ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 178 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Андреев С.С., Дбар С.А.,
Климов Ю.А., Лацис А.О.,
Плоткина Е.А.

Квантовая модель вычислений глазами классического программиста

Рекомендуемая форма библиографической ссылки: Квантовая модель вычислений глазами классического программиста / С.С.Андреев [и др.] // Препринты ИПМ им. М.В.Келдыша. 2018. № 178. 30 с. doi:10.20948/prepr-2018-178
URL: http://library.keldysh.ru/preprint.asp?id=2018-178

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

С.С.Андреев, С.А.Дбар, Ю.А.Климов, А.О.Лацис, Е.А.Плоткина

Квантовая модель вычислений глазами классического программиста

Москва — 2018

Андреев С.С., Дбар С.А., Климов Ю.А., Лацис А.О., Плоткина Е.А.
Квантовая модель вычислений глазами классического программиста.

Работы по созданию действующих прототипов квантовых вычислителей многокубитного размера в последние годы продвинулись далеко вперед. Количество возможных применений квантового компьютера, модельных реализаций квантовых алгоритмов также растет ускоряющимся темпом. Необходимо включать в эти работы гораздо большее число специалистов по высокопроизводительным вычислениям, чем это имеет место сегодня. Этому, однако, препятствует отсутствие традиции краткого и внятного объяснения сути квантовой модели вычислений на языке, понятном классическому программисту. В нашей работе мы попытались внести свой вклад в решение этой проблемы.

Ключевые слова: квантовый компьютер, квантовый алгоритм, модель вычислений, высокопроизводительные вычисления.

Sergey Sergeevich Andreev, Svetlana Alekseevna Dbar, Yuriy Andreevich Klimov, Aleksey Ottovich Lacis, Elena Aronovna Plotkina
Quantum computation model: the classical programmer's viewpoint.

Building the working prototypes of many-qubit quantum computers advance rapidly during the last few years. Number of possible implementations of quantum computers, together with the number of sample quantum algorithm implementations, grow rapidly as well. It seems necessary to involve much more HPC experts in those works than it is done today. A big problem in doing this is an absolutely unacceptable way in which quantum computation model is traditionally explained today to classical programmers. In our work we tried to contribute to the solution of this problem.

Key words: quantum computer, quantum algorithm, computation model, HPC.

Работа выполнена при поддержке Программы Президиума РАН №26 «Фундаментальные основы создания алгоритмов и программного обеспечения для перспективных сверхвысокопроизводительных вычислений»

Введение

Идея квантового компьютера интенсивно обсуждается уже более 20 лет. До недавнего времени к перспективе ее практической реализации в обозримом будущем мало кто относился всерьез. В [1], например, можно встретить следующий тезис: «Лично у меня есть большие сомнения, что когда-либо будет создан работающий и решающий "важные народнохозяйственные задачи" квантовый компьютер. Однако в любом случае работа над созданием квантовых компьютеров сыграет важную роль...». Автор сетует не только на исключительную сложность физической реализации квантового компьютера, но и на отсутствие сколько-нибудь заметного количества полезных алгоритмов для него. В этом с ним полностью солидарен автор работы [2]. Отметим, что работа [1] была написана 10, а работа [2] — 6 лет назад, то есть в 2012 году. Однако проходит всего 3 года, и в 2015 году, то есть 3 года назад, в работе [3] обнаруживаем главу "Квантовый зоопарк", из которой вполне недвусмысленно следует, что число квантовых алгоритмов измеряется уже десятками. Что же касается физической реализации, то число кубитов в изготавливаемых сегодня прототипах универсального квантового компьютера год назад перешагнуло, по некоторым данным, за 50 [4], хотя совсем недавно и 20 кубитов трудно было вообразить.

Словом, вероятность того, что сказка о квантовом компьютере станет былью не просто в обозримом будущем, а довольно скоро, растет устрашающими темпами. Это вынуждает сообщество специалистов по классическим высокопроизводительным вычислениям если не участвовать в этих работах непосредственно, то, по крайней мере, быть в курсе и иметь обоснованную позицию. Игнорировать проблему "по умолчанию", в силу, якобы, отсутствия самой темы для обсуждения, более нельзя.

Чтобы сформулировать позицию по некоторому вопросу, этот вопрос следует сначала изучить. С этим у сообщества "классических суперкомпьютерщиков" серьезнейшие проблемы, как, впрочем, и вообще у подавляющего большинства классических программистов. Казалось бы, четверть века существования квантовой информатики — достаточный срок для появления учебников, объясняющих основы этой научной дисциплины так же доходчиво, как в соответствующих учебниках разъясняются основы классического программирования. Однако никакой традиции внятного разъяснения основ квантовой информатики (в письменном виде) не появилось до сих пор. Это поражает, и с выяснения причины такого грандиозного "методического провала" имеет смысл начать разбор самого предмета.

1. Почему классические программисты не понимают квантовой модели вычислений

1.1. Кто такие классические программисты

Во всех без исключения работах, посвященных квантовой информатике, последняя противопоставляется информатике классической. Понятно, что классический программист — это кто-то, твердо владеющий "основами", но, возможно, не очень хорошо знающий "новшества".

Но что есть "новшества"? К твердому программистскому фундаменту "на века" без колебаний можно отнести, пожалуй, язык Фортран-4. По мере развития программирования "новшествами" были Паскаль, язык Си, затем — Си++, Фортран77 и Фортран-95, Java, Python, … — список получается длинным. Где граница "классики"?

Сведущий в моделях и технологиях программирования читатель, скорее всего, предложит считать "классикой" императивные, или, точнее, фоннеймановские, языки программирования. Но тогда, быть может, квантовое программирование легче освоить специалистам по нетрадиционным моделям параллельного программирования, вроде OpenCL или языков прямой схемной реализации вычислений в ПЛИС? Или, быть может, все наоборот: со дня на день появится компилятор Фортрана-4 на квантовый компьютер, и все мучения с пресловутым "распараллеливанием на графический процессор" останутся, наконец, в прошлом?

В действительности, к сожалению, все гораздо хуже. Абсолютно **все** применяемые сегодня на практике технологии программирования, от таких древних и фундаментальных, как Фортран-4, до таких изощренных и временами трудно изучаемых, как функциональные языки, или языки программирования нетрадиционных суперкомпьютерных архитектур, **относятся, без сомнения, к области классической информатики**. Все эти технологии имеют в своей основе следующие тезисы:

- детерминированная арифметическая и логическая обработка чисел, где под числом понимается двоичное целое конечной разрядности, возможно, интерпретируемое как вещественное с плавающей точкой;
- хранение обрабатываемых данных в некоторой адресуемой памяти.

Все различия классических, как мы теперь знаем, архитектур между собой касаются исключительно способа записи процедур обработки числовых данных, допускающего или подразумевающего те или иные формы параллельного доступа к данным и выполнения вычислений. Сама обработка, сам способ получения одних чисел из других, вообще говоря, не зависит от того, выполняется ли императивная программа или функциональная, работает ли традиционный процессор, GPGPU или спецвычислитель на ПЛИС. Если мы по окончании расчета мысленно "прокрутим назад" вычислительную

процедуру, мы увидим, что выходные данные получены из входных, грубо говоря, по одним и тем же формулам, какой бы вид классического компьютера ни использовался.

Природа квантового компьютера, напротив, принципиально другая. В озвученные выше два тезиса он категорически не укладывается, он принципиально иначе обрабатывает принципиально другие данные, хранимые и представляемые принципиально другим способом. Он ни в коей мере не является "вариацией на тему" CUDA или OpenCL, "распараллелить на квантовый компьютер" произвольный алгоритм в принципе невозможно. Квантовый компьютер не продолжает ряд нетрадиционных (но все же классических!) архитектур, а стоит далеко вне этого ряда. "Фортрана-4 для квантового компьютера", скорее всего, не будет никогда.

Итак, мы пояснили понятие "классический программист" и заодно выяснили, чем квантовый компьютер **не является**. Вопрос о том, чем же он тогда **является**, обсудим совсем скоро. Сначала попытаемся понять, почему все попытки объяснить это даются, вплоть до сегодняшнего дня, с таким неимоверным трудом.

1.2. Что именно не так с объяснением квантовой модели

Причина всякого "массового непонимания" кроется в ошибках объяснения. Если непонимание длится четверть века, ошибка должна быть привычной, стать частью традиции. Это делает ее (ошибку) менее заметной, а ее повторение — упорным как никогда. Чтобы понять, в чем состоит вошедшая за долгие годы в традицию, воспроизводимая многими авторами снова и снова ошибка в разъяснении модели квантовых вычислений, воспользуемся простой аналогией.

