args);

    } catch (Exception e) {
        LOGE("OnMessage", e);
    }
}

@OnError
public void onError(Throwable t) throws Throwable {
    LOGE("WSError: " + t.toString());
}
}

```

В Java обработчики событий на стороне сервера WebSocket реализованы в соответствии со стандартом для аннотаций J2EE, что позволяет всем разработчикам повторно использовать код.

- В строке 1 листинга 6.17 в классе Java WSConnector определяется аннотация `@ServerEndpoint(value = "/WSBattleship")`. И это все, что нам нужно для создания обработчика событий на стороне сервера. Значение `WSBattleship` соответствует названию обработчика событий, таким образом, полное название конечной точки соединения с сервером будет `ws://host:POT/Battleship/WSvattleship?QUERY-STRING`.
- Обратные вызовы для событий открытия, закрытия и выдачи сообщения объявляются с помощью аннотаций `@OnOpen`, `@OnClose` и `@OnMessage` соответственно.

Обратите внимание на то, что название метода несущественно, значимыми являются следующие параметры:

- `OnOpen` — получает объект `Session`, который содержит информацию о соединении;

- `OnClose` — без параметров, выполняется при завершении соединения с браузером;
  - `OnMessage` — самый важный вызов в этом наборе. Он выполняется, когда клиент отправляет текстовое сообщение с данными в качестве параметра.
- Имейте в виду, что для каждого клиентского соединения будет создано по одному экземпляру класса `WSConnector`. В строке 5 для отслеживания всех клиентских соединений используется потокобезопасный статический список `List<WSConnectionDescriptor>`. В строке 5 объявляется статическая хеш-таблица для игровых данных, где ключом является ID игрока, а значением — объект JSON, отправленный браузером, например `[Player-1 => {name: 'Player-1', ships: "0,0,0:", bombs: "0,0,0,0,0", device: "local_qasm_simulator"}]`.
- Когда происходит обратный вызов при получении сообщения (строки 201–253 в файле `WSConnector.java`), текстовое сообщение преобразуется в формат JSON, данные сохраняются в памяти, применяются правила игры и, если все корректно, вызывается скрипт Python с позициями кораблей и бомб. В конце полученные результаты отправляются каждому клиенту для обновления.

Для отправки сообщения на сторону клиента можно использовать объект `Session session.getBasicRemote().sendText("Some Text")`.

Чтобы отправить групповое сообщение всем клиентам, можно воспользоваться списком сообщений:

```
static void multicast ( String message ) {  
    for ( WSConnectionDescriptor conn : connections ) {  
        conn.session.getBasicRemote().sendText(message)  
    }  
}
```

## Вызов Python и установка прав доступа к файлам в Java

Несмотря на то что язык Java спроектирован так, чтобы быть независимым от ОС, системные команды можно запустить с помощью системного вызова `Runtime.getRuntime().exec("command")`.

В листинге 6.18 показан очень простой класс для выполнения команды и считывания ее стандартного вывода в строковый буфер.

**Листинг 6.18.** Выполнение системных команд и извлечение результатов (SysRunner.java)

```
public class SysRunner {
    final String command;
    final StringBuffer stdout = new StringBuffer();
    final StringBuffer stderr = new StringBuffer();

    public SysRunner(String command) {
        this.command = command;
    }

    public void run () throws IOException, InterruptedException {
        final Process process = Runtime.getRuntime().exec(command);
        pipeStream(process.getInputStream(), stdout);
        pipeStream(process.getErrorStream(), stderr);
        process.waitFor();
    }

    private void pipeStream (InputStream is, StringBuffer buf) throws
        IOException {
        BufferedReader br = new BufferedReader(new
            InputStreamReader(is));
        String line;

        while ((line = br.readLine()) != null) {
            buf.append(line);
        }
    }

    public StringBuffer getStdOut () {
        return stdout;
    }

    public StringBuffer getStdErr () {
        return stderr;
    }
}
```

Чтобы получить выходные данные команды, используйте входной поток процесса, прочитайте из него данные и сохраните в строковом буфере (строки 17–24 в листинге 6.18): `pipeStream(process.GetInputStream(), stdout)`. Теперь у нас есть инструмент для запуска программы на Python, но нам все еще нужно иметь дело с правами доступа к файлам в Linux. Помните, что скрипт Python необходимо включить в само веб-приложение

(рис. 6.9). Поэтому, когда сервер приложений извлечет в файловую систему веб-приложение Battleship вместе с кодом Python, сценарий получит установленное по умолчанию право доступа к файлу 644 (не исполняемому в среде). Это приведет к сбою сценария при запуске.



Рис. 6.9. Схема проекта J2EE для Cloud Battleship

Чтобы установить корректные права доступа к файлам для кода Python внутри веб-приложения, выполните системную команду `chmod` с названиями файлов, как показано в следующем отрывке кода:

```
// Получить путь к основному каталогу для кода Python
// ...webapps/BattleShip/python/
String root = IOTools.getResourceAbsolutePath("/") + "../..";

// Нельзя использовать специальные символы *&$#
String cmd = "/bin/chmod 755 " + base + "python" + File.separator;

String[] names = { "Qconfig.py", "qiskit-basic-test.py"
    , "qiskit-driver.sh", "qbattleship-sim.py", "qbattleship.py"};
```

```

for (int i = 0; i < names.length; i++) {
    SysRunner r = new SysRunner(cmd + names[i]);
    r.run();
}

```

Путь к основному каталогу установки приложения в Java можно получить с помощью отражения (reflection), как показано далее:

```

public static String getResourceAbsolutePath(String resourceName) throws
UnsupportedEncodingException {
    URL url = IOTools.class.getResource(resourceName);
    String path = URLDecoder.decode(url.getFile(), DEFAULT_ENCODING);

    // path -> Windows: /C:/.../Workspaces/.../
    // path-> Linux: /home/users/foo...
    if ( path.startsWith("/") && OS_IS_WINDOWS) {
        // Должен быть удален первый символ / (только в Windows!)
        path = path.replaceFirst("/", "");
    }
    return path;
}

```

Наконец, квантовую программу на Python можно выполнить из обратного вызова при получении сообщения WebSocket, как показано в листинге 6.19.

**Листинг 6.19.** Выполнение квантовой программы и отправление результатов на сторону клиента

```

// Аргументы: ships1=0,0,0 ships2=0,0,0 bombs1=0,0,0,0,0 bombs2=0,0,0,0,0
// device=local_qasm_simulator
private void linuxExecPython (String args) throws Exception {
    // STDOUT {status: 200, message: 'Some text', damage:
    // [[0,0,0,0,0],[0,0,0,0,0]]}
    StringBuffer stdout = IOTools.executePython(SCRIPT_ROOT, args);
    JSONObject resp = new JSONObject(stdout.toString());

    // Отправить назад клиентам в обратном порядке
    JSONArray damage = resp.getJSONArray("damage");
    resp.remove("damage");
    final int size = damage.length() - 1;

    for (int i = 0; i < connections.size(); i++) {
        resp.put("damage", damage.get( size - i));
        unicast(connections.get(i).session, resp.toString());
        resp.remove("damage");
    }
}

```



```
// Основной каталог: WEAPP_PATH/python/qiskit-driver.sh
// Аргументы: WEAPP_PATH/python/qbattleship.py
//          0,0,0 0,0,0 0,0,0,0,0 0,0,0,0,0 device
public static StringBuffer executePython (String base, String args)
throws IOException, InterruptedException {
    String driver = base + File.separator + "python" + File.separator +
    "qiskit-driver.sh";
    String program = base + File.separator + "python" + File.separator +
    "qbattleship.py";
    String cmd = driver + " " + program + ( args != null ? " " + args : "");

    SysRunner r = new SysRunner(cmd);
    r.run();
    return r.getStdOut();
}
```

Чтобы выполнить квантовую программу на Python, код в листинге 6.19:

- получает расположение (`LOCATION`) каталога с кодом на Python внутри веб-приложения. Это `ТОМКАТ-ROOT/webapps/Battleship/python`;
- выполняет скрипт драйвера `LOCATION/qiskit-driver.sh` `LOCATION/qbattleship.py` с аргументами:
  - `ships1` — расположение кораблей первого игрока;
  - `ships2` — расположение кораблей второго игрока;
  - `bombs1` — счетчики бомб первого игрока;
  - `bombs2` — счетчики бомб второго игрока;
  - `device` — квантовое устройство;
- отправляет результаты клиентам.

Наконец, из своей IDE экспортируйте веб-архив Cloud Quantum Battleship (WAR), разверните его в контейнере Tomcat и играйте из двух веб-браузеров, перейдя по адресу <http://localhost:8080/BattleShip/> (рис. 6.10). И хотя я не сомневаюсь в том, что у вас достаточно опыта, все же на всякий случай приведу порядок действий.

1. Экспортируйте веб-приложение как веб-архив WAR и щелкните правой кнопкой мыши на проекте `Ch06_Battleship` в своей IDE (см. рис. 6.9). Нажмите **Export** ► **Web Archive** (Экспортировать ► Веб-архив) и выберите файл (то есть `Ch06_Battleship.war`).

2. Убедитесь в том, что Tomcat запущен и выполняется. Если он не установлен в вашей ОС по умолчанию, то вам помогут следующие команды:

```
yum -y install java (CentOS 6,7)
yum -y install tomcat7 tomcat7-webapps tomcat7-admin-webapps
(CentOS 6,7)

service tomcat7 start (CentOS 6)
systemctl start tomcat7 (CentOS 7)
```

3. Используйте интерфейс пользователя менеджера Tomcat, находящийся по адресу <http://yourhost:8080/manager/>, чтобы загрузить и развернуть архив в своем контейнере Tomcat в Linux. (Совет: менеджер запросит имя пользователя и пароль, если у вас их нет, отредактируйте файл `/etc/tomcat7/tomcat-users.xml`.)
4. Теперь у вас должна быть возможность указать два браузера на <http://localhost:8080/BattleShip/> (совет: веб-приложения Tomcat развернуты в папке `/var/lib/tomcat7/webapps`). Есть проблемы? Проверьте журналы контейнера в `/var/log/tomcat7/catalina.out`.

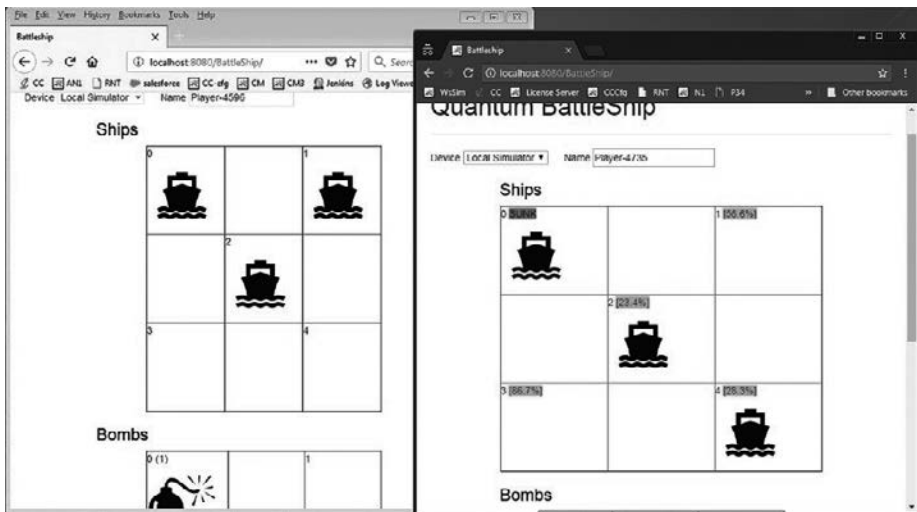


Рис. 6.10. Улучшенная версия Cloud Battleship с двумя браузерами

В этой главе было показано, как популярную игру Battleship можно запустить на квантовом компьютере с применением однокубитных частичных вентилей НЕ для вычисления степени повреждения корабля.

С этой целью был использован пример Quantum Battleship из учебника по QISKit.

Кроме того, мы вывели игру на новый уровень, значительно улучшив пользовательский интерфейс. Вы узнали, как квантовый код можно вызывать в облаке с помощью сценариев CGI через сервер Apache HTTPD. Дальнейшие улучшения были сделаны, чтобы можно было играть сразу в двух браузерах через контейнер Tomcat J2EE. Код Eclipse для обоих проектов доступен в исходниках книги в папках `Workspace\Ch06` и `Workspace\Ch06_BattleShip` соответственно.

В следующей главе рассматриваются две игры-головоломки, которые демонстрируют удивительную мощь квантовых алгоритмов по сравнению с их классическими аналогами: загадка про фальшивую монету и магический квадрат Мермина — Переса. Это примеры квантовой псевдотелепатии, или способности игроков достигать результатов, которые возможны, только если они во время игры читали мысли друг друга.

# 7

## Теория игр: с квантовой механикой преимущество всегда на вашей стороне

В этой главе исследуются две игровые загадки, которые демонстрируют впечатляющее превосходство квантовых алгоритмов в сравнении с их классическими аналогами.

- Загадка про фальшивую монету. Это классическая задача на взвешивание, предложенная математиком Е. Д. Шеллом в 1945 году. В ней нужно при помощи лабораторных весов за ограниченное число взвешиваний определить монету, вес которой отличается от веса других (фальшивую).
- Магический квадрат Мермина — Переса. Это пример квантовой псевдотелепатии, или способности игроков достигать результатов, которые возможны, только если они во время игры читают мысли друг друга.

В обоих случаях квантовые вычисления наделяют игроков псевдомагическими способностями, как если бы они все время жульничали. Давайте посмотрим, как это происходит.

### Загадка про фальшивую монету

У игрока есть восемь монет и лабораторные весы. Одна из монет фальшивая и поэтому весит меньше остальных. Вы можете найти ее? Давайте вкратце рассмотрим решение, которое показано на рис. 7.1.

1. Положите монеты 1–3 на левую чашу весов, а 4–6 — на правую. Отложите на время монеты 7 и 8.
2. Если перевесила правая чаша весов, то фальшивая — среди монет 1–3 (слева). Помните, что поддельная монета легче. Затем уберите монету 3 и положите на левую чашу весов монету 1, а на правую — монету 2.
  - Если перевешивает правая чаша, то фальшивая — монета 1.
  - Если перевешивает левая чаша, то фальшивая — монета 2.
  - Если весы уравнились, то фальшивая — монета 3.
3. Если перевесила левая чаша весов, то фальшивая — среди монет 4–6. Уберите монету 6 и положите на левую чашу весов монету 4, а на правую — монету 5.
  - Если перевешивает правая чаша, то фальшивая — монета 4.
  - Если перевешивает левая чаша, то фальшивая — монета 5.
  - Если весы уравнились, то фальшивая — монета 6.
4. Если весы уравнились, то фальшивая монета либо 7, либо 8. Положите на левую чашу весов монету 7, а на правую — монету 8 и взвесьте.
  - Если перевешивает правая чаша, то фальшивая — монета 7.
  - Если перевешивает левая чаша, то фальшивая — монета 8.

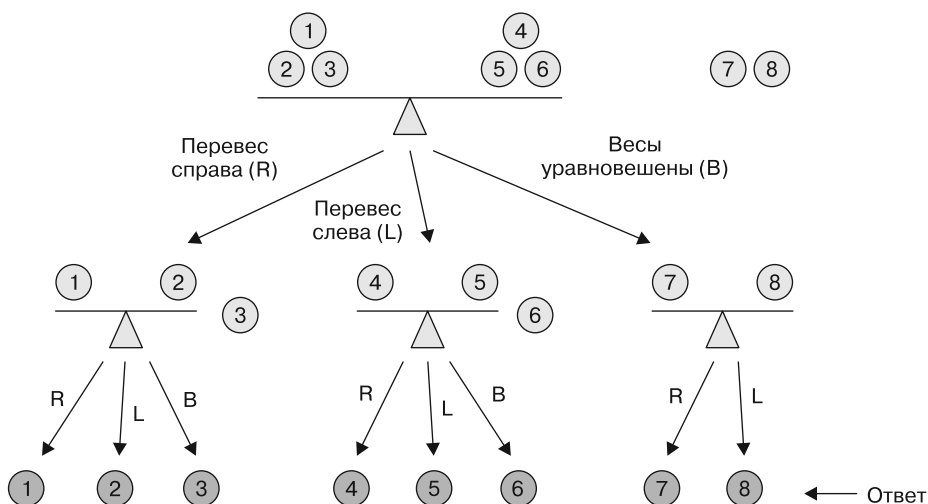
Классический алгоритм можно реализовать вне зависимости от общего числа монет  $N$  и количества фальшивых монет  $k$ . В целом временная сложность для обобщенной задачи о поддельной монете составляет  $O(k \log(N/k))$ .

---

#### ПРИМЕЧАНИЕ

Было доказано, что для обнаружения одной фальшивой монеты при помощи лабораторных весов на классическом компьютере нужны минимум две попытки.

---



**Рис. 7.1.** Решение загадки про фальшивую монету для восьми монет

## Квантовый способ решения

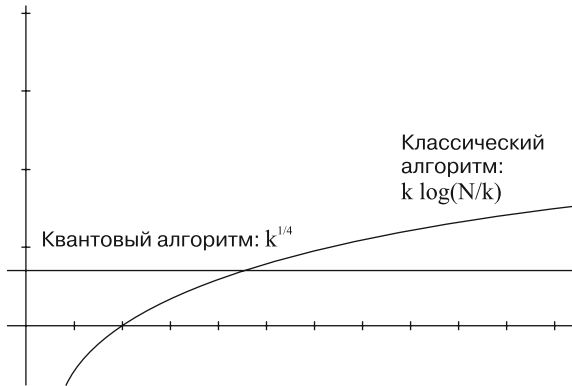
Хотите верить, хотите нет, но существует квантовый алгоритм, который может найти фальшивую монету за одно квантовое взвешивание вне зависимости от количества монет  $N$ ! Вообще говоря, для любого количества фальшивых монет  $k$  независимо от  $N$  временная сложность такого алгоритма составляет  $O(k^{1/4})$ .

### ПРИМЕЧАНИЕ

Квантовый алгоритм определения фальшивой монеты является примером ускорения четвертой степени по сравнению с его классическим аналогом.

Так, на рис. 7.2 показано превосходство квантового алгоритма над классическим аналогом при решении загадки про фальшивую монету. Рассмотрим его подробнее. Квантовый алгоритм поиска одной фальшивой

монеты ( $k = 1$ ) можно разделить на три этапа: запрос к квантовым весам, создание квантовых весов и определение фальшивой монеты.



**Рис. 7.2.** Сравнение временной сложности квантового и классического алгоритмов для загадки про фальшивую монету

## Шаг 1. Запрос к квантовым весам

Квантовый алгоритм будет выполнять запрос к квантовым весам в суперпозиции. Чтобы сделать это, используем бинарную строку запроса для кодирования монет на чашах весов. Например, строка запроса 11101111 означает, что на весах лежат все монеты, кроме монеты с индексом 3. Весы уравновешены, если нет ни одной фальшивой монеты, и наклонены в ином случае. Это проиллюстрировано в следующей таблице.

Количество монет $N$	Индекс фальшивой монеты $F$	Строка запроса	Результат
8	3	11101111	Весы уравновешены (0)
8	3	11111111	Весы наклонены (1)

Алгоритм действий следующий.

1. Использовать два квантовых регистра для запроса к квантовым весам, где первый регистр предназначен для строки запроса, а второй — для результата.

2. Подготовить суперпозицию всех бинарных строк запроса с четным количеством единиц.
3. Для получения суперпозиции состояний с четным количеством единиц выполнить преобразование Адамара в базисном состоянии  $|0\rangle$  и проверить, является ли вес Хэмминга для  $|x\rangle$  четным. Может быть показано, что вес Хэмминга для  $|x\rangle$  является четным тогда и только тогда, когда  $x_1 \oplus x_2 \oplus \dots \oplus x_N = 0$ .

---

#### ПРИМЕЧАНИЕ

Вес Хэмминга ( $hw$ ) строки — это количество символов, отличных от нулевого символа используемого алфавита. Например,  $hw(11101) = 4$ ,  $hw(11101000) = 4$ ,  $hw(000000) = 0$ .

---

4. Наконец, измерить второй регистр. Если наблюдается состояние  $|0\rangle$ , то первый регистр является суперпозицией всех желаемых бинарных строк запроса. Если получено  $|1\rangle$ , то нужно повторять процедуру, пока не будет наблюдаться состояние  $|0\rangle$ .

Обратите внимание, что при каждом повторе вероятность успеха составляет точно 0,5. Однако после нескольких повторов мы сможем получить желаемую суперпозицию состояний. В листинге 7.1 показана реализация квантовой программы для запроса к весам, а соответствующая графическая схема приведена на рис. 7.3.

---

#### ПРИМЕЧАНИЕ

Для упрощения восприятия программа определения фальшивой монеты разбита на листинги 7.1–7.3. Хотя я рассчитываю, что вы сможете объединить эти листинги для запуска программы, полный код есть в исходниках в файле `Workspace\Ch07\p_counterfeitcoin.py`.

---



**Листинг 7.1.** Скрипт запроса к квантовым весам

```
# ----- Запрос к квантовым весам
Q_program = QuantumProgram()
Q_program.set_api(Qconfig.APIToken, Qconfig.config["url"])

# Создание numberOfCoins +1 квантовых/классических регистров
# Один дополнительный кубит для запоминания результата
# квантовых весов
qr = Q_program.create_quantum_register("qr", numberOfCoins + 1)

# для запоминания измерения на qr
cr = Q_program.create_classical_register("cr", numberOfCoins + 1)

circuitName = "QueryStateCircuit"
circuit = Q_program.create_circuit(circuitName, [qr], [cr])

N = numberOfCoins

# Создание равновзвешенной суперпозиции всех строк длиной N
for i in range(N):
    circuit.h(qr[i])

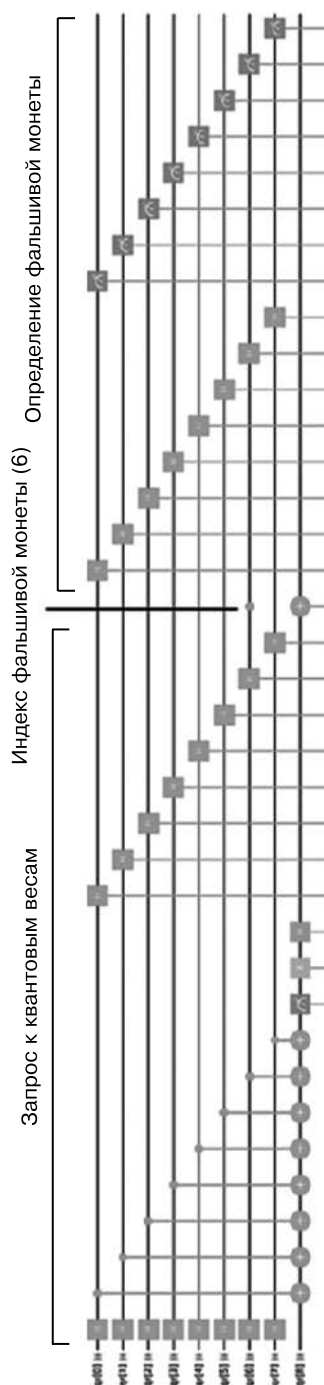
# Выполнение XOR(x) с последовательным применением вентилей CNOT с qr[0]
# по qr[N-1] и сохранением результата в qr[N]
for i in range(N):
    circuit.cx(qr[i], qr[N])

# Измерение qr[N] и сохранение результата в cr[N]. продолжить,
# если cr[N] равен нулю, в противном случае повторить измерение
circuit.measure(qr[N], cr[N])

# Сделать запрос к квантовым весам, если значение нулевое для всех
# cr[0]...cr[N], подготовив состояние вентиля Адамара  $|1\rangle$ ,
# то есть  $|\theta\rangle - |1\rangle$  в qr[N]
circuit.x(qr[N]).c_if(cr, 0)
circuit.h(qr[N]).c_if(cr, 0)

# повторить заново вычисление при ненулевом cr[N]
for i in range(N):
    circuit.h(qr[i]).c_if(cr, 2*N)
```

На рис. 7.3 приведена полная схема для загадки о фальшивой монете с восемью монетами и одной фальшивой с индексом 6. На ней показаны все описанные здесь этапы для платформы IBM Q Experience. Второй этап алгоритма — создание весов.



**Рис. 7.3.** Квантовая схема для загадки про фальшивую монету с  $N = 8$ ,  $k = 1$  и фальшивой монетой с индексом 6 (этот граф в полную величину можно найти на странице загрузки исходного кода)

## Шаг 2. Создание квантовых весов

В предыдущем разделе мы создали суперпозицию всех бинарных строк запроса, у которых вес Хэмминга четный. На данном шаге создаем квантовый балансир, устанавливая позицию фальшивой монеты. Таким образом, если  $k$  — позиция фальшивой монеты относительно бинарной строки  $|x_1, x_2 \dots x_N\rangle|0\rangle$ , то квантовые веса вернут:

$$|x_1, x_2 \dots x_N\rangle|0 \oplus x_k\rangle.$$

Это реализовано с помощью вентиля CNOT с  $x_k$  в качестве управляющего кубита и второго регистра в качестве целевого (см. листинг 7.2).

### Листинг 7.2. Создание квантовых весов

```
#----- Создать квантовые веса
k = indexOfFalseCoin

# Применить квантовые веса к желаемой суперпозиции состояний
# (помеченной как cr, равное нулю)
circuit.cx(qr[k], qr[N]).c_if(cr, 0)
```

## Шаг 3. Определение фальшивой монеты

Чтобы выявить фальшивую монету после запроса к весам, примените преобразование Адамара к бинарной строке запроса. Предполагается, что мы делаем запрос к квантовым весам с бинарными строками с четным весом Хэмминга, поэтому, выполнив измерение в вычислительном базисе после преобразования Адамара, можем определить фальшивую монету, так как только ее метка отличается от меток большинства (см. листинг 7.3).

### Листинг 7.3. Определение фальшивой монеты

```
# --- Определение фальшивой монеты
# Применение преобразования Адамара к qr[0] ... qr[N-1]
for i in range(N):
    circuit.h(qr[i]).c_if(cr, 0)

# Измерение qr[0] ... qr[N-1]
for i in range(N):
    circuit.measure(qr[i], cr[i])
```

```

results = Q_program.execute([circuitName], backend=backend, shots=shots)
answer = results.get_counts(circuitName)

print("Device " + backend + " counts " + str(answer))

# Получение наиболее часто встречающейся метки
for key in answer.keys():
    normalFlag, _ = Counter(key[1:]).most_common(1)[0]

for i in range(2, len(key)):
    if key[i] != normalFlag:
        print("False coin index is: ", len(key) - i - 1)

```

Когда крайний слева бит равен 0, индекс фальшивой монеты можно определить, если найти ту, чей вес отличается от веса остальных. Например, при  $N = 8$  и индексе фальшивой монеты 6 результат должен быть 01011111 или 001000000. Обратите внимание на то, что, поскольку мы используем `cr[N]` для управления операцией до начала и после запроса к весам:

- если крайний слева бит равен 0, то мы успешно определили фальшивую монету;
- если крайний слева бит равен 1, то мы не получили желаемой суперпозиции и должны повторить процесс сначала.

При запуске программы на удаленном моделирующем устройстве IBM Q Experience будет получен результат, приведенный в исходниках книги `workspace\Ch07\p_counterfeitcoin.py`. Обратите внимание, что я использую Windows:

```

c:\python36-64\python.exe p_counterfeitcoin.py
Device ibmq_qasm_simulator counts {'001000000': 1}
False coin index is: 6

```

Если у вас нет доступа к исходникам книги, но вы все равно хотите поэкспериментировать с этим скриптом, то поместите отрывки кода из предыдущих разделов в скрипт-контейнер из листинга 7.4 (проверьте отступы, эта особенность синтаксиса Python просто сводит с ума).

#### **Листинг 7.4.** Основной скрипт-контейнер для загадки про фальшивую монету

```

import sys
import matplotlib.pyplot as plt
import numpy as np

```

```

from math import pi, cos, acos, sqrt
from collections import Counter
from qiskit import QuantumProgram
sys.path.append('../Config/')
import Qconfig

# Импорт основных средств для вывода графики
import basic plot tools
from qiskit.tools.visualization import plot_histogram

def main(M = 16, numberOfCoins = 8 , indexOffalseCoin = 6
        , backend = "local_qasm_simulator" , shots = 1 ):

    if numberOfCoins < 4 or numberOfCoins >= M:
        raise Exception("Please use numberOfCoins between 4 and ", M-1)
    if indexOffalseCoin < 0 or indexOffalseCoin >= numberOfCoins:
        raise Exception("indexOffalseCoin must be between 0 and ",
            numberOfCoins-1)

    // Вставьте листинги 7.1–7.3 сюда

#####
# main
#####
if __name__ == '__main__':
    M = 8          # Максимальное количество доступных кубитов
    numberOfCoins = 4    # До M-1, где M – количество доступных кубитов
    indexOffalseCoin = 2    # Должен быть 0, 1... numberOfCoins – 1

    backend = "ibmq_qasm_simulator"
    #backend = "ibmqx3"
    shots = 1          # Мы проводим эксперимент с одним запуском

    main(M, numberOfCoins, indexOffalseCoin, backend, shots)

```

## Обобщенный алгоритм для любого количества фальшивых монет

Для загадки про фальшивую монету математики Терхал и Смолин в 1998 году создали обобщенный алгоритм для любого количества фальшивых монет ( $k > 1$ ). В их реализации используется модель «Б-оракул» («балансный оракул»), при этом:

- на вход поступает  $N$  бит  $x = x_1x_2...x_n \in \{0, 1\}^N$ ;
- создается строка запроса, состоящая из  $N$  троек таких битов, что  $q = q_1q_2...q_n \in \{0, 1, -1\}^N$ , с одинаковым количеством 1 и  $-1$ ;

○ ответом является один такой бит, что

$$\chi(x; q) = \begin{cases} 0 & \text{если } q_1x_1 + q_2x_2 + \dots + q_nx_n = 0 \text{ (весы уравновешены);} \\ 1 & \text{иначе (весы наклонены).} \end{cases}$$

### ПРИМЕЧАНИЕ

Оракул является частью алгоритма, рассматриваемой как черный ящик. Он используется для упрощения схем и сравнения сложности квантовых и классических алгоритмов. Хороший оракул должен быть быстрым, универсальным и легко реализуемым.

Пример применения Б-оракула для двух фальшивых монет с  $k = 2$  и  $N = 6$  приведен на рис. 7.4.

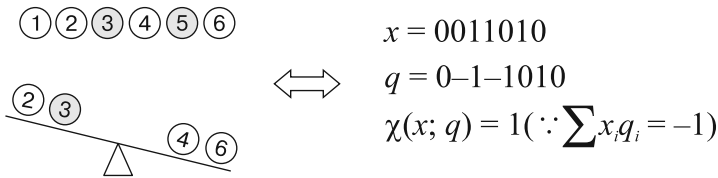


Рис. 7.4. Модель Б-оракула для  $k = 2$  и  $N = 6$

В общем, загадка о фальшивой монете — типичный пример ускорения квантового алгоритма по сравнению с классическим аналогом. В следующем разделе рассмотрим еще одну своеобразную псевдомагическую головоломку под названием «магический квадрат Мермина — Переса».

## Магический квадрат Мермина — Переса

Это еще одна классическая загадка, впервые предложенная физиками Д. Мермином и А. Пересом в качестве примера квантовой псевдотелепатии, или способности двух игроков общаться сверхъестественным об-

разом незаметно для наблюдателей. Это возможно благодаря волшебству запутывания. Рассмотрим ее подробнее.

Два игрока, Алиса и Боб, ведут игру против арбитра. Магический квадрат — это матрица размерностью  $3 \times 3$  со следующими правилами (рис. 7.5).

- Все элементы представлены либо 0, либо 1, так что сумма элементов в каждой строке четная, а в каждом столбце — нечетная. Игра называется магическим квадратом, потому что подобный квадрат невозможен. Как показано на рис. 7.5, не существует верной комбинации, где сумма строк четная, а столбцов — нечетная (возьмите бумагу и ручку и проверьте сами). Это обусловлено нечетным количеством элементов в матрице.

Арбитр отправляет целое число  $a \in \{1, 2, 3\}$  Алисе, а  $b \in \{1, 2, 3\}$  — Бобу. Ответом Алисы должна быть  $a$ -я строка квадрата, ответом Боба —  $b$ -й столбец.

- Алиса и Боб выигрывают, если сумма элементов Алисы четная, а Боба — нечетная и в пересечении их ответов находятся одинаковые значения. В противном случае побеждает арбитр.
- Перед стартом Алиса и Боб могут разработать стратегию и обмениваться информацией. Например, могут решить давать ответы при помощи матрицы, приведенной на рис. 7.5. Однако им не разрешается общаться во время игры.

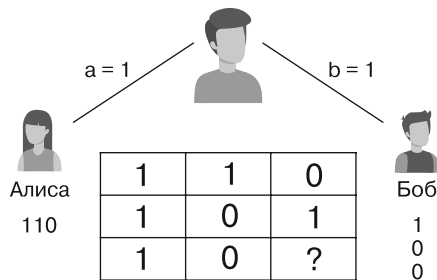


Рис. 7.5. Магический квадрат Мермина — Переса

К примеру, в приведенной ранее матрице, если бы арбитр отправил  $a = 1$  Алисе и  $b = 2$  — Бобу, ответом Алисы было бы 110 (первая строка), а Боба — 100 (второй столбец). Элементы, находящиеся на пересечении ответов (первой строки и второго столбца), одни и те же (1), поэтому Алиса и Боб выигрывают. Можно показать, что в классическом варианте максимальная вероятность победы для Алисы и Боба составляет  $8/9$ . То есть в данном квадрате восемь из девяти перестановок являются выигрышными. Поэтому максимальная вероятность победы Алисы и Боба —  $88,8\%$ .

Протестируем это утверждение на простом примере, чтобы доказать, что максимальная вероятность выигрыша для классической версии магического квадрата действительно составляет  $8/9$  ( $88,88\%$ ).

## Упражнение для магического квадрата Мермина — Переса

1. Создайте магический квадрат, аналогичный приведенному на рис. 7.5, используя бинарный код  $(1, -1)$  вместо  $(1, 0)$ , где произведение элементов строки 1 (четное), а произведение элементов столбца —  $-1$  (нечетное). Подтвердите, что это фактически невозможно.
2. Создайте таблицу перестановок для значений  $a$  и  $b$  арбитра при помощи квадрата из первого шага, включающую:
  - номер перестановки;
  - значения для  $a$  и  $b$ ;
  - ответы Алисы и Боба;
  - значения на пересечении ответов Алисы и Боба (помните, они должны быть одинаковыми, чтобы Алиса и Боб победили);
  - результат итерации игры: Win = W (победа), Loose = L (поражение).
3. Наконец, рассчитайте вероятность победы и докажите, что максимально она составляет  $8/9$ . (**Примечание:** ответ приведен в конце главы.)



## Квантовая стратегия победы

Благодаря мощи квантовой механики и магии запутывания результаты Алисы и Боба можно значительно улучшить. Фактически они могут выигрывать в 100 % случаев, как если бы общались телепатически (отсюда и термин «псевдотелепатия»). Квантовая стратегия победы впервые была предложена Brassardом и его коллегами, она подразделяется на три этапа.

1. *Общее запутанное состояние.* Это ключ к победам Алисы и Боба в 100 % случаев.
2. *Унитарные преобразования входных данных Алисы и Боба.* Так создаются ответы, которые отправляются арбитру.
3. *Измерения в вычислительном базисе.* Конечный этап создания окончательного ответа.

## Общее запутанное состояние

В квантовой стратегии победы у Алисы и Боба общее запутанное состояние:

$$\psi = \frac{1}{2}|0011\rangle - \frac{1}{2}|0110\rangle - \frac{1}{2}|1001\rangle + \frac{1}{2}|1100\rangle.$$

Для реализации схемы требуются два кубита для Алисы и два кубита для Боба (рис. 7.6).

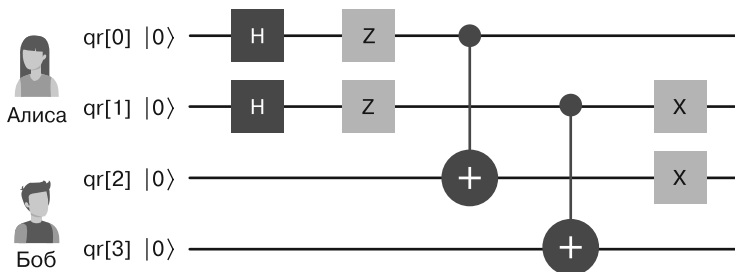


Рис. 7.6. Запутанное состояние для магического квадрата

- Известно, что преобразование Адамара сопоставляет базисному состоянию:

$$H|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

- Применение данного преобразования к первым двум кубитам дает:

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

- Затем к первым двум кубитам применяется вентиль Z. Помните, что он оставляет нулевое состояние неизменным и переводит 1 в -1, меняя на противоположный знак при третьем члене в приведенном ранее выражении. На данном этапе состояние принимает вид:

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

- Затем для запутывания кубитов 0-2 и 1-3 применяется вентиль CNOT:

$$\psi = \frac{1}{2}|0000\rangle - \frac{1}{2}|0101\rangle - \frac{1}{2}|1010\rangle + \frac{1}{2}|1111\rangle.$$

- Наконец, состояния двух последних кубитов меняются на противоположные:

$$\psi = \frac{1}{2}|0011\rangle - \frac{1}{2}|0110\rangle - \frac{1}{2}|1001\rangle + \frac{1}{2}|1100\rangle.$$

Скрипт Python для создания запутанного состояния приведен в листинге 7.5.

#### Листинг 7.5. Квантовая стратегия победы, запутанное состояние

```
# Создание запутанного состояния
Q_program = QuantumProgram()
Q_program.set_api(Qconfig.APIToken, Qconfig.config["url"])

# Четыре кубита (Alice = 2, Bob = 2)
N = 4

# Создание регистров
qr = Q_program.create_quantum_register("qr", N)

# для запоминания результата измерения на qr
cr = Q_program.create_classical_register("cr", N)
```

```

circuitName = "sharedEntangled"
sharedEntangled = Q_program.create_circuit(circuitName, [qr], [cr])

# Создание равновзвешенной суперпозиции всех строк длиной 2
for i in range(2):
    sharedEntangled.h(qr[i])

# Амплитуда отрицательная, если количество единиц нечетное
for i in range(2):
    sharedEntangled.z(qr[i])

# Копирование содержимого первых двух кубитов в последние два кубита
for i in range(2):
    sharedEntangled.cx(qr[i], qr[i+2])

# Изменение состояния последних двух кубитов на противоположное
for i in range(2,4):
    sharedEntangled.x(qr[i])

```

Теперь, когда у Алисы и Боба есть общее запутанное состояние, они могут начать игру и получить входные данные от арбитра.

## Унитарные преобразования

Получив свои входные данные  $a \in \{1, 2, 3\}$  и  $b \in \{1, 2, 3\}$ , Алиса и Боб применяют к общим запутанным состояниям следующие унитарные преобразования:  $A1, A2, A3$  для Алисы и  $B1, B2, B3$  — для Боба<sup>1</sup>.

$$\begin{aligned}
 A1 &= \frac{1}{\sqrt{2}} \begin{bmatrix} i & 0 & 0 & 1 \\ 0 & -i & 1 & 0 \\ 0 & i & 1 & 0 \\ 1 & 0 & 0 & i \end{bmatrix}; \quad A2 = \frac{1}{2} \begin{bmatrix} i & 1 & 1 & i \\ -i & 1 & -1 & i \\ i & 1 & 1 & -i \\ -i & 1 & -1 & -i \end{bmatrix}; \quad A3 = \frac{1}{2} \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 \end{bmatrix}; \\
 B1 &= \frac{1}{2} \begin{bmatrix} i & -i & 1 & 1 \\ -i & -i & 1 & -1 \\ 1 & 1 & -i & i \\ -i & i & 1 & 1 \end{bmatrix}; \quad B2 = \frac{1}{2} \begin{bmatrix} -1 & i & 1 & i \\ 1 & i & 1 & -i \\ 1 & -i & 1 & i \\ -1 & -i & 1 & -i \end{bmatrix}; \quad B3 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \end{bmatrix}.
 \end{aligned}$$

<sup>1</sup> Применяют только одно из них соответственно полученному от арбитра номеру. — *Примеч. науч. ред.*

**ПРИМЕЧАНИЕ**

Обратите внимание, что, применив приведенные ранее преобразования к своим запутанным состояниям, Алиса и Боб смогут создать первые два бита соответствующих ответов арбитру.

В листинге 7.6 показаны унитарные преобразования для Алисы и Боба, а соответствующие им графические схемы приведены в табл. 7.1.

**Листинг 7.6.** Унитарные преобразования для Алисы и Боба