Представим себе выдающегося ученого, скажем, конца 18-го века, неким волшебным образом узнавшего и понявшего, что такое классический компьютер и каковы его вычислительные приложения. Как он стал бы объяснять это другим? Наверное, примерно так:

- компьютер строится из диодов, емкостей и транзисторов. Начнем с законов Кирхгофа;
- затронем комбинационную схемотехнику, затем, отдельной главой — теорию конечных автоматов;
- без теории грамматик и языков, методов разбора и построения атрибутированных семантиками синтаксических деревьев тоже никуда не деться;
- затем, естественно, структурное программирование и структуры данных;
- теперь настала очередь численных методов, теории разностных схем...

Выстроенный нами ряд не окончен, хотя и близок к концу. Он, действительно, повествует о взаимосвязанных и взаимозависимых вещах, без понимания, хотя бы поверхностного, каждой из которых представление о классическом компьютере не может быть полным. Любой действительно

образованный специалист по вычислительному делу все это знает, хотя, конечно, разные темы из упомянутых — в разной степени. И знает он, хотя бы понемногу, **каждую** из этих тем благодаря тому, что в свое время имел возможность изучить их **по отдельности, в произвольном порядке и независимо**. При изучении программирования о законах Кирхгофа временно забывают, точно так же, как при изучении свойств разностных схем забывают об особенностях реализации лексического анализа в компиляторе языка Паскаль. Иными словами, классическая информатика за время своего существования разделилась на слои, или уровни абстракции, позволяющие независимо рассматривать изучаемый предмет — вычислительную систему — под разными углами зрения. Как совершенно справедливо отмечается в [2], человек мыслит последовательно, но не потому, что плохо владеет технологиями MPI и OpenMP, а потому, что склонен дробить стоящие перед ним задачи на части и справляться с этими частями по очереди. И к задаче изучения чего-либо это относится как к никакой другой.

Продолжим наш мысленный эксперимент. Вымышленный нами ученый конца 18-го века чудесным образом знает все перечисленное выше, но истории развития классической информатики он не изучал. Очевидно, при попытке передать свои знания он попробует объяснить все целиком, без какого-либо деления на слои. От слушателей будет требоваться равномерно напрягать мозги одновременно по всем перечисленным выше темам — и никак иначе!

Результат, без сомнения, будет ужасен. Возможно, люди, способные воспринять классическую информатику сразу, целиком, в едином, нерасчлененном виде, существуют, но их очень мало, и сил они затратят уйму. У остальных же учеников нашего вымышленного лектора шансов просто нет.

К величайшему сожалению, сегодня, в 21-м веке, квантовая информатика объясняется, по большей части, именно тем способом, который мы мысленно сконструировали только что, то есть нарочито целостно, без попытки выделения хотя бы условно независимых смысловых слоев (в лучшем случае, в качестве отдельно рассматриваемого слоя изолируются вопросы физической реализации квантового компьютера). Не всегда удачные метафоры, наряду с желанием авторов подробно рассказать еще и о своем любимом способе программной реализации эмулятора квантовых вычислений, довершают дело. Не зная уже заранее, о чем идет речь, понять невозможно ничего. При этом нельзя сказать, что авторы работ по квантовой информатике не видят проблемы или игнорируют ее. В [3], например, с самого начала признается наличие порочного круга в традиционных способах введения читателя в курс дела, как и дурной традиции объяснять все недостаточно просто. И даже есть глава под названием "Наивное понимание модели квантовых вычислений". Не делается (или делается слишком робко) только завершающий шаг: не проводится в должной мере так необходимое разделение на условно независимые смысловые слои. Упрощение изложения понимается как упрощение все той же комплексной, нерасчлененной картины, а не как результат ее (картины)

надлежащего расчленения. Справедливости ради следует отметить, что как раз работа [3] в этом плане относится скорее к лучшим, чем к худшим.

Таким образом, если мы хотим постараться что-то не просто осознать "звериным чутьем", а понять так, чтобы можно было однажды понятое объяснить другим, наша первоочередная задача — разбить модель квантовой вычислительной системы на слои, подобные упомянутым выше слоям в классической информатике. Не претендуя на полное решение этой важной задачи, попробуем выделить и описать один, но очень важный, слой, соответствующий в классической информатике, приблизительно, рассмотрению процессора на уровне чуть выше уровня системы команд. Именно его обычно называют (забывая зачастую надлежащим образом отделить от других) "квантовой моделью вычислений".

2. Квантовая модель вычислений, только она и ничего, кроме нее

2.1. Место квантовой модели вычислений в будущей иерархии смысловых слоев

Называя квантовую модель вычислений условно независимым смысловым слоем в единой модели квантовой информатики, мы предполагаем, что какие-то слои лежат в иерархии уровней абстракции ниже нее, а какие-то — выше. Перед тем как окончательно углубиться в квантовую модель вычислений, обрисуем, очень кратко и приблизительно, ее место в иерархии уровней абстракции.

Ниже уровня квантовой модели вычислений лежит, без сомнения, все, что связано с **физической реализацией квантового компьютера**. Таковых на сегодня известно несколько, различные прототипы строятся принципиально разными способами. Все это чрезвычайно, прямо-таки критически, важно для физиков, занятых попытками реализации квантового компьютера "в железе", но модели вычислений не затрагивает. Точно так же, как в основах классического программирования мы не найдем ответа на вопрос о том, является ли машина, о которой идет речь, электронной, электромеханической или же чисто механической, как у Чарльза Беббиджа. Также ниже уровня модели вычислений лежит **квантовая механика**, то есть та математическая теория, которая хорошо описывает явления микромира и "побочным продуктом" которой является квантовая модель вычислений. Мы будем рассматривать эту модель саму по себе, не задумываясь о том, откуда она взялась.

Выше уровня квантовой модели вычислений лежит все, что связано с созданием и обоснованием **квантовых алгоритмов**. Для квантовой модели вычислений это не важно, подобно тому, как для проектирования, тестирования, наладки и ремонта классических ламповых процессоров совершенно не требовалось понимать, каким образом именно эти

последовательности процессорных команд находят приближение к решению того или иного уравнения.

Все случаи, когда нам потребуется вторгаться в соседние слои нашей иерархии, будем далее отмечать отдельно.

2.2. Какие данные обрабатывает квантовый компьютер

Практически все литературные источники, от самых популярных до самых серьезных, начинают рассказ о квантовых вычислениях с одной и той же метафоры. Речь идет о метафоре значения кубита — элемента обрабатываемых квантовым компьютером данных. Согласно этой метафоре, кубит, в отличие от классического бита, способен принимать не одно из значений — 0 или 1 — а "как бы сразу оба" этих значения. Метафора кубита, принимающего "сразу все" возможные значения, стала настолько привычной, что мало кто задумывается об ее, в общем-то, неудачности, поскольку она мало что объясняет, но многое запутывает с самого начала.

Для объяснения того, что такое кубит и квантовый регистр, будем далее использовать более уместную "магазинную аналогию". В этой метафоре значение кубита описывает поведение человека, который пошел в магазин с 30-процентным, например, намерением купить некоторую вещь и, соответственно, с 70-процентным намерением посмотреть, подумать, но от покупки воздержаться. Учитывая, что человек, в отличие от кубита, существо одушевленное, мы можем, конечно, говорить о том, что мысленно он **находится сразу в обоих состояниях** — в состоянии "купил" и в состоянии "не купил", но правильнее было бы сказать, что он **пока не находится ни в одном из состояний**, а только размышляет, в какое из них ему бы следовало перейти. Именно такое положение дел скрывается за словами **квантовая суперпозиция.**

Пусть теперь наш вымышленный покупатель ходил по магазину так долго, что прозвенел звонок, сигнализирующий о закрытии магазина через 5 минут. Иными словами, покупатель испытал внешнее воздействие, в результате которого он вынужден реализовать одно из своих базовых состояний, то есть либо купить (с вероятностью 30%), либо не купить (с вероятностью 70%). Такое внешнее воздействие, вынуждающее кубит реализовать окончательно то или иное базовое состояние, называется **наблюдением**, или **измерением** кубита. Измерение необратимо, информация о том, каковы были вероятности реализации базовых состояний до измерения, теряются.

С практической точки зрения отдельный кубит большого интереса не представляет. Все вычисления в действительности происходят с квантовыми регистрами, то есть "ячейками памяти", которые складываются из кубитов точно так же, как классические регистры, или ячейки памяти, складываются из классических битов. Кубиты в пределах квантового регистра могут находиться в состоянии **квантовой запутанности**, что, собственно, и делает квантовый регистр интересным с вычислительной точки зрения.

Для начала проясним понятие квантовой запутанности при помощи все той же магазинной аналогии.

Пусть покупателей, пришедших в магазин без твердого намерения купить интересующий их товар, теперь не один, а двое. Для описания их состояния теперь необходим двухкубитный квантовый регистр, в котором намерения каждого покупателя представлены отдельным кубитом. Далее возможны две принципиально различных ситуации:

- либо покупатели не знакомы и не обращают друг на друга никакого внимания. В этом случае кубиты независимы, квантовая запутанность отсутствует;

- либо покупатели знакомы, наблюдают друг за другом, и решение одного способно оказать влияние (не важно, в какую сторону) на решение другого. Точнее, решения наших двух покупателей коррелируют. В этом случае имеет место квантовая запутанность кубитов.

Продолжим магазинную аналогию для случая квантовой запутанности. Пусть один из покупателей невольно вызвал подозрение у охранника, и тот потребовал от него немедленно покинуть магазин, с покупкой или без. Такое внешнее воздействие на квантовый регистр называется **частичным измерением** (один кубит измерен, второй — нет). После частичного измерения оставшийся покупатель будет знать, какой выбор сделал его товарищ, что повлияет на его (оставшегося покупателя) дальнейшее поведение, то есть на вероятности купить или не купить, когда ему самому придется покидать магазин.

Теперь попробуем выразить все то же самое на немного более строгом языке.

Квантовый регистр шириной **N** состоит из **N** упорядоченных кубитов. Он может находиться в одном из двух состояний: измеренном или неизмеренном. Значения, принимаемые квантовым регистром в этих двух состояниях, отличаются на качественном уровне (в терминах классического программирования – имеют разные типы).

Пусть **квантовый регистр полностью измерен**. При измерении, как мы знаем, каждый кубит становится обычным битом, а квантовый регистр, соответственно, обычным регистром из **N** битов. Значит, значением измеренного квантового регистра является **N**-разрядное двоичное число, "выпавшее" при измерении.

Пусть имеет место **квантовая суперпозиция** (квантовый регистр не измерен). Тогда его значением является информация о вероятностях "выпадения" при измерении каждого из $\mathbf{2^N}$ возможных **N**-разрядных двоичных чисел. Эту информацию всегда можно представить в виде вектора (одномерного массива) **P** длиной $\mathbf{2^N}$ вещественных чисел. Значение **k**-го элемента вектора **P[k]**

равно вероятности "выпадения", при измерении квантового регистра, числа **k**. Все элементы вектора, таким образом, неотрицательны, а в сумме дают единицу.

Если **кубиты независимы** (квантовой запутанности нет), то для каждого кубита определена индивидуальная вероятность выпадения единицы при измерении, а элементы упомянутого выше вектора вероятностей **P** можно выразить через эти индивидуальные вероятности, которых всего **N** штук. Получается, что информацию об итоговых вероятностях выпадения каждого из возможных значений квантового регистра можно представить в виде вектора из всего **N** элементов. Представление итоговых вероятностей в виде вектора длиной $\mathbf{2^N}$ также возможно, просто значения элементов этого вектора не будут произвольными, а будут обязательно выражаться через вероятности выпадения нулей и единиц в отдельных разрядах.

Если **кубиты зависимы** (возможна квантовая запутанность), то само понятие вероятности выпадения единицы в отдельных разрядах квантового регистра перестает иметь смысл. Представление значения неизмеренного квантового регистра в виде вектора **P** длиной $\mathbf{2^N}$ вещественных чисел становится единственным возможным. В отличие от случая независимых кубитов, значения элементов вектора **P** становятся произвольными – лишь бы они были неотрицательными и давали в сумме единицу. Именно этот произвол в задании значений элементов вектора и позволяет задать любую конкретную форму корреляции между кубитами в составе квантового регистра (квантовой запутанности).

Поскольку квантовая запутанность принципиальна для квантовых вычислений, значением неизмеренного квантового регистра является вектор длиной именно $\mathbf{2^N}$ (а не **N**) элементов.

Сразу сделаем несколько важных замечаний.

1) Значения квантового регистра не только хранятся и измеряются, но и обрабатываются (о том, как именно, расскажем ниже). Информацию, необходимую для измерения, естественно представлять в виде вектора длиной $\mathbf{2^N}$ вещественных чисел. Операции же обработки квантового регистра, в отличие от операции его измерения, имеют дело не только с информацией о вероятности выпадения того или иного значения при измерении, но еще и с некоторой дополнительной информацией. Эта дополнительная информация также должна быть представлена в значении квантового регистра. В этой работе мы сознательно не обсуждаем вопрос о том, что это за информация, каков именно ее физический (или хотя бы математический) смысл. Обсуждение этого вопроса неизбежно увело бы нас в квантовую механику, то есть отвлекло бы от главной цели данного раздела — от разъяснения квантовой модели вычислений в ее "программистском" приближении. Поэтому здесь мы просто скажем (без каких-либо пояснений), как именно представляется в значении квантового

регистра эта дополнительная информация. Значением квантового регистра в действительности является вектор из 2^N не вещественных, как говорилось выше, а комплексных чисел, причем вероятностью является квадрат модуля соответствующего комплексного числа. Сумма квадратов модулей всех элементов вектора равна единице. При измерении, частичном или полном, конкретное значение комплексного числа не важно, важно только значение квадрата его модуля.

2) Интуитивно очевидно, что квантовое вычислительное устройство тем лучше, чем больше в его составе кубитов. Но тогда возникает вопрос о распределении этих кубитов по квантовым регистрам. На этот вопрос имеется вполне однозначный ответ. Поскольку кубиты могут быть запутаны только в пределах одного квантового регистра, в действительно универсальном квантовом вычислителе все кубиты должны быть собраны в один квантовый регистр, то есть потенциально могли бы быть запутаны произвольным образом. На уровне конкретного квантового алгоритма может оказаться удобно оперировать несколькими регистрами, роль которых в этом случае легко выполнят отдельные поля в составе единого большого регистра. Отступление от этого правила является недостатком, ограничивающим возможности квантового вычислительного устройства.

2.3. В чем заключается обработка

Обработка значений квантового регистра заключается в применении к нему в определенном порядке **унитарных преобразований**, то есть умножения вектора значений регистра длиной 2^N на произвольную унитарную матрицу размером 2^N на 2^N, с получением нового вектора. Унитарные преобразования сохраняют свойство вектора давать единицу в сумме квадратов модулей всех его элементов. Они обладают еще целым рядом замечательных свойств — например, они обратимы. Нам же сейчас крайне важно воздержаться от "краткого" и, конечно же, "совсем простого, в отличие от пояснений других авторов", экскурса в основы ТФКП и продолжить рассказ в терминах именно модели вычислений.

Первое, что должно заинтересовать к этому моменту внимательного читателя, — это, конечно, вопрос о том, как устроена "квантовая программа".

В самом деле, в случае классического компьютера понятие модели вычислений никак не исчерпывается перечнем доступных арифметических и логических операций. Не менее важно понятие программы, выполняющей эти операции в заданной последовательности, а также то, что последовательность может меняться в зависимости от промежуточных результатов вычислений. Что, хотя бы примерно, соответствует этим понятиям в квантовом случае?

Программе классического компьютера приблизительно соответствует так называемая **квантовая схема**. Она представляет собой суперпозицию унитарных преобразований, как бы последовательно применяемых к

квантовому регистру в целом или к его частям (применение унитарного преобразования к части регистра тривиально сводится к применению преобразования, также унитарного, ко всему регистру). Подобно тому, как в классической схемотехнике схемы строятся из элементарных устройств преобразования данных, называемых вентилями, квантовая схема строится из некоторых базисных унитарных преобразований, называемых **квантовыми вентилями**. Все квантовые вентили, как и любые унитарные преобразования, обратимы, и потому каждый из них с необходимостью содержит одинаковое число входов и выходов. Как и в случае классической схемотехники, вентили, применяемые к двум кубитам, обычно легко обобщаются на случай **N** кубитов.

В квантовой схеме нет ничего напоминающего циклы или ветвления классического программирования (хотя элементы условной логики могут быть встроены внутрь некоторых унитарных преобразований). Таким образом, квантовая схема — это просто одна большая формула, как на входе, так и на выходе которой имеется вектор комплексных чисел длиной $\mathbf{2^N}$, сумма квадратов модулей элементов которого равна единице. Поскольку суперпозиция унитарных преобразований дает также унитарное преобразование, любую квантовую схему можно представить в виде всего одной унитарной матрицы, умножение вектора на которую и будет означать применение к нему этой схемы. Из того, что такая матрица существует, не следует, что с ней обязательно удобно работать именно в явном виде. В квантовых алгоритмах эту матрицу обычно не вычисляют, а оставляют представленной в виде квантовой схемы, то есть суперпозиции либо отдельных квантовых вентилей, либо чего-то вроде "квантовых типовых логических узлов". Вопрос о том, будет ли матрица унитарного преобразования, соответствующая всей схеме целиком, действительно сначала вычисляться, а затем применяться к квантовому регистру за один прием при работе реального квантового компьютера, мы здесь не обсуждаем, но допускаем такую возможность.