```
#----- Схемы для операций, производимых Алисой и Бобом.
# Сначала определяем управляемые U-вентили, необходимые
# для назначения фаз
from math import pi
```

```
def ch(qProg, a, b):
    """ Controlled-Hadamard gate """
    qProg.h(b)
    qProg.sdg(b)
    qProg.cx(a, b)
    qProg.h(b)
    qProg.t(b)
    qProg.cx(a, b)
    qProg.t(b)
    qProg.h(b)
    qProg.s(b)
    qProg.x(b)
    qProg.s(a)
    return qProg
```

```
def cu1pi2(qProg, c, t):
    """ Controlled-u1(phi/2) gate """
    qProg.u1(pi/4.0, c)
    qProg.cx(c, t)
    qProg.u1(-pi/4.0, t)
    qProg.cx(c, t)
    qProg.u1(pi/4.0, t)
    return qProg
```

```
def cu3pi2(qProg, c, t):
    """ Controlled-u3(pi/2, -pi/2, pi/2) gate """
    qProg.u1(pi/2.0, t)
    qProg.cx(c, t)
    qProg.u3(-pi/4.0, 0, 0, t)
    qProg.cx(c, t)
    qProg.u3(pi/4.0, -pi/2.0, 0, t)
    return qProg
```

```

#-----
# Определение схем, которые Алиса и Боб используют
# для всех своих входных данных: 1, 2, 3
# Словарь для операций и схем Алисы
aliceCircuits = {}

# Квантовые схемы для входных данных Алисы 1, 2, 3
for idx in range(1, 4):
    circuitName = "Alice"+str(idx)
    aliceCircuits[circuitName]
        = Q_program.create_circuit(circuitName, [qr], [cr])
    theCircuit = aliceCircuits[circuitName]

    if idx == 1:
        # Схема для A_1
        theCircuit.x(qr[1])
        theCircuit.cx(qr[1], qr[0])
        theCircuit = cu1pi2(theCircuit, qr[1], qr[0])
        theCircuit.x(qr[0])
        theCircuit.x(qr[1])
        theCircuit = cu1pi2(theCircuit, qr[0], qr[1])
        theCircuit.x(qr[0])
        theCircuit = cu1pi2(theCircuit, qr[0], qr[1])
        theCircuit = cu3pi2(theCircuit, qr[0], qr[1])
        theCircuit.x(qr[0])
        theCircuit = ch(theCircuit, qr[0], qr[1])
        theCircuit.x(qr[0]) theCircuit.x(qr[1])
        theCircuit.cx(qr[1], qr[0])
        theCircuit.x(qr[1])

    elif idx == 2:
        theCircuit.x(qr[0])
        theCircuit.x(qr[1])
        theCircuit = cu1pi2(theCircuit, qr[0], qr[1])
        theCircuit.x(qr[0])
        theCircuit.x(qr[1])
        theCircuit = cu1pi2(theCircuit, qr[0], qr[1])
        theCircuit.x(qr[0])
        theCircuit.h(qr[0])
        theCircuit.h(qr[1])

    elif idx == 3:
        theCircuit.cz(qr[0], qr[1])
        theCircuit.swap(qr[0], qr[1]) # Не поддерживается в composer
        theCircuit.h(qr[0])
        theCircuit.h(qr[1])
        theCircuit.x(qr[0])
        theCircuit.x(qr[1])
        theCircuit.cz(qr[0], qr[1])
        theCircuit.x(qr[0])
        theCircuit.x(qr[1])

```

```

# Измерение первых двух кубитов в вычислительном базисе
theCircuit.measure(qr[0], cr[0])
theCircuit.measure(qr[1], cr[1])

# Словарь для операций и схем Боба
bobCircuits = {}

# Квантовые схемы для Боба при получении входных данных 1, 2, 3
for idx in range(1,4):
    circuitName = "Bob"+str(idx)
    bobCircuits[circuitName]
        = Q_program.create_circuit(circuitName, [qr], [cr])
    theCircuit = bobCircuits[circuitName]
    if idx == 1:
        theCircuit.x(qr[2])
        theCircuit.x(qr[3])
        theCircuit.cz(qr[2], qr[3])
        theCircuit.x(qr[3])
        theCircuit.u1(pi/2.0, qr[2])
        theCircuit.x(qr[2])
        theCircuit.z(qr[2])
        theCircuit.cx(qr[2], qr[3])
        theCircuit.cx(qr[3], qr[2])
        theCircuit.h(qr[2])
        theCircuit.h(qr[3])
        theCircuit.x(qr[3])
        theCircuit = cu1pi2(theCircuit, qr[2], qr[3])
        theCircuit.x(qr[2])
        theCircuit.cz(qr[2], qr[3])
        theCircuit.x(qr[2])
        theCircuit.x(qr[3])

    elif idx == 2:
        theCircuit.x(qr[2])
        theCircuit.x(qr[3])
        theCircuit.cz(qr[2], qr[3])
        theCircuit.x(qr[3])
        theCircuit.u1(pi/2.0, qr[3])
        theCircuit.cx(qr[2], qr[3])
        theCircuit.h(qr[2])
        theCircuit.h(qr[3])

    elif idx == 3:
        theCircuit.cx(qr[3], qr[2])
        theCircuit.x(qr[3])
        theCircuit.h(qr[3])

# Измерение 3-го и 4-го кубитов в вычислительном базисе
theCircuit.measure(qr[2], cr[2])
theCircuit.measure(qr[3], cr[3])

```

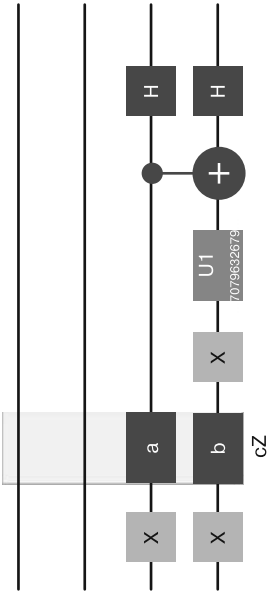
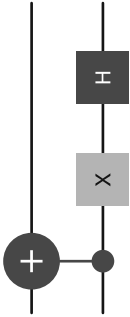
В табл. 7.1 показаны квантовые схемы для унитарных преобразований  $A1-2$  и  $B1-3$  для IBM Q Experience Composer.

**Таблица 7.1.** Квантовые схемы для унитарных преобразований из листинга 7.6

Преобразование	Схема
$A1 = \frac{1}{\sqrt{2}} \begin{bmatrix} i & 0 & 0 & 1 \\ 0 & -i & 1 & 0 \\ 0 & i & 1 & 0 \\ 1 & 0 & 0 & i \end{bmatrix}$	
$A2 = \frac{1}{2} \begin{bmatrix} i & 1 & 1 & i \\ -i & 1 & -1 & i \\ i & 1 & 1 & -i \\ -i & 1 & -1 & -i \end{bmatrix}$	
$B1 = \frac{1}{2} \begin{bmatrix} i & -i & 1 & 1 \\ 1 & -i & -i & -1 \\ 1 & 1 & -i & i \\ -i & i & 1 & 1 \end{bmatrix}$	

Продолжение ⇨

Таблица 7.1 (продолжение)

Преобразование	Схема
$B2 = \frac{1}{2} \begin{bmatrix} -1 & i & 1 & i \\ 1 & i & 1 & -i \\ 1 & -i & 1 & i \\ -1 & -i & 1 & -i \end{bmatrix}$	
$B3 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \end{bmatrix}$	



Обратите внимание, что в табл. 7.1 не включено преобразование A3 ввиду того, что Composer не поддерживает вентиль SWAP, который необходим для кода из листинга 7.6. Тем не менее это не значит, что квантовую программу нельзя запустить на моделирующем или реальном устройстве. Это просто означает, что схема не будет создана в Composer. Поэтому на последнем шаге Алиса и Боб измеряют свои кубиты в вычислительном базисе.

## Измерение в вычислительном базисе

После измерения у Алисы и Боба есть по два бита, которые представляют результаты измерений. Чтобы получить последний бит и, таким образом, окончательный ответ, они применяют свои правила проверки четности. То есть у Алисы сумма должна быть четной, а у Боба — нечетной, например для  $a = 2$ , для  $b = 3$  (табл. 7.2):

$$(A2 \otimes B3)|\psi\rangle = \frac{1}{2\sqrt{2}} \times \\ \times [ |0000\rangle - |0010\rangle - |0101\rangle + |0111\rangle + |1001\rangle - |1011\rangle - |1100\rangle - |1110\rangle ].$$

**Таблица 7.2.** Перестановки ответов для магического квадрата при  $a = 2$ ,  $b = 3$

$\psi$	Ответ Алисы	Ответ Боба	Квадрат
$ 0000\rangle$	000	001	$\begin{bmatrix} & 0 \\ 0 & 0 & 0 \\ & 1 \end{bmatrix}$
$ 0010\rangle$	000	100	$\begin{bmatrix} & 1 \\ 0 & 0 & 0 \\ & 0 \end{bmatrix}$
$ 0101\rangle$	011	010	$\begin{bmatrix} & 1 \\ 0 & 1 & 1 \\ & 0 \end{bmatrix}$
$ 0111\rangle$	011	111	$\begin{bmatrix} & 1 \\ 0 & 1 & 1 \\ & 1 \end{bmatrix}$

Продолжение ⇨

**Таблица 7.2** (продолжение)

$\psi$	Ответ Алисы	Ответ Боба	Квадрат
$ 1001\rangle$	101	010	$\begin{bmatrix} & 0 \\ 1 & 0 & 1 \\ & 0 \end{bmatrix}$
$ 1011\rangle$	101	111	$\begin{bmatrix} & 1 \\ 1 & 0 & 1 \\ & 1 \end{bmatrix}$
$ 1100\rangle$	110	001	$\begin{bmatrix} & 0 \\ 1 & 1 & 0 \\ & 1 \end{bmatrix}$
$ 1110\rangle$	110	101	$\begin{bmatrix} & 1 \\ 1 & 1 & 0 \\ & 1 \end{bmatrix}$

В листинге 7.7 показан фрагмент скрипта для прохода в цикле по всем раундам магического квадрата.

- Проход в цикле по  $a[1, 3]$  и  $b[1, 3]$  включительно.
- Для каждой  $(a, b)$  схемы Алисы (Alice-a) и Боба (Bob-b) возвращаются в исходное состояние из листинга 7.6.
- Общее запутанное состояние  $\psi$  и схемы Алисы и Боба загружаются для запуска на моделирующее или реальное квантовое устройство.
- Для Алисы и Боба из ответа извлекается по два бита, например  $\{ '0011': 1 \}$ .
- Применение правил проверки четности: у Алисы сумма должна быть четной, а у Боба — нечетной.
- В конце ответ проверяется на корректность и отображается вероятность победы.

**Листинг 7.7.** Скрипт для всех раундов магического квадрата

```
def all_rounds(backend, real_dev, shots=10):
    nWins = 0
    nLost = 0
    for a in range(1,4):
        for b in range(1,4):
```

---

```

print("Asking Alice and Bob with a and b are: ", a,b)
rWins = 0
rLost = 0

aliceCircuit = aliceCircuits["Alice" + str(a)]
bobCircuit = bobCircuits["Bob" + str(b)]
circuitName = "Alice" + str(a) + "Bob"+str(b)
Q_program.add_circuit(circuitName,
sharedEntangled+aliceCircuit+
bobCircuit)

if real_dev:
    ibmqx2_backend = Q_program.get_backend_
    configuration(backend)
    ibmqx2_coupling = ibmqx2_backend['coupling_map']
    results = Q_program.execute([circuitName],
    backend=backend, shots=shots
    , coupling_map=ibmqx2_coupling, max_credits=3,
    wait=10, timeout=240)
else:
    results = Q_program.execute([circuitName],
    backend=backend, shots=shots)

answer = results.get_counts(circuitName)
for key in answer.keys():
    kfreq = answer[key]
    # Частоты появления ключей, полученные при измерениях
    aliceAnswer = [int(key[-1]), int(key[-2])]
    bobAnswer = [int(key[-3]), int(key[-4])]
    if sum(aliceAnswer) % 2 == 0:
        aliceAnswer.append(0)
    else:
        aliceAnswer.append(1)
    if sum(bobAnswer) % 2 == 1:
        bobAnswer.append(0)
    else:
        bobAnswer.append(1)

    if(aliceAnswer[b-1] != bobAnswer[a-1]):
        #print(a, b, "Alice and Bob lost")
        nLost += kfreq
        rLost += kfreq
    else:
        #print(a, b, "Alice and Bob won")
        nWins += kfreq
        rWins += kfreq
print("\t#wins = ", rWins, "out of ", shots, "shots")

print("Number of Games = ", nWins+nLost)
print("Number of Wins = ", nWins)
print("Winning probabilities = ", (nWins*100.0)/(nWins+nLost))

```

```
#####
# main
##### if __name__ == '__main__':
    backend = "ibmq_qasm_simulator"
    #backend = "ibmqx2"
    real_dev = False

    all_rounds(backend, real_dev)
```

Запуск листинга 7.7 на удаленном моделирующем устройстве IBM Q Experience показан в листинге 7.8.

**Листинг 7.8.** Упрощенный стандартный вывод для запуска всех раундов магического квадрата