Как мы видим, квантовый компьютер представляет собой устройство с довольно простым управлением. Настолько простым, что в роли самодостаточной вычислительной системы его представить себе трудно. Действительно, квантовый компьютер принципиально мыслится как сопроцессор при компьютере классическом. Квантовый алгоритм (пишется без кавычек, поскольку это — именно алгоритм, в классическом значении этого слова) выполняется на классическом компьютере, используя квантовый вычислитель для реализации отдельных элементарных шагов. Действия, которые классический компьютер может выполнить со своим квантовым сопроцессором, следующие:

- заготовить необходимые квантовые схемы;
- задать квантовому регистру некоторое начальное значение;
- применить квантовую схему к квантовому регистру;
- измерить квантовый регистр, узнать результат.

Пока мы видим несколько обременительный в силу ряда "нелепых ограничений", но все же довольно обыкновенный векторный сопроцессор. Почему же квантовые алгоритмы так трудно изобретать? Ответ — в единственном, но очень важном замечании, которое мы намеренно откладывали вплоть до этого момента. Вот оно:

Содержимое вектора состояния квантового регистра недоступно напрямую. Все унитарные преобразования выполняются так, как если бы этот вектор действительно хранился в квантовом вычислителе, со всеми своими элементами, но достать их значения наружу нельзя. Единственное, что можно сделать для извлечения информации из квантового регистра, — это измерить его, частично или целиком.

Таким образом, все выполняемые с квантовым регистром унитарные преобразования направлены на то, чтобы как можно больше повысить вероятность выпадения правильного ответа, когда придет время измерения. Просто прочитать "как бы хранящиеся" в векторе состояний вероятности, чтобы с чем-нибудь сложить или на что-нибудь умножить эти числа — не получится. Начальные значения, как мы говорили выше, задать можно: это делается путем применения к квантовому регистру некоторых вспомогательных квантовых схем, а вот достать изменившиеся в результате вычислений значения обратно — нельзя.

Отсюда становится ясно, в чем примерно заключается так часто упоминаемый **квантовый параллелизм**. Квантовый компьютер применяет квантовую схему любого размера и сложности практически мгновенно. При его эмуляции на классическом компьютере это выливается в громадные векторные вычисления, которые квантовый компьютер, в отличие от классического, выполняет "как бы параллельно". В действительности он их не выполняет совсем, поскольку векторы состояния квантовых регистров в виде массивов комплексных чисел "на самом деле" не существуют. Точнее, они существуют в голове классического программиста, пытающегося понять, как работает квантовый компьютер, в качестве способа объяснения этой работы на языке классических вычислений. И, конечно, эти векторы существуют в памяти классического компьютера, когда он эмулирует квантовый.

Несколько особое место в квантовой модели вычислений занимает проблема **квантовой декогеренции**. Проблема эта состоит в том, что при работе квантового компьютера часто происходят сбои, выражающиеся в "порче" значения еще не измеренного квантового регистра. Порча происходит всегда в направлении "распутывания" кубитов. Строго говоря, это проблема нижележащего уровня в иерархии уровней абстракции. Квантовая модель вычислений должна была бы предполагать квантовый компьютер исправным, не склонным "разваливаться на ходу". Однако, поскольку на нижележащих уровнях, а именно — на уровне физической реализации квантового компьютера, уже не первый год наблюдаются серьезные проблемы, разработчикам квантовых алгоритмов (опять "не наш" уровень, на этот раз —

более высокий) приходится учитывать возможность декогеренции в своих алгоритмах. В программных эмуляторах квантовых компьютеров часто имеется специальный режим имитации декогеренции.

3. Виды деятельности в квантовой информатике

Построенная нами вчерне квантовая модель вычислений, точнее, рассуждения при ее построении, могут помочь нам с ответом на важный практический вопрос. А именно — на вопрос о том, какие виды профессиональной деятельности связаны с квантовой информатикой, какие специалисты необходимы в этой отрасли и почему. Довольно грубо и приблизительно список мог бы выглядеть так:

- **физическое изготовление квантового вычислителя**. Этим занимаются физики-экспериментаторы и специалисты по квантовой механике;
- **математическое моделирование (симуляция) его конкретных физических реализаций**. Этим занимаются математики-прикладники, специалисты по основам квантовой механики и по численным методам. Большое внимание при построении таких моделей уделяется проблеме декогеренции;
- **разработка прикладных квантовых алгоритмов**. Это — новая область деятельности, специалистов в которой еще только предстоит воспитать, скорее всего, из тех, кто занимается сегодня классической информатикой;
- **моделирование (эмуляция) квантовых вычислений на классических компьютерах**. Такие действующие модели необходимы разработчикам прикладных квантовых алгоритмов. Этим занимаются программисты, в частности — параллельные. Знать необходимо квантовую модель вычислений, и примеры квантовых алгоритмов.

На последнем в этом кратком перечне виде деятельности хотелось бы остановиться особо.

С одной стороны, по мере все более активной разработки прикладных квантовых алгоритмов, появляется и растет спрос на квантовый вычислительный эксперимент. Например, такие современные приложения, как (возможное) использование методов квантовой информатики в искусственном интеллекте и машинном обучении, без вычислительного эксперимента просто немыслимы.

С другой стороны, настоящие квантовые компьютеры пока дороги, слабоваты и ненадежны, а имеющиеся их классические программные эмуляторы требуют суперкомпьютерных мощностей. В этой связи интересной представляется идея реализации квантового эмулятора на нетрадиционных высокопроизводительных архитектурах, например, GPGPU или ПЛИС. После рассмотрения примера квантового алгоритма мы вернемся к этой теме.

4. Пример: реализация алгоритма Гровера средствами библиотеки libquantum

До этого момента наши рассуждения о квантовой модели вычислений имели довольно абстрактный характер. Цель этих рассуждений состояла в том, чтобы сообщить в явном виде те важные начальные сведения, о которых большинство авторов работ по квантовым вычислениям почему-то предпочитают умалчивать, создавая у читателя стойкое ощущение "вырванных страниц". Будем надеяться, что хотя бы часть из них нам удалось "вклеить обратно".

Теперь посмотрим, как все рассказанное к этому моменту выглядит на практике. Чтобы не начинать нового тура абстрактных рассуждений, будем параллельно рассматривать две вещи: библиотеку эмуляции квантовых вычислений libquantum [5] и реализованный ее средствами алгоритм Гровера. Тестовая реализация этого алгоритма входит в дистрибутив библиотеки. Мы эту реализацию рассмотрим и немного усовершенствуем. Хотя теперь мы от общих, абстрактных рассуждений переходим к анализу конкретных текстов программ, способных компилироваться и выполняться, ощущение "вырванных страниц" не оставит нас еще долго. И связано оно будет вовсе не со сложностью понимания устройства квантовых алгоритмов. Так, одному из авторов этого текста в свое время потребовалось несколько дней, чтобы понять, что делает (не **как** работает, а именно **что** делает) эта самая тестовая реализация алгоритма Гровера. Впрочем, к обсуждению алгоритма нам трудновато будет приступить, если мы не скажем нескольких слов о библиотеке, средствами которой он реализован. С этого и начнем.

4.1. Библиотека libquantum

Библиотека libquantum [5] — свободно распространяемая библиотека эмуляции квантовых вычислений, написанная на Си, предназначенная для использования, в первую очередь, в Linux. Поддерживает тип данных "квантовый регистр", операции применения квантовых вентилей к квантовому регистру, операции измерения. Эмулятор конкретного квантового алгоритма представляет собой программу на Си, вызывающую функции применения квантовых вентилей к квантовым регистрам, измерения квантовых регистров, и некоторые другие.

Как и во всех случаях, когда средством эмуляции гибридной вычислительной системы является библиотека, главная проблема состоит в разграничении "процессорной" и "сопроцессорной", то есть, в нашем случае, классической и квантовой, частей алгоритма. Например, рассмотрим такой фрагмент исходного текста:

```
for(i=0;i<reg->width;i++)
  {
    if(!(state & (1 << i))) quantum_sigma_x(i, reg);
```

```
 }
quantum_toffoli(0, 1, reg->width+1, reg);
……
```

Про библиотечные функции **quantum_sigma_x()** и **quantum_toffoli()** нам пока достаточно знать, что вызов каждой из них означает применение к квантовому регистру некоторого квантового вентиля, то есть очень простого, элементарного унитарного преобразования.

Теперь вспомним о том, что мы имеем дело с эмулятором, то есть с имитационной моделью квантовой вычислительной системы. Квантовая вычислительная система, как мы знаем, состоит из классического компьютера и квантового сопроцессора. Любые действия, записанные в программе эмуляции, при переносе на квантовую вычислительную систему, пусть даже пока — на гипотетическую, должны выполняться либо в классической ее части, либо в квантовом сопроцессоре. Глядя на исходный текст эмулятора, мы должны быть в состоянии легко проводить это разграничение буквально для каждого оператора. Уделяя недостаточно внимания этому разграничению, мы рискуем написать, в качестве "эмулятора", программу, которой не соответствует вообще никакой квантовый алгоритм.

Попробуем сообразить, в какой части вычислительной системы выполняется приведенный выше фрагмент. В квантовой части он выполняться не может — квантовый вычислитель не знает таких понятий, как "условный оператор", "цикл" или "переменная". В классической — тоже не может: классический процессор не умеет применять квантовые вентили к квантовым регистрам. Следовательно, весь "управляющий скелет", включающий в себя цикл и ветвления, выполняется в классической части, а применения каждого квантового вентиля — по отдельности — в квантовой? Но, будь это так, это сильно напоминало бы запись данных на жесткий диск порциями по одному биту. От квантового параллелизма, который распространяется далеко за пределы применения отдельного квантового вентиля, в этом случае мало что осталось бы. Но тогда что же написано в приведенном выше фрагменте и что имелось в виду?

Очевидно, автор программы, из которой взят этот фрагмент, имел в виду примерно следующее.

Квантовая схема — это суперпозиция квантовых вентилей, а применение ее к квантовому регистру эквивалентно последовательному, в правильном порядке, применению отдельных вентилей. Если разработчиками эмулятора принято решение эмулировать квантовые вентили по отдельности (а разработчиками библиотеки libquantum такое решение, несомненно, принято), то при эмуляции можно не задумываться о том, насколько мелкими порциями перемешаны в программе эмуляция отдельных квантовых вентилей (квантовая часть работы) и управляющие конструкции – условия и циклы (классическая часть работы). На скорости работы эмулятора это не отразится.

При использовании же реального квантового компьютера, или просто более реалистичного и более эффективного эмулятора, может оказаться несравненно более эффективным сначала целиком построить квантовую схему, а затем целиком (или какими-то крупными кусками, но не отдельными вентилями) ее применить к квантовому регистру. Чтобы записать именно такие действия в квантовом алгоритме, необходимо предусмотреть во входном языке эмулятора отдельные предписания для построения квантовой схемы: вместо (или вместе с) предписанием "применить квантовый вентиль X" должно появиться предписание "включить квантовый вентиль X в формируемую квантовую схему". Затем эти новые предписания придется использовать, то есть в классической части программы должна появиться отдельная часть, формирующая квантовую схему, и отдельная часть, применяющая ее к квантовому регистру.

Таким образом, учет в программе необходимости сначала сформировать и как-то запомнить квантовую схему, а только затем ее применить, усложняет программу, а предоставление программисту такой возможности усложняет входной язык эмулятора. Программисты модельных алгоритмов, записанных при помощи эмуляторов библиотечного типа, обычно не могут или не хотят идти на такие сложности, и записывают программу как произвольную смесь как угодно мелких фрагментов классической и квантовой обработки данных. При этом они используют мелкие классические вкрапления в квантовую часть обработки для того, чтобы строить квантовую схему, тут же применяя ее построенные фрагменты по ходу дела и нигде не запоминая результат построения.

Чтобы позволить себе "роскошь" так поступать при разъяснении алгоритма, мы должны быть уверены, что всегда сможем, при необходимости, легко перейти к правильному способу записи программы. Для этого нам необходим сущий пустяк — заранее, еще на этапе подготовки квантовой схемы, выполнить все те циклы и ветвления по условиям, которые при "ленивом" способе откладываются до выполнения собственно расчета.

Например, мы хотели бы выполнить заранее, в подготовительной части программы, оператор:

if(!(state & (1 << i))) quantum_sigma_x(i, reg);

где под **quantum_sigma_x()** теперь понимается действие "включить вентиль sigma_x в формируемую квантовую схему". Но для этого надо заранее, еще в подготовительной части программы, знать, чему будет равно значение **state**, и понимать, будет ли оно меняться по ходу расчета. Более того, нам надо четко представлять себе, что для разных значений **state** построенная заранее квантовая схема будет разной! Забегая вперед, отметим, что рассматриваемый нами фрагмент взят из программы поиска в базе данных, а в переменной **state** хранится значение ключа искомой записи. Следовательно, при использовании того алгоритма, из которого позаимствован наш фрагмент, для поиска записи с

каждым новым значением ключа придется строить новую квантовую схему! Это нетривиальная информация, которую легко было бы проглядеть, если бы мы не занялись, с подлинно буквоедской основательностью, выяснением вопроса о том, какой оператор программы эмуляции для какой части квантовой вычислительной системы предназначен.

На этом примере мы видим (и при разборе алгоритма Гровера убедимся в этом снова), что у квантового вычислителя нет отдельного входа для данных, в отличие от привычных нам вычислений, где присутствует разделение на данные и программы. Все нужные данные должны быть закодированы квантовыми схемами непосредственно.

4.2. Алгоритм Гровера

В качестве примера квантового алгоритма рассмотрим тестовую реализацию алгоритма Гровера. Мы не будем пытаться до конца объяснить, как алгоритм Гровера устроен внутри. Об этом написано, с разной степенью подробности, практически во всех вводных работах по квантовой информатике. Наша задача куда скромнее — постараться "вклеить" очередные "вырванные страницы". Попробуем для начала понять, что в точности делает этот алгоритм, какие данные имеет на входе, а какие — на выходе. Подавляющее большинство авторов забывают о необходимости дать эти не очень сложные, но очень необходимые пояснения, из-за чего все рассуждения о внутреннем устройстве алгоритма "повисают в воздухе". Порядок дальнейшего изложения следующий:

- общая формулировка задачи, решаемой алгоритмом Гровера;
- тестовая задача;
- отображение обрабатываемых данных на квантовый регистр;
- демонстрация бессмысленности тестовой задачи в исходной постановке, новая формулировка задачи;
- изменения в алгоритме;
- выводы, обсуждение.

Исходную постановку тестовой задачи возьмем из [1,2,3], как и сведения об отображении обрабатываемых данных на квантовый регистр. Реализацию задачи в исходной постановке возьмем из [5], этот же исходный текст будем менять, внося изменения в алгоритм.

4.2.1. Общая формулировка задачи, решаемой алгоритмом Гровера

Общую формулировку, включая пример, изложим в том виде, в каком она представлена в [3]. Алгоритм Гровера решает задачу неструктурированного поиска.

Пример **конкретной постановки**:

- пусть имеется база данных автовладельцев, в которой хранятся их фамилии и регистрационные номера их автомобилей (для простоты будем считать, что однофамильцев не бывает, а автомобиль у каждого владельца только один);

- записи в базе упорядочены по фамилиям (например, в словарном порядке);
- требуется найти фамилию владельца автомобиля с заданным регистрационным номером.

Поскольку записи в базе не отсортированы по регистрационным номерам, искать придется перебором. Смысл алгоритма Гровера — в том, что он этот перебор выполняет некоторым "правильным, квантовым" образом и за счет этого его сокращает.

Общая постановка задачи выглядит так:

- пусть на множестве всех возможных **N**-разрядных двоичных чисел задана функция, которая возвращает единицу для одного, и только одного, набора значений аргументов, и соответственно, нуль — для всех остальных наборов значений;
- требуется найти тот набор значений аргументов, на котором функция принимает значение 1.

Связь с приведенной выше конкретной постановкой состоит в том, что фамилии можно закодировать в виде целых (двоичных) чисел. Функция же имеет в качестве аргумента фамилию, а в качестве значения — ответ на вопрос, принадлежит ли автовладельцу с этой фамилией автомобиль с заданным номером. Проверяемый номер считается "встроенным" в функцию.