```
c:\python36-64\python.exe p_magicsq.py
For a = 1, b = 1
ibmq_qasm_simulator answer: 1000 Alice: [0, 0, 0] Bob:[0, 1, 0]
ibmq_qasm_simulator answer: 1010 Alice: [0, 1, 1] Bob:[0, 1, 0]
ibmq_qasm_simulator answer: 1111 Alice: [1, 1, 0] Bob:[1, 1, 1]
ibmq_qasm_simulator answer: 0111 Alice: [1, 1, 0] Bob:[1, 0, 0]
ibmq_qasm_simulator answer: 0000 Alice: [0, 0, 0] Bob:[0, 0, 1]
ibmq_qasm_simulator answer: 0101 Alice: [1, 0, 1] Bob:[1, 0, 0]
# 10 побед при 10 запусках
For a = 1, b = 2
ibmq_qasm_simulator answer: 1000 Alice: [0, 0, 0] Bob:[0, 1, 0]
ibmq_qasm_simulator answer: 1001 Alice: [1, 0, 1] Bob:[0, 1, 0]
ibmq_qasm_simulator answer: 1111 Alice: [1, 1, 0] Bob:[1, 1, 1]
ibmq_qasm_simulator answer: 0110 Alice: [0, 1, 1] Bob:[1, 0, 0]
ibmq_qasm_simulator answer: 0000 Alice: [0, 0, 0] Bob:[0, 0, 1]
ibmq_qasm_simulator answer: 0001 Alice: [1, 0, 1] Bob:[0, 0, 1]
# 10 побед при 10 запусках
...
For a = 3, b = 3
ibmq_qasm_simulator answer: 1000 Alice: [0, 0, 0] Bob:[0, 1, 0]
ibmq_qasm_simulator answer: 1011 Alice: [1, 1, 0] Bob:[0, 1, 0]
ibmq_qasm_simulator answer: 1101 Alice: [1, 0, 1] Bob:[1, 1, 1]
ibmq_qasm_simulator answer: 1110 Alice: [0, 1, 1] Bob:[1, 1, 1]
ibmq_qasm_simulator answer: 0111 Alice: [1, 1, 0] Bob:[1, 0, 0]
ibmq_qasm_simulator answer: 0010 Alice: [0, 1, 1] Bob:[0, 0, 1]
# 10 побед при 10 запусках
Number of Games = 90
Number of Wins = 90
Winning probability = 100.0
```

### ПРИМЕЧАНИЕ

При запуске на реальном устройстве вероятность выигрыша не будет 100%-ной из-за шумов окружающей среды и ошибок вентиляей.

## Ответы для упражнения с магическим квадратом

1. Магический квадрат, в котором произведение строки четное, а произведение столбца — нечетное, приведен далее. На самом деле такого квадрата не бывает ввиду нечетного числа клеток.

-1	-1	1
-1	1	-1
-1	1	?

2. Таблица перестановок для квадрата из первого ответа.

N	a	b	Алиса	Боб	Пересечение	Победа/ поражение (W/L)
1	1	1	-1, -1, 1	-1, -1, -1	-1/-1	W
2	1	2	-1, -1, 1	-1, 1, 1	-1/-1	W
3	1	3	-1, -1, 1	1, -1, ? (1)	1/1	W
4	2	1	-1, 1, -1	-1, -1, -1	-1/-1	W
5	2	2	-1, 1, -1	-1, 1, 1	1/1	W
6	2	3	-1, 1, -1	1, -1, ? (1)	-1/-1	W
7	3	1	-1, 1, ? (-1)	-1, -1, -1	-1/-1	W
8	3	2	-1, 1, ? (-1)	-1, 1, 1	1/1	W
9	3	3	-1, 1, ? (-1)	1, -1, ? (1)	-1/1	L

3. Обратите внимание, что на предыдущем шаге в строках 7–9 ответ Алисы должен быть -1, поэтому произведение может быть четным (1). К тому же в столбцах 3, 6 и 9 ответ Боба должен быть 1, в связи с чем произведение может быть нечетным (-1). Наконец, вероятность рассчитывается делением общего числа побед на общее количество перестановок. Таким образом:

$$P = \frac{\Sigma W}{N} = \frac{8}{9} = 88,88\%.$$

Из этой главы вы узнали, как квантовая запутанность может обеспечить значительное ускорение по сравнению с классическими вычислениями. Для квантовых весов можно добиться ускорения четвертой степени при

решении классических головоломок, таких как загадка про фальшивую монету. В других, таких как магический квадрат, запутывание наделяет игроков псевдомагической телепатией. Если бы Брассард и его коллеги смогли придумать квантовую стратегию выигрыша в блек-джек или покер, мы бы точно сорвали большой куш в Вегасе. В целом эта глава показала, что квантовая механика столь же запутанная, причудливая и увлекательная, как и всегда. Она никогда не разочаровывает.

В следующей, последней главе рассказывается о самом, пожалуй, известном квантовом алгоритме — пресловутой факторизации целых чисел Шора. Это алгоритм, который может нанести сокрушительный удар по асимметричному шифрованию!

# 8

## Алгоритмы Гровера и Шора: ускоренный поиск и угроза основам асимметричного шифрования

В этой главе мы завершим свои изыскания рассмотрением двух алгоритмов, которые вызвали волнение по поводу возможностей практических квантовых вычислений.

- *Алгоритм поиска Гровера.* Это неструктурированный алгоритм квантового поиска, созданный Ловом Гровером. Он способен с высокой вероятностью определять входные данные с помощью функции — черного ящика или оракула. Алгоритм позволяет найти элемент за  $O(\sqrt{N})$  шагов, тогда как среднее значение для классического аналога —  $N/2$  шагов.
- *Алгоритм факторизации целых чисел Шора.* Печально известная квантовая факторизация, которая, по словам экспертов, сможет поставить на колени современное асимметричное шифрование. По алгоритму Шора целые числа можно факторизовать приблизительно за  $\log(n^3)$  шагов, в отличие от наиболее быстрого классического алгоритма — метода решета числового поля, которому требуется  $\exp(k \log(n^{1/3})(\log \log n)^{2/3})$  шагов.

Давайте начнем.

## Квантовый неструктурированный поиск

Алгоритм Гровера — это неструктурированная процедура квантового поиска, предназначенная для нахождения строки из  $n$  бит в числовом «стоге сена» из  $N$  элементов. Как показано на рис. 8.1, квантовый алгоритм Гровера дает значительное ускорение  $O(\sqrt{N})$  шагов. Может показаться, что это не так уж и много в сравнении с классическим решением, но, когда речь идет о миллионах строк, квадратный корень из  $10^6$  намного меньше, чем  $10^6$ .



**Рис. 8.1.** Временная сложность неструктурированного поиска

Если  $x$  — элемент, который мы ищем, то алгоритм Гровера можно описать с помощью следующего псевдокода.

1. Подготовка входных данных при заданной функции (оракула)  $f: \{0, 1 \dots N-1\} \rightarrow \{0, 1\}$ . Обратите внимание, что размер ввода равен  $2^n$ , где  $n$  — это количество битов, а  $N$  — количество шагов, или размер «стога сена». Конечная цель состоит в том, чтобы найти  $x$ , при котором  $f(x) = 1$ .
2. Приведение всех входных кубитов в состояние суперпозиции.
3. Выполнение инверсии фаз на входных кубитах.
4. Выполнение инверсии относительно среднего значения на входе.
5. Повторение шагов 3 и 4 минимум  $\sqrt{N}$  раз. Существует большая вероятность того, что  $x$  будет найден на этой стадии.



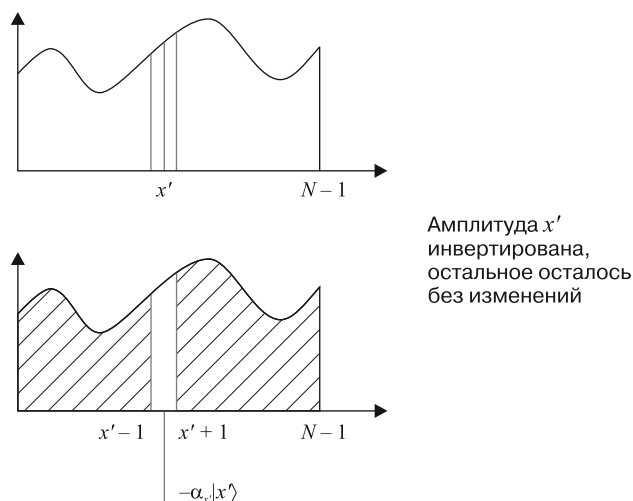
Давайте внимательнее рассмотрим критические шаги фазовой инверсии и инверсии относительно среднего значения.

## Фазовая инверсия

Это первый шаг в алгоритме, и он должен выполняться в суперпозиции всех состояний в «стоге сена». Если искомый элемент равен  $x'$ , где  $f(x') = 1$ , то суперпозицию можно выразить как  $\sum \alpha_x |x\rangle$ . В конечном счете фазовая инверсия выполняет следующее:

$$\sum \alpha_x |x\rangle \rightarrow -\alpha_{x'} |x'\rangle + \sum_{x \neq x'} \alpha_x |x\rangle$$

То есть если данный  $x$  не является искомым элементом ( $x \neq x'$ ), то он оставляет суперпозицию без изменений. В противном случае он инвертирует фазу (знак «минус» перед комплексным коэффициентом  $\alpha_{x'} |x'\rangle$  кубита, см. графическое представление на рис. 8.2).



**Рис. 8.2.** Графическое представление фазовой инверсии

Это первый шаг в алгоритме Гровера: мы увидим, как инверсия фазы помогает найти искомый элемент, но сейчас рассмотрим второй шаг — инверсию относительно среднего значения.

## Инверсия относительно среднего значения

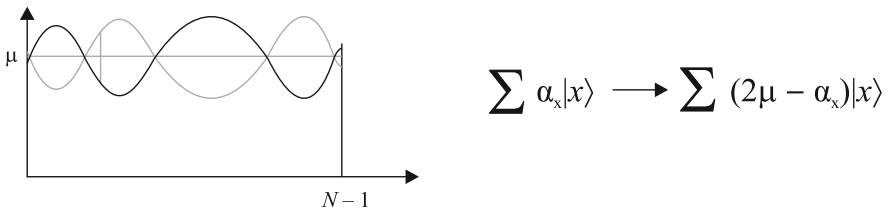
С учетом приведенной ранее суперпозиции  $\sum \alpha_x |x\rangle$  определяем среднее значение  $\mu$  как среднее значение амплитуд:

$$\mu = \frac{\sum_{x=0}^{N-1} \alpha_x}{N}.$$

Теперь мы должны зеркально отобразить амплитуды относительно этого среднего значения, то есть:

$$\begin{aligned} \alpha_x &\rightarrow (2\mu - \alpha_x); \\ \sum \alpha_x |x\rangle &\rightarrow \sum (2\mu - \alpha_x) |x\rangle. \end{aligned}$$

Чтобы можно было лучше понять это, на рис. 8.3 приведено графическое представление инверсии относительно среднего значения.



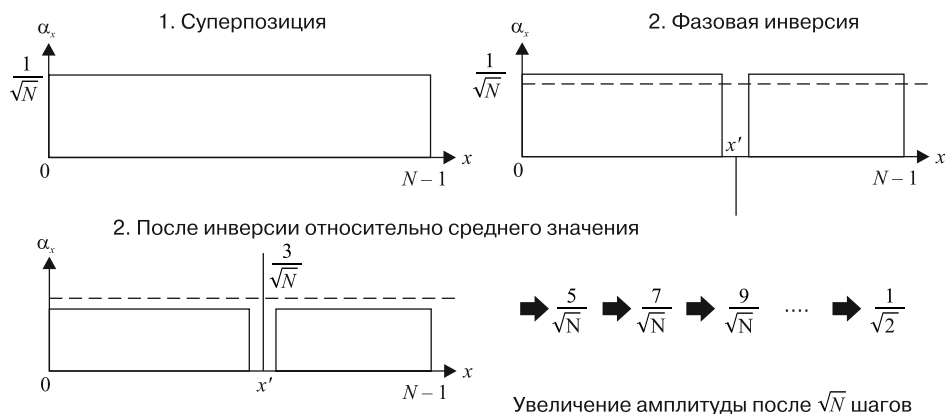
**Рис. 8.3.** Графическое представление инверсии относительно среднего значения

Здесь показано полученное методом суперпозиции состояние кубитов, которое определяется как волновая функция  $\psi$ . Среднее значение  $\mu$  этой функции на графике представлено горизонтальной линией. Инверсия относительно среднего значения выполняет зеркальное отражение волны (показана пунктирной линией). Это эквивалентно повороту волны относительно оси  $\mu$ . Давайте разберемся во всем этом, объединив все шаги, чтобы увидеть их в действии.

На рис. 8.4 показано следующее.

Суперпозиция всех кубитов переводит все амплитуды в  $\frac{1}{\sqrt{N}}$ .

Затем фазовая инверсия переводит амплитуду для  $x'$  в  $-\frac{1}{\sqrt{N}}$ . Обратите внимание, что здесь есть эффект небольшого снижения среднего значения  $\mu$  (см. рис. 8.4, шаг 2).



**Рис. 8.4.** Одна итерация алгоритма Гровера

После инверсии относительно среднего значения средняя амплитуда немного уменьшится, но  $x'$  так же, как и  $\frac{2}{\sqrt{N}}$ , поднимется выше среднего значения  $\mu$ .

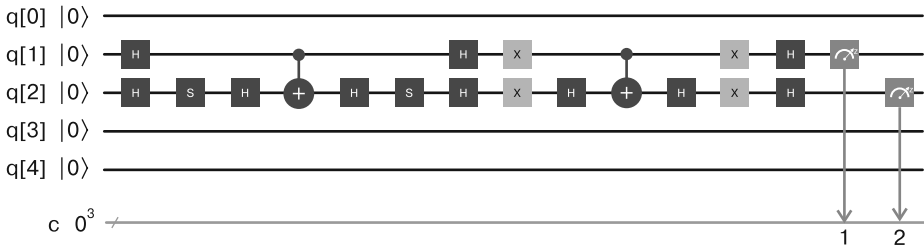
Если повторить эту последовательность шагов, то амплитуда  $x'$  будет увеличиваться примерно на  $\frac{2}{\sqrt{N}}$ , пока после приблизительно  $\sqrt{N}$  шагов не станет равной  $\frac{1}{\sqrt{2}}$ .

- В этот момент, если мы измерим наши кубиты, вероятность нахождения  $x'$  (искомого элемента) согласно принципам квантовой механики равна квадрату амплитуды, что составляет  $1/2$ .
- Таким образом, все готово. Примерно за  $\sqrt{N}$  шагов мы нашли отмеченный элемент  $x'$ .

Теперь соберем все это в квантовой схеме с соответствующей реализацией в коде.

## Практическая реализация

Рассмотрим схему для алгоритма Гровера в IBM Q Experience. Она демонстрирует одну итерацию алгоритма при помощи двух кубитов (рис. 8.5).



**Рис. 8.5.** Квантовая схема для алгоритма Гровера с двумя кубитами и  $A = 01$

Скрипт Python для создания схемы, изображенной на рис. 8.5, дан в листинге 8.1.

**Листинг 8.1.** Скрипт Python для схемы, приведенной на рис. 8.5