Такая функция, которую алгоритм Гровера будет, по мере необходимости, вызывать изнутри себя, называется **оракулом**. Оракул, строго говоря, не часть алгоритма Гровера. Снабжая алгоритм разными оракулами, мы можем решать задачу поиска в разных конкретных постановках. В полной аналогии с тем, как в классическом случае библиотечная функция, выполняющая сортировку, снабжается отдельным "оракулом", отвечающим на вопрос о том, меньше ли первый аргумент функции, чем второй. Такое вынесение проверки отношения "меньше" за пределы алгоритма сортировки позволяет сортировать одной и той же функцией, например, как числа, целые или вещественные, так и строки текста.

В действительности говорить о том, что алгоритм Гровера **вызывает** изнутри себя функцию — оракула, несколько неточно. Алгоритм Гровера является квантовым, в нем предусмотрено применение к квантовому регистру специально построенной квантовой схемы (будем называть ее квантовой схемой Гровера). Оракул также является квантовой схемой, которая при построении квантовой схемы Гровера вставляется в нее, как черный ящик, причем несколько раз.

4.2.2. Тестовая задача

Теперь мы знаем, что варианты алгоритма Гровера, решающие разные конкретные задачи, отличаются друг от друга используемыми оракулами. Так, оракул для приведенного выше примера конкретной постановки задачи имеет дело с двумя массивами: с массивом фамилий автовладельцев и массивом

регистрационных номеров. Для тривиального теста, демонстрирующего суть алгоритма и ничего больше, это слишком сложно.

Тестовая реализация алгоритма Гровера [5] устроена проще. Она не оперирует фамилиями и регистрационными номерами автомобилей, а просто ищет заданное целое число в массиве целых чисел. Это вполне укладывается в приведенную выше общую постановку. Оракул для такой программы получает на вход элемент массива и отвечает на вопрос о том, равен ли этот элемент встроенному в оракул искомому значению. Совсем скоро мы покажем, что авторы тестовой реализации несколько перестарались с упрощением постановки задачи. В такой постановке, да еще и в квантовом исполнении, задача практически лишена смысла, и нам придется эту постановку немного модифицировать в сторону усложнения.

4.2.3. Отображение обрабатываемых данных на квантовый регистр

В квантовом регистре массив целых чисел, в котором мы собираемся искать заданное число, представляется следующим образом.

Как мы помним, значение не измеренного квантового регистра — это вектор длиной $\mathbf{2^N}$, где **N** – число кубитов в регистре, а элемент вектора номер **K** кодирует вероятность того, что при измерении квантового регистра выпадет сочетание нулей и единиц, представляющее собой в двоичном виде число **K.** Будем считать, что массив целых чисел задан, если для каждого целого числа **K**, которое мы хотели бы включить в массив, вероятность получить это число при измерении регистра положительна, и все такие вероятности равны между собой, давая в сумме единицу. Тем самым, обрабатываемыми данными в нашем "векторном процессоре" являются не значения элементов вектора, как можно было бы подумать, а их номера. Обработка же должна заключаться в том, чтобы вероятность, соответствующая искомому числу, повысилась почти до единицы, за счет всех остальных вероятностей. Тогда измерение квантового регистра по окончании обработки даст нам, почти наверняка, искомое число.

4.2.4. Демонстрация бессмысленности тестовой задачи в исходной постановке, новая формулировка задачи

Сформулировав задачу так, как мы это сделали только что, в предыдущем подразделе, мы поставили себя перед необходимостью заново определить смысл слова "найти" в фразе "найти число в массиве". В самом деле, в классическом случае все числа, среди которых мы ищем одно, сложены в некоторый индексируемый массив, и результатом поиска является значение индекса, например: "поиск показал, что искомое значение 113 представлено в массиве элементом номер 7, так что результат поиска — число 7". В том представлении массива, о котором мы договорились выше, в предыдущем подразделе, понятия "номер элемента в массиве" не предусмотрено. Число 113 может "находиться" в векторе вероятностей только на позиции 113, и нигде больше. Может быть, тогда смысл слова "найти" заключается в том, чтобы

получить ответ на вопрос о том, есть ли в массиве заданное число? Попытка модифицировать исходный текст реализации алгоритма Гровера из [5] таким образом, чтобы исходные положительные вероятности распределялись поровну не между всеми числами рассматриваемой разрядности, а только между некоторыми, к успеху не приводит: алгоритм перестает работать. В этом нет ничего удивительного, если внимательно перечитать данную выше формулировку из подраздела "Общая постановка задачи, решаемой алгоритмом Гровера". Звучит она, напомним, так: "Пусть на множестве **всех возможных N-разрядных** двоичных чисел задана функция, которая ...". То есть единственным поиском, на который способна тестовая реализация из [5], а также подобные ей, является поиск заданного числа … среди всех возможных, с получением ответа: "да, среди всех возможных целых чисел рассматриваемой разрядности искомое число тоже есть". Такая постановка подразумевает оракула, отвечающего на вопрос о том, равен ли аргумент функции искомому числу, и в целом представляется слишком бессодержательной даже для тривиального теста. Зато теперь мы понимаем, почему у тестовой задачи "найти заданное число в массиве" в [5] не предусмотрено ни в каком виде аргумента под названием "массив, в котором искать". Его там просто не может быть.

Единственный смысл, который можно усмотреть в такой тестовой задаче, состоит в том, чтобы продемонстрировать работоспособность инвариантной части алгоритма на тривиальном оракуле. Нам такая задача представляется недостаточно интересной.

Чтобы придать постановке тестовой задачи хоть какую-то минимальную осмысленность, нам придется снабдить ее хотя бы тривиальной моделью базы данных. В качестве таковой будем использовать целочисленный массив, длиной $\mathbf{2^N}$, который содержит произвольную перестановку всевозможных **N**-разрядных целых чисел. Задача алгоритма Гровера будет заключаться в том, чтобы найти позицию искомого целого числа в этом массиве.

Например:

Пусть дана следующая перестановка всевозможных 3-разрядных целых чисел:

5, 3, 1, 7, 0, 2, 4, 6

Тогда результатом поиска числа 7 в этой перестановке будет число 3, а результатом поиска числа 5 — число 0 (элементы массива занумерованы от нуля). У такой тестовой задачи, в отличие от исходного варианта из [5] и ему подобных, исходные данные включают в себя не только конкретное число, **которое** надо найти, но и конкретный массив, **в котором** его надо найти. Результатом поиска, как и в классическом случае, является позиция в массиве, на котором обнаружено искомое число.

4.2.5. Изменения в алгоритме

Планируя изменения в исходной версии алгоритма из [5], попробуем, помимо задачи, решить еще сверхзадачу. Выше мы отмечали, что реализация

алгоритма Гровера состоит из двух частей: собственно алгоритма, реализующего поиск "правильным, квантовым" образом, и оракула — "сменной насадки", умеющей по значению аргумента просто отвечать на вопрос, "то" это значение или "не то". Попробуем реализовать изменения, запланированные в предыдущем подразделе, изменив только оракула и оставив без изменения собственно алгоритм. Тогда нам придется оставить без изменения рассмотренное выше представление всевозможных целых чисел в виде вероятностей, а массив перестановок, в котором мы теперь собираемся искать, как-то добавить к новой программе.

Поскольку мы собираемся менять текст оракула, необходимо посмотреть внимательно на его исходный вариант:

```
void oracle(int state, quantum_reg *reg)
{
 int i;

 for(i=0;i<reg->width;i++)
  {
   if(!(state & (1 << i))) quantum_sigma_x(i, reg);
  }
 quantum_toffoli(0, 1, reg->width+1, reg);
 for(i=1;i<reg->width;i++) { quantum_toffoli(i, reg->width+i,
                                reg->width+i+1, reg); }
 quantum_cnot(reg->width+i, reg->width, reg);
 for(i=reg->width-1;i>0;i--) { quantum_toffoli(i, reg->width+i,
                                 reg->width+i+1, reg); }
 quantum_toffoli(0, 1, reg->width+1, reg);
 for(i=0;i<reg->width;i++)
  {
   if(!(state & (1 << i))) quantum_sigma_x(i, reg);
  }
}
```

По уже установившейся традиции, начнем с "вклейки вырванных страниц".

Из сформулированной выше общей постановки задачи, решаемой алгоритмом Гровера, следует, что оракул должен быть функцией, на входе которой — целое число, а на выходе — ответ на вопрос, равно ли оно искомому. Как это соотносится с приведенным здесь исходным текстом?

В этом тексте мы видим два аргумента, передаваемых в оракул. Первый из них (**int state**) — как раз искомое число, второй (**quantum_reg *reg**) — указатель на квантовый регистр. Искомое число будет неизменным на протяжении каждого запуска программы, квантовый регистр тоже будет всегда один и тот же, поскольку он в этой программе всего один. Где же тогда то

целое число, которое оракул должен сравнивать с искомым и которое при разных обращениях к оракулу будет, естественно, разным?