```
import sys,time,math
# Импорт QISKit
from qiskit import QuantumCircuit, QuantumProgram

# Конфигурация Q Experience
sys.path.append('../Config/')
import Qconfig

# Импорт базовых средств отображения графики
from qiskit.tools.visualization import plot_histogram

# Установка значений входных битов, которые будут использоваться для поиска
def input_phase (circuit, qubits):
    # Раскомментировать при A = 00
    # Закомментировать при A = 11
    circuit.s(qubits[0])
    #circuit.s(qubits[1])
    return

# circuit – двухкубитная схема для алгоритма Гровера
# qubits – массив кубитов (размером 2)
def invert_over_the_mean (circuit, qubits):
    for i in range (2):
        circuit.h(qubits[i])
        circuit.x(qubits[i])

    circuit.h(qubits[1])
    circuit.cx(qubits[0], qubits[1])
    circuit.h(qubits[1])

    for i in range (2):
        circuit.x(qubits[i])
        circuit.h(qubits[i])
```

```
def invert_phase (circuit, qubits):
    # Оракул
    circuit.h(qubits[1])
    circuit.cx(qubits[0], qubits[1])
    circuit.h(qubits[1])

def main():
    # Настройка квантовой программы
    qp = QuantumProgram()

    qp.set_api(Qconfig.APIToken, Qconfig.config["url"])

    # Создание кубитов/регистров
    size = 2
    q = qp.create_quantum_register('q', size)
    c = qp.create_classical_register('c', size)

    # Квантовая схема
    grover = qp.create_circuit('grover', [q], [c])

    # 1. Перевод всех кубитов в суперпозицию
    for i in range (size):
        grover.h(q[i])

    # Настройка входных данных
    input_phase(grover, q)

    # 2. Фазовая инверсия
    invert_phase(grover, q)

    input_phase(grover, q)

    # 3. Инверсия относительно среднего значения
    invert_over_the_mean (grover, q)

    # Измерение
    for i in range (size):
        grover.measure(q[i], c[i])

    circuits = ['grover']

    # Выполнение квантовой схемы на моделирующем устройстве
    backend = "local_qasm_simulator"
    # Количество запусков эксперимента
    shots = 1024

    result = qp.execute(circuits, backend=backend, shots=shots
        , max_credits=3, timeout=240)
    counts = result.get_counts("grover")
    print("Counts:" + str(counts))
```