Чтобы это осознать, вспомним, что оракул у нас — квантовый, то есть за одно срабатывание он сравнивает с искомым не одно число, а $\mathbf{2^N}$ чисел, где **N** — количество кубитов в квантовом регистре. Числа эти — конечно же, базовые состояния квантового регистра. У того из них, которое совпадет с искомым, оракул обязан инвертировать фазу соответствующего ему элемента вектора. Таким образом оракул сообщает вызвавшей его программе о своем решении.

Отметив, что оракул у нас — квантовый, мы вынуждены применить к его исходному тексту рассуждения из подраздела "Библиотека libquantum" в начале настоящего раздела. Тем более что рассмотренный в этом подразделе фрагмент исходного текста, очевидно, взят именно из текста нашего оракула. Легко видеть, что записанный явно "ленивым способом" текст оракула можно переписать "правильным способом", если зафиксировать заранее как ширину квантового регистра, так и значение искомого числа (о "ленивом способе" и "правильном способе" подробно рассказывается в подразделе "Библиотека libquantum"). Получается, что значение искомого числа, действительно, буквально встроено в квантовую схему. Изготовление квантовой схемы, строение которой не зависело бы от значения искомого числа, может быть, и возможно, но в данном случае не предусмотрено. Обратим внимание на то, что все приведенные здесь выводы мы сделали, совсем ничего не зная о конкретном смысле функций **quantum_sigma_x()**, **quantum_toffoli()** и **quantum_cnot()**, а зная только то, что они означают применение к квантовому регистру некоторых квантовых вентилей.

Теперь нам предстоит все-таки узнать об этих квантовых вентилях немного больше. Это легко сделать, обратившись, например, к [1,2,3]. Ознакомившись с матрицами унитарных преобразований **quantum_sigma_x()** и **quantum_toffoli()**, убеждаемся, что обе они вещественны и симметричны. Поскольку они еще и унитарны, это означает, что каждое из этих преобразований является обратным само для себя. Если применить такое преобразование к произвольному вектору дважды, то при первом применении вектор как-то поменяется, а при втором — вернется в исходное состояние. Из приведенного текста вполне очевидно, что именно на этом основана работа оракула. Весь его текст легко разделить на две части: до обращения к **quantum_cnot()** и после, причем первая часть является "зеркальным отражением" второй. В первой части квантовый регистр преобразуется таким образом, чтобы применение к нему **quantum_cnot()** вызвало необходимую инверсию фазы у элемента вектора с искомым номером. Затем, во второй части, все изменения, кроме этой самой инверсии фазы, ради которой все и затевалось, "откатываются обратно". Что и требовалось от оракула.

Теперь разберемся, как конкретно работает оракул.

Начнем с цикла:

```
for(i=0;i<reg->width;i++)
```

```
{
  if(!(state & (1 << i))) quantum_sigma_x(i, reg);
}
```

Значение **reg->width** мы в предыдущих рассмотрениях обозначали как **N**, то есть это разрядность наших двоичных чисел. Приведенный выше цикл, таким образом, обходит все разряды двоичного числа и для тех из них, которые в значении искомого числа (**state**) равны нулю, обращается к функции **quantum_sigma_x(i, reg)**. Обращение к этой функции применяет к квантовому регистру **reg** очень простой квантовый вентиль, который меняет местами любые два элемента вектора вероятностей, если их номера отличаются, в двоичном представлении, только значением бита номер **i**. Например, если значение **i** равно нулю, поменяются местами нулевой элемент с первым, второй с третьим, и так далее, все четные с нечетными, попарно.

Поскольку речь идет, в данном случае, только о перестановках элементов вектора, мы можем мыслить их, если нам это удобно, **не как перестановки значений элементов, а как перестановки их номеров**. Встав на эту точку зрения, легко увидеть, что рассматриваемый нами цикл меняет номер элемента **state**, то есть искомый, на номер, состоящий в двоичном виде из всех единиц. Все другие номера он поменяет на новые номера с другими значениями.

Теперь задача оракула упростилась. Она свелась к необходимости инвертировать фазу не у элемента вектора с произвольным номером **state**, а у элемента с совершенно конкретным номером, состоящим в двоичном представлении из всех единиц. Посмотрим, что для построения такой квантовой схемы нам нужно и что из этого у нас есть.

Очевидно, нужны нам две вещи:

- квантовые вентили условной перестановки элементов вектора вероятностей;
- второй, дополнительный набор элементов вектора, с такими же вероятностями, как в основном наборе, но с инверсной фазой, чтобы было, что с чем переставлять.

О дополнительном наборе элементов вектора алгоритм Гровера позаботился сам, причем за пределами оракула. Пусть, для конкретности, **N** (оно же **reg->width**) равно 4. Тогда квантовый регистр должен состоять из 4 кубитов и иметь вектор состояния из 16 элементов. Поскольку мы имеем дело с эмулятором, а не с настоящим квантовым компьютером, у нас есть возможность напечатать вектор состояния квантового регистра, например, при входе в оракул. Сделав это, убеждаемся, что вектор состояния состоит не из 16, а из 32 элементов, причем элементы с нулевого по 15-й отличаются от соответствующих элементов с 16-го по 31-й только инвертированной фазой. Значит, нам осталось решить совсем простую задачу: инвертировать 4-й разряд в номере элемента вектора (фактически, применить **quantum_sigma_x(4,reg)**), но сделать это не со всеми номерами, а с тем и только с тем, который в двоичном представлении состоит из всех единиц.

Нам, очевидно, потребуются библиотечные функции условного применения **quantum_sigma_x()**. Таковых имеется две: **quantum_cnot()** и **quantum_toffoli()**.

quantum_cnot(c1,i,reg) делает то же, что и **quantum_sigma_x(i,reg)**, но при условии, что в переставляемых номерах элементов вектора разряд с номером **c1** равен единице.

quantum_toffoli(c1,c2,i,reg) отличается от **quantum_cnot(c1,i,reg)** тем, что условием выполнения перестановки является равенство единице двух разрядов, **c1** и **c2**.

Следовательно, из базовой возможности проверки на равенство единице одного и двух разрядов надо сделать проверку **N** разрядов.

Это делается следующим фрагментом программы:

```
quantum_toffoli(0, 1, reg->width+1, reg);
for(i=1;i<reg->width;i++) { quantum_toffoli(i, reg->width+i,
                                reg->width+i+1, reg); }
```

Чтобы понять смысл этого фрагмента, вспомним еще раз, что мы имеем дело с таким преобразованием вектора состояния квантового регистра, которое сводится не более чем к перестановкам элементов вектора. Это дает нам право трактовать его, если нам так удобно, как преобразование не значений элементов вектора, а их номеров. Приведенный фрагмент пытается выстроить в номере каждого элемента вектора, добавляя к нему старшие разряды, сплошную цепочку из единиц длиной **N**. Очередная единица появляется в цепочке только тогда, когда на соответствующей позиции в исходном номере тоже стоит единица. Если хотя бы на одной позиции в исходном номере стоит нуль, цепочка не дотянется до задуманной длины. Последний же разряд в цепочке как раз и есть условие инверсии фазы, выполняемой обращением к **quantum_cnot()**.

Мы закончили разбор первой части (до обращения к **quantum_cnot()**) исходного варианта оракула. Вторая его часть (после обращения к **quantum_cnot()**) просто восстанавливает исходный вид вектора состояния, за исключением инверсии фазы найденного элемента вектора. Этим замечанием разбор исходного варианта оракула завершен полностью.

Теперь необходимо написать модифицированный вариант оракула. Отметим, что похожая (но в более сложной постановке) задача решалась, как минимум, один раз [6]. Мы же рассмотрим совсем простое решение для постановки, описанной выше, в подразделе "Демонстрация бессмысленности задачи в исходной постановке, новая формулировка задачи". Для этого нам придется всего лишь встроить массив перестановок в квантовый вычислитель. Не будем сводить это встраивание к конкретным квантовым вентилям конкретной библиотеки, покажем лишь, что задача эта принципиально очень простая.

Утверждение это основано на том, что матрица перестановки элементов вектора унитарна (хотя и не симметрична), как и обратная к ней. Тем самым,

ничто не мешает нам добавить в начало первой части оракула прямую, а в конец второй части — обратную перестановку. Задача запланированной модификации тестового варианта алгоритма Гровера в принципе решена. Правда, ценой встраивания конкретного вида перестановки в квантовую схему, наряду с разрядностью чисел и искомым значением.