```

# Необязательно
#plot_histogram(counts)

#####
# main
if __name__ == '__main__':
    start_time = time.time()
    main()
    print("--- %s seconds ---" % (time.time() - start_time))

```

В листинге 8.1 выполняется одна итерация алгоритма Гровера с двумя кубитами на входе и использованием двух кубитов. И хотя в предыдущем разделе утверждается, что общее количество итераций задается приблизительно  $\sqrt{N}$  шагами, для инверсии относительно среднего значения необходимо это число умножить на  $\pi/4$  и округлить в меньшую сторону (floor) (см. доказательство на рис. 8.8). Следовательно, мы получаем  $IT = \text{floor}\left(\sqrt{N} \cdot \frac{\pi}{4}\right)$ , где  $N = 2^{\text{bits}}$ . Тогда для двух бит  $IT = \text{floor}\left(\sqrt{4} \cdot \frac{\pi}{4}\right) = \text{floor}(1,57) = 1$ .

- Скрипт начинается с создания квантовой схемы с двумя кубитами и двумя классическими регистрами для хранения результатов измерения на них.
- Затем все кубиты переводятся в суперпозицию при помощи вентиля Адамара.
- Прежде чем выполнить итерацию, подготавливаются входные данные с использованием фазового вентиля (S) и правил из табл. 8.1.

**Таблица 8.1.** Правила подготовки входных данных для листинга 8.1

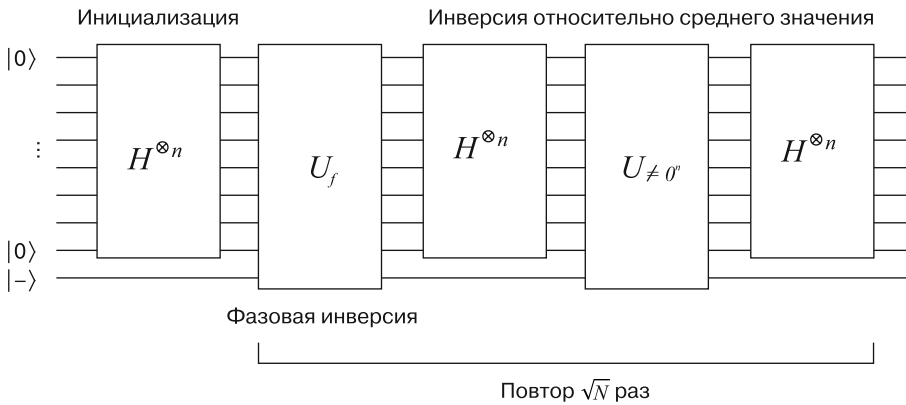
Входные данные (A)	Вентили/кубиты
00	S(01)
10	S(0)
01	S(1)
11	Ничего

- Затем выполняется инверсия фаз, а следом за ней — инверсия относительно среднего значения входных кубитов в соответствии с одной итерацией алгоритма.

- Наконец, измеряются результаты и схема выполняется на локальном или удаленном моделирующем устройстве. Выводятся полученные численные значения.

## Обобщенная схема

В целом схему, представленную на рис. 8.5, можно обобщить для любого количества входных кубитов (рис. 8.6).



**Рис. 8.6.** Обобщенная форма алгоритма Гровера для произвольного числа кубитов

- Первый блок на рис. 8.6 переводит все кубиты в суперпозицию применением вентиля Адамара к входным данным размера  $n$ . Это шаг инициализации.

Затем в схему фазовой инверсии ( $U_f$ ) подаются входные данные в суперпозиции  $\psi = \sum \alpha |x\rangle$  и входные значения фаз (знак «минус»). Это позволяет установить желаемую фазу именно там, где нужно. Таким образом, получены выходные данные  $\sum \alpha (-1)^{f(x)} |x\rangle$ . Но как можно достичь такого результата? Ответ: желаемый эффект  $|b\rangle \rightarrow |f(x) \oplus b\rangle$  был получен с помощью исключающего ИЛИ к входному состоянию со знаком «минус» (рис. 8.7). В третьей строке таблицы истинности для исключающего ИЛИ между  $f(x)$  и  $b$  (в правой части рис. 8.7) показан эффект применения фазовой инверсии.

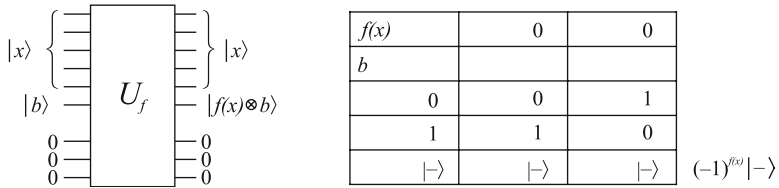


Рис. 8.7. Схема для фазовой инверсии

Наконец, как показано на рис. 8.3, инверсия относительно среднего значения — то же самое, что и отражение относительно  $|\mu\rangle = 1/\sqrt{N} \sum_x |x\rangle$ . Для лучшего понимания полученное методом суперпозиции состояние  $\psi$  и среднее значение  $\mu$  могут быть представлены в виде векторов в двумерном пространстве (см. рис. 8.8). Чтобы отразить  $\psi$ , создайте вектор, ортогональный к  $\mu$ , а затем — проекцию  $\psi$  в новый квадрант под тем же углом  $\theta$ .

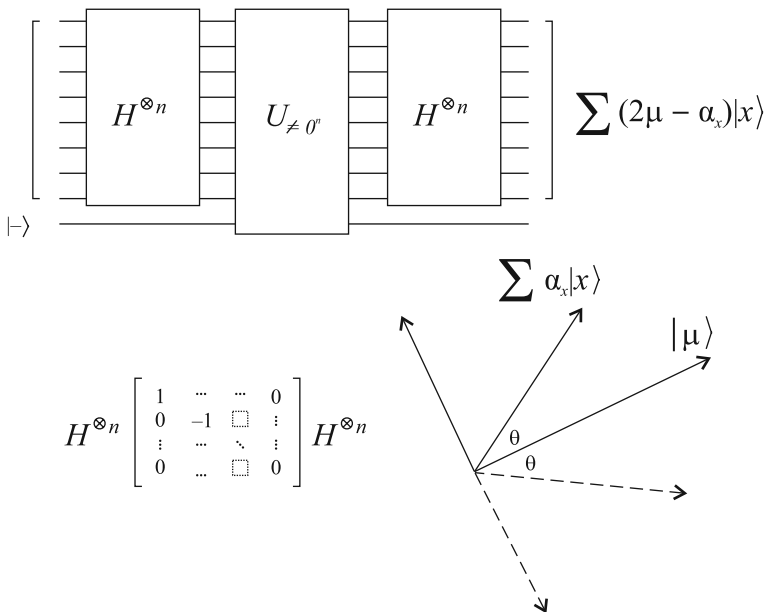


Рис. 8.8. Схема для инверсии относительно среднего значения

Доказательство того, что инверсия относительно среднего значения выполняет преобразование  $\sum \alpha_x |x\rangle \rightarrow \sum (2\mu - \alpha_x) |x\rangle$ , состоит из трех этапов, которые приведены на схеме на рис. 8.8.



1. Преобразование  $|\mu\rangle$  в вектор, состоящий из нулей,  $|0\dots 0\rangle$ . Это достигается применением вентилля Адамара к входным данным.
2. Отражение относительно вектора из нулей  $|0\dots 0\rangle$ . Это можно сделать, умножив его на разреженную матрицу 
$$\begin{bmatrix} 1 & \dots & \dots & 0 \\ 0 & -1 & & \vdots \\ \vdots & \dots & \ddots & \vdots \\ 0 & \dots & & -1 \end{bmatrix}.$$

3. Обратное преобразование  $|0\dots 0\rangle$  в  $|\mu\rangle$  еще одним применением вентилля Адамара. Тогда:

$$\begin{aligned} H^{\otimes n} \begin{bmatrix} 1 & \dots & \dots & 0 \\ 0 & -1 & & \vdots \\ \vdots & \dots & \ddots & \vdots \\ 0 & \dots & & -1 \end{bmatrix} H^{\otimes n} &= H^{\otimes n} \left( \begin{bmatrix} 2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} - I \right) H^{\otimes n} = \\ &= H^{\otimes n} \begin{bmatrix} 2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} H^{\otimes n} - H^{\otimes n} I H^{\otimes n} = \\ &= \begin{bmatrix} 2/N & \dots & 2/N \\ \vdots & \ddots & \vdots \\ 2/N & \dots & 2/N \end{bmatrix} - I = \begin{bmatrix} (2/N)-1 & \dots & 2/N \\ \vdots & \ddots & \vdots \\ 2/N & \dots & (2/N)-1 \end{bmatrix}. \end{aligned} \quad (8.1)$$

Обратите внимание, что  $H^{\otimes n} I H^{\otimes n} = I$  и  $H = \frac{2}{\sqrt{N}} |x\rangle$ . Наконец, применение матрицы (8.1) к состоянию  $\psi = \sum \alpha_x |x\rangle$  дает

$$\begin{bmatrix} (2/N)-1 & \dots & 2/N \\ \vdots & \ddots & \vdots \\ 2/N & \dots & (2/N)-1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_x \\ \vdots \\ \alpha_{N-1} \end{bmatrix} \rightarrow \begin{bmatrix} \dots \\ \frac{2 \sum \alpha_y}{N} - \alpha_x \\ \dots \end{bmatrix} = 2\mu - \alpha_x,$$

где  $\frac{2 \sum \alpha_y}{N} = 2\mu$ .

Таким образом, получен алгоритм Гровера для неструктурированного поиска. Он быстрый, мощный и скоро будет повсюду использоваться в центрах обработки данных для ускорения любых видов поиска в базах данных.

Поскольку он значительно производительнее классического собрата, существует вероятность того, что через несколько лет, когда квантовые компьютеры станут более доступными для коммерческого применения, большая часть поисков в Интернете будет выполняться при помощи этой квантовой рабочей лошади. Прежде чем мы закончим, стоит отметить, что на момент написания книги какой-либо полезной реализации или эксперимента, который может найти что-то реальное, для IBM Q Experience не существует. Надеюсь, это изменится в будущем, но пока алгоритм Гровера хорошо работает в теории. В следующем разделе мы эффектно завершим книгу рассмотрением знаменитого алгоритма Шора для факторизации целых чисел.

## **Факторизация целых чисел при помощи алгоритма Шора**

Игра в кошки-мышки между криптографией и криптоанализом продолжается: первая разрабатывает новые способы шифрования наших ежедневно создаваемых данных, а второй ищет слабые места, чтобы их взломать. Современное асимметричное шифрование опирается на хорошо известную сложность факторизации очень больших простых чисел (в диапазоне сотен цифр). В этом разделе рассматривается внутренняя работа алгоритма Шора — метода, который дает экспоненциальное ускорение для факторизации целых чисел при помощи квантового компьютера. Затем поговорим о реализации с использованием библиотеки под названием ProjectQ. Далее мы смоделируем целочисленные выборки и оценим результаты. Наконец, рассмотрим текущие и перспективные направления факторизации целых чисел в квантовых системах.

## **Квантовая факторизация бросает вызов асимметричному шифрованию**

В основополагающей статье *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer* Питер Шор предложил метод квантовой факторизации, использующий принцип, давно известный математикам: найти период (также известный как порядок) элемента  $a$

в мультипликативной группе по модулю  $N$ , то есть такое наименьшее положительное целое число, что:

$$a^r \equiv 1 \pmod{N},$$

где  $N$  — число, раскладываемое на множители;  $r$  — период  $a$  по модулю  $N$ .

### ПРИМЕЧАНИЕ

Факторизация больших целых чисел — это задача, которая занимала умы математиков на протяжении тысячелетий. В 1976 году Г. Л. Миллер предположил, что с помощью случайных чисел факторизацию можно свести к нахождению периода элемента по модулю  $N$ , что значительно упрощает данную задачу. Это основная идея алгоритма Шора.

Ученый разделил свой алгоритм на три этапа, два из которых выполняются на классическом компьютере за полиномиальное время.

1. *Подготовка входных данных.* Выполняется на классическом компьютере за полиномиальное время  $\log(N)$ .
2. Нахождение периода  $r$  элемента  $a$ , при котором  $a^r \equiv 1 \pmod{N}$ , посредством квантовой схемы. Согласно Шору, для этого потребуется  $O((\log N)^2 (\log \log N) (\log \log \log N))$  шагов на квантовом компьютере.
3. *Обработка выходных данных.* Выполняется на классическом компьютере за полиномиальное время  $\log(N)$ .

Почему этот метод вызывает такую обеспокоенность? Сравните его временную сложность ( $O$ ) с этим же параметром действующего классического чемпиона — метода решета числового поля. Данные приведены в табл. 8.2 (включает еще одного лидера — популярный алгоритм квадратичного решета).

**Таблица 8.2.** Временная сложность популярных алгоритмов факторизации

Алгоритм	Временная сложность
Шора	$(\log N)^2 (\log \log N) (\log \log \log N)$
Решета числового поля	$\exp\left(c (\log N)^{1/3} (\log \log N)^{2/3}\right)$
Квадратичного решета	$\exp\left(\sqrt{\ln N \ln \ln N}\right)$

Невероятно, но алгоритм Шора имеет полиномиальную временную сложность, намного превосходящую экспоненциальное время алгоритма решета числового поля — самого быстрого известного метода факторизации на классическом компьютере. На самом деле эксперты подсчитали, что алгоритм Шора может разложить на простые множители целое число, состоящее более чем из 200 цифр, за считанные минуты. Такой прорыв мог бы сотрясти основы современного асимметричного шифрования, которое используется при генерации ключей шифрования для всех наших веб-коммуникаций.

---

#### ПРИМЕЧАНИЕ

Симметричное шифрование очень устойчиво к квантовым вычислениям и, следовательно, к алгоритму Шора.

---

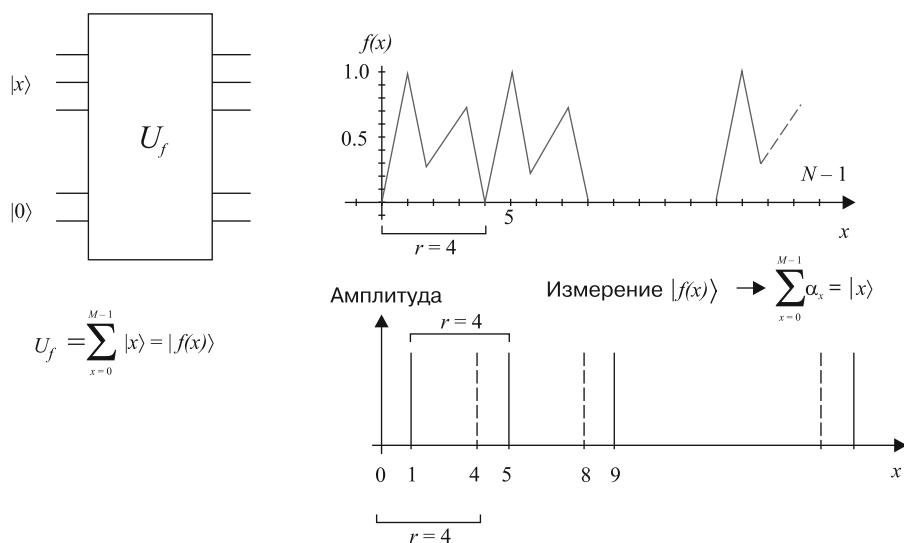
Но пока рано паниковать: до практической реализации на реальном квантовом компьютере еще далеко. Тем не менее данный алгоритм можно смоделировать в классической системе с помощью ProjectQ — прекрасной библиотеки Python. Мы познакомимся с ее реализацией позже, а в следующем разделе посмотрим, как можно эффективно решить задачу факторизации, определив период.

## Нахождение периода

Нахождение периода является базовым блоком алгоритма Шора. При помощи модульной арифметики данная задача может быть сведена к поиску периода ( $r$ ) функции  $f(x) = a^x \bmod N$  (рис. 8.9).

На рис. 8.9 приведен пример периодической функции  $f(x)$  с периодом  $r = 4$ . Чтобы алгоритм был осуществим,  $f(x)$  должна отвечать трем условиям.

1.  $f(x)$  однозначная для каждого периода, то есть значения  $f(x)$  не должны повторяться. На рис. 8.9 данные значения представлены вершинами всех линий за период.


 Рис. 8.9. Периодическая функция  $f(x)$ 

- Для любого количества периодов  $M$  период  $r$  должен быть делителем. Например, при  $M = 100$  и периоде  $r = 4$  отношение  $M / r = 25$ .
- Результат деления  $M$  на  $r$  должен быть больше, чем  $r$ , то есть  $M > r^2$ .

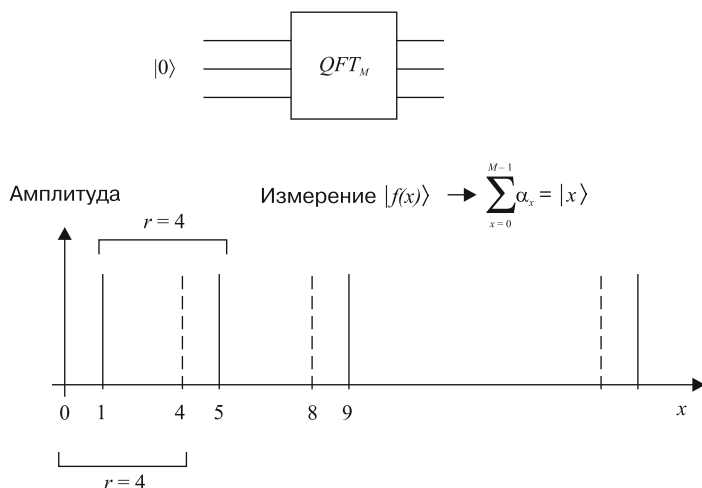
Алгоритм Шора преобразует  $f(x)$  в квантовую схему  $U_f$ , где входные кубиты находятся в суперпозиции. Измерив второй регистр<sup>1</sup> в  $U_f$ , мы можем увидеть значения амплитуд  $\sum_{x=0}^{M-1} \alpha_x |x\rangle$ , показанные на графике амплитуд на рис. 8.9. Здесь амплитуды отстоят друг от друга на четыре деления, что является искомым периодом. В данном частном случае мы получаем периодические суперпозиции с  $r = 4$ . Но что нам делать с ними? Алгоритм Шора использует еще один прием — квантовое преобразование Фурье.

<sup>1</sup> Судя по всему, автор использует такие два регистра: набор входных кубитов и набор выходных кубитов (операции могут производиться над обоими). Здесь речь должна идти о том, что при применении  $f(x)$  на состояние суперпозиции получается запутанное состояние типа  $|x_1, f(x_1)\rangle + |x_2, f(x_2)\rangle$  (как на рисунке), и, если произвести измерение над вторым регистром (то есть выходными кубитами), то получим амплитуды, о которых пишет автор. — *Примеч. науч. ред.*

## Преобразование Фурье

Преобразование Фурье — это процесс изменения данных, который допускает сдвиги во входных данных без изменения распределения выходных данных.

Это хорошо, потому что теперь у нас есть периодическая суперпозиция, где ненулевые амплитуды кратны периоду (рис. 8.10).



**Рис. 8.10.** Преобразование Фурье, показывающее периодическую суперпозицию

Но каким будет выходное значение преобразования Фурье? И чем оно нам поможет? Ответ: его выходным значением является случайное число, кратное  $M / r$ . В данном случае при  $M = 100$  и  $r = 4$  получаем случайное число, кратное  $100 / 4 = 25$ , что соответствует нашей цели. Посмотрим, как это происходит.

## Передача результатов преобразования Фурье в алгоритм Евклида для нахождения наибольшего общего делителя

После многократного выполнения преобразования Фурье мы получим множество случайных  $M / r$ . Например, можем получить 50, 75, 25 и т. д.

Теперь, если мы применим к случайным выходным результатам алгоритм Евклида для нахождения наибольшего общего делителя (НОД), то, разделив  $M$  на НОД, найдем период  $r$ . Таким образом:

$$r = M / \text{НОД}(50, 75\dots) = 100 / 25 = 4.$$

Это краткое описание алгоритма нахождения периода с использованием квантовой схемы. Чтобы понять, каким образом с помощью данного метода можно эффективно найти множитель, приведу пример факторизации числа  $N = 21$ . Решение задачи основано на двух весьма эффективных операциях:

- модульной арифметике:  $a = b \pmod{N}$ . Например,  $3 = 15 \pmod{12}$ ;
- НОД( $a, b$ ). Например,  $\text{НОД}(15, 21) = 3$ .

Тогда при  $N = 21$  нужно решить уравнение  $x^2 \equiv 1 \pmod{21}$ . А именно найти такой нетривиальный квадратный корень  $x$ , что:

- $N$  является делителем  $(x + 1)(x - 1)$ ;
- $N$  не является делителем  $(x \pm 1)$ .

Наконец, восстановить простой множитель, применив  $\text{НОД}(N, x + 1)$ .

Чтобы найти нетривиальный множитель для  $N = 21$ , выберем случайным образом  $x$ . Например, при  $N = 21$  выбираем  $x = 2$ , тогда:

$$\begin{aligned} 2^0 &\equiv 1 \pmod{21} \\ 2^1 &\equiv 2 \pmod{21} \\ 2^2 &\equiv 4 \pmod{21} \\ 2^3 &\equiv 8 \pmod{21} \\ 2^4 &\equiv 16 \pmod{21} \\ 2^5 &\equiv 11 \pmod{21} \\ 2^6 &\equiv 1 \pmod{21}. \end{aligned} \text{ Получен период } r = 6.$$

В данном случае  $2^6 = (2^3)^2$ . Значит,  $2^3 = 8$  — нетривиальный множитель, при котором 21 является делителем  $(8 + 1)(8 - 1)$ . Наконец, мы восстанавливаем значение множителя с помощью  $\text{НОД}(N, x + 1) = \text{НОД}(21, 9) = 3$ . Вообще говоря, нужно выбрать  $x$  случайным образом и затем пройти в цикле по  $x^0, x^1 \dots x^r \equiv \text{mod } N$ . Если нам повезло и  $r$  четный, то  $(x^{r/2})^2 \equiv 1 \pmod{N}$ . И тогда существует нетривиальный квадратный корень  $1 \pmod{N}$ .

**ПРИМЕЧАНИЕ**

Было доказано, что вероятность того, что нам повезет и  $g$  окажется четным, а  $x^2 \equiv 1 \pmod{N}$ , равна  $1/2$ . Однако, учитывая высокую вероятность успеха, это незначительно в общем масштабе.

Теперь запустим алгоритм, воспользовавшись прекрасной библиотекой Python под названием ProjectQ.

**Алгоритм Шора с использованием ProjectQ**

ProjectQ — это платформа с открытым исходным кодом для квантовых вычислений, которая реализует алгоритм Шора при помощи схемы, предложенной Стефаном Борегаром<sup>1</sup>. В ней применяются  $2n + 3$  кубитов, где  $n$  — количество битов факторизируемого числа  $N$ . Метод Борегара подразделяется на следующие шаги.

1. Если  $N$  четное, возвращается множитель 2.
2. Классическим способом определяется, выполняется ли  $N = p^q$  при  $p \geq 1$  и  $q \geq 2$ , и если да, то возвращается множитель  $p$  (на классическом компьютере это можно выполнить за полиномиальное время).
3. Выбирается такое случайное число  $a$ , что  $1 < a \leq N - 1$ . При помощи алгоритма Евклида для нахождения наибольшего общего делителя определяем, верно ли, что  $\text{НОД}(a, N) > 1$ . Если да, то возвращается множитель  $\text{НОД}(a, N)$ .
4. Для нахождения порядка  $r$  для  $a$  по модулю  $N$  используется квантовая схема. На квантовом компьютере данный шаг выполняется за полиномиальное время.
5. Если  $r$  нечетное или  $r$  четное, но  $a^{r/2} \equiv -1 \pmod{N}$ , то происходит переход к шагу 3. В противном случае вычисляется  $\text{НОД}(a^{r/2} - 1, N)$  или  $\text{НОД}(a^{r/2} + 1, N)$ . Выполняется проверка, является ли одно из этих значений нетривиальным множителем  $N$ , и если да, то возвращается данный множитель (на классическом компьютере это может быть выполнено за полиномиальное время).

<sup>1</sup> *Beauregard S.* Circuit for Shor's algorithm using  $2n+3$  qubits. Département de Physique et, Université de Montréal.



Борегар находит период, выполнив несколько управляемых суммирований и умножений в пространстве Фурье, чтобы найти решение  $f(x) = ax \pmod N \rightarrow a^x \equiv 1 \pmod N$  (рис. 8.11).

○ Управляемый умножитель  $U_a$  производит отображение  $|x\rangle \rightarrow |a^x \pmod N\rangle$ , где:

- $a$  — это классическое взаимно простое с модулем число, которое используется как основание для  $ax \pmod N$ ;
- $x$  — квантовый регистр;
- $c$  — регистр таких управляющих кубитов, что  $U_a = ax \pmod N$ , если  $c = 1$ , и  $x$  в ином случае.

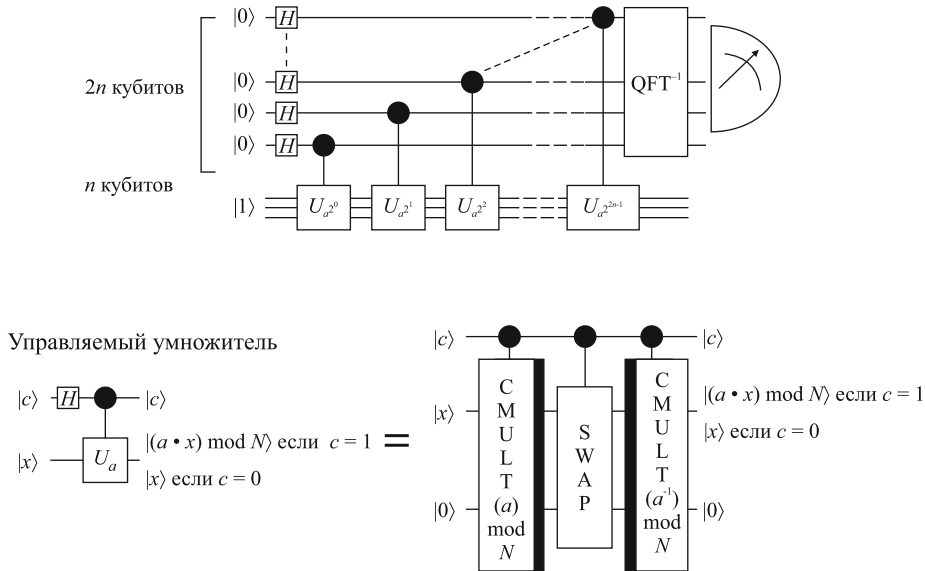


Рис. 8.11. Квантовая схема Борегара для нахождения периода

○ Управляемый умножитель  $U_a$ , в свою очередь, реализован как ряд модульных сумматоров с двойным управлением, при которых:

- если оба управляющих кубита  $c_1 = c_2 = 1$ , то на выходе  $f(x) = |a + b \pmod N\rangle$ . То есть это  $a + b \pmod N$  в пространстве Фурье. Обратите внимание, что этот вентиль складывает два числа: взаимно простое с модулем  $a$  и квантовое  $b$ ;

- если оба управляющих кубита,  $c1$  и  $c2$ , находятся в состоянии  $|0\rangle$ , то  $f(x) = |\varphi(b)\rangle$ .
- Модульный сумматор с двойным управлением построен на основе квантовой суммирующей схемы по Драперу. В данной схеме реализовано сложение классического значения ( $a$ ) с квантовым значением ( $b$ ) в пространстве Фурье.

## Факторизация с помощью ProjectQ

Установим ProjectQ и протестируем алгоритм. В первую очередь воспользуемся менеджером пакетов Python для загрузки и установки ProjectQ (для простоты я использую Windows; пользователи Linux должны следовать той же процедуре):