4.2.6. Выводы, обсуждение

Первый вывод, который буквально напрашивается даже после такого поверхностного, как у нас, разбора алгоритма Гровера — это вывод о критической важности всего, что связано с представлением и хранением квантовой схемы. Возможности квантового компьютера очевидным образом ограничены числом кубитов, из которых складывается его квантовый регистр. Но ведь квантовый регистр — это, по аналогии с классическим компьютером, "память данных". Где же тогда "память программ"? В ней должна храниться квантовая схема, и этой памяти, как и любой другой, наверняка присущи какие-то ограничения. На уровне сформулированной нами в этой работе квантовой модели вычислений этого не видно совсем. О квантовой схеме принято говорить, что она может реализовать любое унитарное преобразование, а вообще-то про ее представление и хранение мало что известно [3]. Все сказанное в этом разделе, тем не менее, ясно показывает, что многое не только из самого квантового алгоритма, но и из того, что в классическом случае имело бы форму исходных данных, может быть "зашито" в квантовую схему. В связи с этим возникают совершенно естественные вопросы, например:

- каков объем памяти, хранящей квантовую схему (слово "память" здесь, конечно, понимается в максимально широком смысле)?
- как быстро эта "память" записывается? Надежна ли она, или же подвержена какой-нибудь порче, вроде декогеренции?
- какие тут вообще имеются ограничения, которые, возможно, даже не приходят нам в голову, но которые, тем не менее, важны?

Ответы на эти вопросы точно существуют [8], но пока мы не готовы их сформулировать в тех терминах, в которых мы постарались описать квантовую модель вычислений. Отсутствие должного учета этих факторов может привести к ситуации, хорошо знакомой классическим схемотехникам, когда схема, спроектированная для укладки в ПЛИС, прекрасно эмулируется, но не трассируется на необходимой частоте, то есть не может быть реализована физически.

Второй вывод из рассказанного в этом разделе касается наличия довольно простой и очень важной связи между классической и квантовой схемотехникой, которую также принято почему-то "помещать на вырванных страницах".

Во многих работах [1,2,3] объяснению, что такое квантовый вентиль, предшествует рассказ о том, что такое вентиль классический. Приводятся примеры обратимых классических вентилей: инвертора, условного "не" и вентиля Тоффоли. Далее вводятся их квантовые аналоги: квантовое условное

"не" и квантовый вентиль Тоффоли. Степень "родства" между классическими и квантовыми "однофамильцами" остается невыясненной. С одной стороны, речь идет, вроде бы, не более чем об аналогии.

В действительности, как мы видели в этом разделе, связь между одноименными классическими и квантовыми вентилями есть, и довольно прямая. Правда, касается она не всех квантовых вентилей, а только тех, действие которых сводится к перестановке элементов вектора состояния квантового регистра. В нашем разборе работы оракула для алгоритма Гровера мы эту связь фактически установили. Состоит она в том, что действие таких квантовых вентилей можно представлять себе не только как действия со значениями элементов вектора, но и как действия с их номерами. При этом действия с номерами описываются одноименными классическими вентилями: например, для квантового условного "не" — классическим условным "не". Все это особенно хорошо видно при изучении исходных текстов библиотеки [5].

5. О проблеме эффективной эмуляции квантового вычислителя

Всевозможных эмуляторов квантовых вычислений существует великое множество [4,5,7,8]. При разработке большинства из них задача ускорения эмуляции всерьез не ставилась, поскольку предназначены эти программы не для расчетов, а для изучения работы квантовых алгоритмов на сравнительно небольших примерах. Вплоть до конца этого раздела мы будем полагать, что задача ускорения действительно сложных в вычислительном отношении квантовых эмуляций действительно стоит, и попробуем обозначить возможные пути ее решения.

При эмуляции квантовых вычислений на классическом компьютере приходится решать две проблемы:

- проблему экспоненциального (относительно числа кубитов) объема памяти;
- проблему объема вычислений, необходимых для восполнения "квантового параллелизма" классическими средствами.

Проблему объема памяти решить в общем виде нельзя, но во многих частных случаях можно решить частично. Вектор состояния квантового регистра во многих алгоритмах оказывается сильно разреженным, и правильный выбор формы представления разреженного вектора в памяти классического компьютера может создать у программиста иллюзию наличия памяти очень большого объема. Тот способ представления, который выбран, например, в [5], во многом удобен, но вовсе не является единственным возможным.

Казалось бы, проблема необходимого объема вычислений не имеет даже такого частичного решения, как проблема необходимого объема памяти. В действительности это, к счастью, не так.

Вспомним, что нам известно о природе квантового параллелизма. Состоит он, грубо говоря, в том, что квантовый вычислитель выполняет унитарные преобразования вектора состояния как единые, элементарные "операции", затраты времени на которые не зависят от числа кубитов. В то же время для их классической эмуляции требуется объем вычислений, пропорциональный длине вектора состояния. Поскольку длина эта, в свою очередь, экспоненциальна относительно количества кубитов, попытка "угнаться" за квантовым параллелизмом при помощи параллелизма классического быстро становится безнадежной. Проблема здесь, как мы видим, не в том, что квантовый компьютер что-то делает быстро, в то время как классический — медленно, а в том, что квантовый компьютер это "что-то" не делает совсем.

Ситуация не была бы такой безнадежной, если бы мы нашли способ избежать прохода циклом по элементам вектора состояния при эмуляции унитарных преобразований. В общем случае этого сделать, скорее всего, нельзя, но в довольно массовом частном случае решение лежит буквально на поверхности. Речь идет о тех самых, не раз уже упомянутых, квантовых вентилях, действие которых сводится к систематическим манипуляциям с отдельными разрядами номера элемента в массиве. Для реализации таких перестановок вовсе не требуется обходить массив — достаточно запомнить, что впредь, при индексации массива, зачем бы она ни понадобилась, некоторые разряды в значении индекса надо переставлять. Особенно дешево это реализуется при схемном исполнении эмулятора, например, в ПЛИС. В этом случае можно было бы говорить о **классической реализации квантового параллелизма** (частичной, конечно).

Надо сказать, что работы по схемной реализации квантовых эмуляторов в ПЛИС ведутся, особенно — за рубежом, довольно интенсивно и уже не первый год [7]. При этом сообщается о возможности ускорения эмуляции в 3-4 порядка по сравнению с [5]. По нашему мнению, сравнение с [5] не вполне отражает преимущества именно схемной реализации, поскольку реализация [5] сама может, при необходимости, быть значительно ускорена, просто путем переписывания программы в более эффективном виде. Для оценки перспективности схемного подхода мы полагаем необходимым проделать похожую работу.

6. Благодарности

Авторы благодарны Смольянову Ю.П. за идею этой работы, а также Климову Анд.В. и Романенко С.А. за ценные замечания.

Литература

1. Хренников А.Ю. **Введение в квантовую теорию информации.** — М.: ФИЗМАТЛИТ, 2008. — 284 с. — ISBN 978-5-9221-0951-2.

2. Федотов И.Е. **Модели параллельного программирования**. — М.: СОЛОН-ПРЕСС, 2012. — 384 с. — ISBN 978-5-91359-102-9.

3. Душкин Р.В. **Квантовые вычисления и функциональное программирование**. — М.:ДМК Пресс, 2015. — 232с.: ил. — ISBN 978-5-97060-275-1.

4. Главная страница проекта квантового компьютера IBM Q на сайте исследовательского подразделения IBM.

URL:https://www.research.ibm.com/ibm-q/ Дата обращения 25.07.2018.

5. Главная страница свободно распространяемой библиотеки моделирования квантовой механики libquantum.

URL:http://www.libquantum.de Дата обращения 31.07.2018.

6. Ross D.A. **A Modification of Grover's Algorithm as a Fast Database Search** Department of Physics and Astronomy, University of Southampton, Southampton SO17 1BJ, United Kingdom July 1998

URL:https://www.researchgate.net/publication/2203163_A_Modification_of_Grover%27s_Algorithm_as_a_Fast_Database_Search Дата обращения 4.08.2018.

7. Lee Y. H., Khalil-Hani M., Marsono M. N. An FPGA-Based Quantum Computing Emulation Framework Based on Serial-Parallel Architecture. International Journal of Reconfigurable Computing. Volume 2016, Article ID 5718124, 18 pages

URL:http://dx.doi.org/10.1155/2016/5718124. Дата обращения 6.08.2018.

8. Ryan LaRose. Overview and Comparison of Gate Level Quantum Software Platforms. Department of Computational Mathematics, Science, and Engineering, Michigan State University. (Dated: June 22, 2018) https://quantumcomputingreport.com/wp-content/uploads/2018/06/Overview-and-Comparison-of-Gate-Level-Quantum-Software-Platforms-Final-June-21-2018.pdf Дата обращения 31.08.2018.

Оглавление