```
C:\> pip install projectq
```

Затем возьмите скрипт `shor.py` из папки с примерами для ProjectQ или исходников книги (`Workspace\Ch08\p08.shor.py`). Запустите его и введите факторизируемое число (листинг 8.2).

### Листинг 8.2. Алгоритм Шора с ProjectQ в действии

```
C:\>python shor.py
Number to factor: 21

Factoring N = 21: .....

Factors found : 7 * 3 = 21
Gate class counts:
  AllocateQubitGate : 166
  CCR : 1467
  CR : 7180
  CSwapGate : 50
  CXGate : 200
  DeallocateQubitGate : 166
  HGate : 2600
  MeasureGate : 11
  R : 608
  XGate : 206

Gate counts:
  Allocate : 166
```

```

CCR(0.098174770425) : 18
CCR(0.196349540849) : 30
CCR(0.392699081699) : 70
CCR(0.490873852124) : 18
CCR(0.785398163397) : 80
CCR(0.981747704246) : 38
CCR(1.079922474671) : 20
CCR(1.178097245096) : 16
...
R(5.252350217719) : 1
R(5.301437602932) : 1
R(5.497787143782) : 1
X : 206

```

```

Max. width (number of qubits) : 13.
--- 5.834410190582275 seconds ---

```

При  $N = 21$  скрипт выдает набор весьма полезных статистических данных, таких как:

- *количество задействованных кубитов.* При  $N = 21$  нужно пять кубитов, тогда общее количество кубитов  $2 \cdot 5 + 3 = 13$ ;
- *общее количество использованных вентилей по типам.* В данном случае с двойным управлением  $CCR = 1467$ ,  $CR = 7180$ ,  $C\text{Swap} = 50$ ,  $CX = 200$ ,  $R = 608$ ,  $X = 206$  и т. д. до общего количества 12 646 квантовых вентилей.

В ProjectQ квантовая схема для нахождения периода с помощью алгоритма Борегара приведена в листинге 8.3.

- Функция `run_shor` принимает три аргумента:
  - квантовый движок или моделирующее устройство, предоставляемое ProjectQ;
  - $N$  — факторизируемое число;
  - $a$  — взаимно простое с модулем число, которое используется в качестве основания для  $a^x \bmod N$ .
- Затем функция проходит в цикле от  $a = 0$  до  $a = \ln(N)$  с входным квантовым регистром  $x$  в суперпозиции и выполняется квантовая схема для  $f(a) = a^x \bmod N$  (рис. 8.11).

- Далее выполняется преобразование Фурье на регистре  $x$ , зависящем от предыдущих выходных значений, и производятся измерения.
- Наконец, измеренные значения суммируются, суммой является число в пределах  $[0, 1]$ . Затем выполняется разложение в цепную дробь, чтобы вернуть делитель или возможное значение периода ( $r$ ).

### Листинг 8.3. Квантовая подпрограмма для нахождения периода в ProjectQ

```
def run_shor(eng, N, a):
    n = int(math.ceil(math.log(N, 2)))

    x = eng.allocate_ureg(n)

    X | x[0]

    measurements = [0] * (2 * n) # будет хранить 2n результатов измерений

    ctrl_qubit = eng.allocate_qubit()

    for k in range(2 * n):
        current_a = pow(a, 1 << (2 * n - 1 - k), N)

        # Одна итерация QPE с одним кубитом
        H | ctrl_qubit

        with Control(eng, ctrl_qubit):
            MultiplyByConstantModN(current_a, N) | x

        # Выполнить инверсию QFT --> повороты в зависимости
        # от предыдущих выходных значений
        for i in range(k):
            if measurements[i]:
                R(-math.pi/(1 << (k - i))) | ctrl_qubit

        H | ctrl_qubit
        # и измерить
        Measure | ctrl_qubit
        eng.flush()
        measurements[k] = int(ctrl_qubit)
        if measurements[k]:
            X | ctrl_qubit

    Measure | x
    # Перевод измеренных значений в числа, лежащие в пределах [0, 1)
    y = sum([(measurements[2 * n - 1 - i]*1. / (1 << (i + 1)))
            for i in range(2 * n)])
```

```
# Разложение на множители для получения делителя (или периода?)
r = Fraction(y).limit_denominator(N-1).denominator

# Возврат (возможного) значения периода
return r
```

В следующем пункте приведен набор результатов факторизации для различных значений  $N$ .

## Результаты моделирования

Квантовая подпрограмма для нахождения периода в ProjectQ создается моделированием квантовой схемы на классическом компьютере, из-за чего ее применение для факторизации больших чисел нецелесообразно. Фактически она не способна разложить число, состоящее более чем из четырех цифр, на домашнем компьютере за приемлемое время. В табл. 8.3 приведен набор результатов для различных значений  $N$  — от 15 до 2491, полученных на моем ноутбуке.

**Таблица 8.3.** Результаты факторизации различных значений  $N$

Число $N$	Кубиты	Время, с	Память, Мбайт	Количество квантовых вентилей
15	11	2,44	50	CCR = 792; CR = 3186; CSwap = 32; CX = 128; H = 1408; R = 320; X = 130; Measure = 9
105	17	27,74	200	CCR = 3735; CR = 25 062; CSwap = 98; CX = 392; H = 6666; R = 1568; X = 393; Measure = 15

Продолжение ⇨

Таблица 8.3 (продолжение)

Число $N$	Кубиты	Время, с	Память, Мбайт	Количество квантовых вентилей
1150	25	17 542,12 (4,8 ч)	500	CCR = 15 366; CR = 139 382; CSwap = 242; CX = 968; H = 24 222; R = 5829; X = 981; Measure = 23
2491	27	246 164,74 (68,3 ч)	2048	CCR = 20 601; CR = 194 670; CSwap = 288; CX = 1152; H = 31 126; R = 7509; X = 1166; Measure = 25

На факторизацию четырехзначного числа 2491 ушло более 68 часов на 64-разрядном ПК с Windows 7, с процессором Intel Core i-5, с тактовой частотой 2,6 ГГц и с 16 Гбайт оперативной памяти. Я попытался разложить немного большее  $N = 8122$ , но через неделю сдался. В целом эти результаты показывают, что алгоритм можно успешно смоделировать для небольших  $N$ , однако его необходимо реализовать на реальном квантовом компьютере, чтобы проверить его действительную мощь.

В этой главе мы завершили свои исследования двумя алгоритмами, которые вызвали волнения по поводу возможностей практических квантовых вычислений. Это алгоритм Гровера — неструктурированный метод квантового поиска, способный находить входные данные в среднем за  $\sqrt{N}$  шагов, что намного быстрее, чем лучшее классическое решение со средним значением  $N/2$  шагов. Предполагаю, что в будущем все поиски в Интернете будут выполняться при помощи алгоритма Гровера.

Алгоритм Шора для факторизации на квантовом компьютере, по словам экспертов, может заменить современное асимметричное шифрование. Возможно, это самый известный квантовый алгоритм, он является яр-

ким примером мощных квантовых вычислений, обеспечивая экспоненциальное ускорение по сравнению с лучшим классическим решением.

Наконец, я хотел бы подвести итог, сказав, что из всех сил пытался объяснить сложные концепции квантовых вычислений, смешивая математику, программное обеспечение и столько числовых данных, сколько смог собрать. Множество чашек кофе было выпито, много бессонных ночей было потрачено на написание этой книги. Я надеюсь, что вам понравилось читать эту книгу так же, как мне — писать ее. И помните слова великого физика Ричарда Фейнмана: «Если кто-то говорит вам, что он понимает квантовую механику, это значит, что он не понимает квантовую механику».

*Владимир Силва*

## **Разработка с использованием квантовых компьютеров**

Перевел с английского *К. Сеница*

Заведующая редакцией	<i>Ю. Сергиенко</i>
Руководитель проекта	<i>С. Давид</i>
Ведущий редактор	<i>Н. Гринчик</i>
Научный редактор	<i>М. Коробко</i>
Литературный редактор	<i>Н. Рощина</i>
Художественный редактор	<i>Н. Васильева</i>
Корректоры	<i>Е. Павлович, Т. Радецкая</i>
Верстка	<i>Г. Блинов</i>

Изготовлено в России. Изготовитель: ООО «Прогресс книга».  
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,  
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 01.2019. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —  
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214,  
тел./факс: 208 80 01.

Подписано в печать 18.12.19. Формат 70х100/16. Бумага офсетная. Усл. п. л. 28,380. Тираж 700. Заказ 0000.

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».

142300, Московская область, г. Чехов, ул. Полиграфистов, 1.

Сайт: [www.chpk.ru](http://www.chpk.ru). E-mail: [marketing@chpk.ru](mailto:marketing@chpk.ru)

Факс: 8(496) 726-54-10, телефон: (495) 988-63-87