Глава 1. Анализ задачи

* 1. История развития квантовых вычислений

Современные компьютеры – и в теории (Машины Тьюринга), и на практике, основаны на классической физике. Они ограничены, тем фактом, что система может быть только в одном состоянии. Используя современные знания в квантовой физике, можем предложить концепцию квантовой системы, которая может находится в суперпозиции нескольких различных состояний одновременно. Более того, территориально разделенные квантовые системы могут быть переплетены друг с другом и благодаря этому, операции перестают быть локальными.

Квантовые вычисления рассматривают вычислительные мощности и другие свойства компьютеров, спроектированные на основе принципов квантовой механики. Основная задача состоит в поиске квантовых алгоритмов, которые значительно быстрее любого классического алгоритма, решающего такую же проблему. Впервые формальная модель универсального квантового компьютера была предложена П. Бениоффом и развита Д. Дойчем.

Квантовое состояние представляет собой суперпозицию классических состояний, которые можно измерить или применить унитарную операцию. Представим, что некая физическая система, которая может быть в N различных классических состояний. Назовем эти состояния |1>, |2>, …,|N>. Грубо говоря, под классическим понимается состояние, в котором система может быть измерена. Квантовое состояние |φ> это суперпозиция классических состояний: |φ> = α1|1> + α2|2> +…+ αN|N>, где αi – комплексное число. Таким образом, можно сказать, что система в квантовом состоянии находится во всех классических состояниях одновременно. Говоря математическим языком, состояние |1>, ..., |N> формирует ортонормальный базис N-размерного Гильбертова пространства, в котором квантовое состояние |φ> является вектором. С квантовым состоянием можно проводить 2 операции: измерить и изменить унитарно, без измерения.

Как мы уже выяснили, информация может быть закодирована и использована на основе принципов квантовой механики. Задача обработки квантовой информации состоит в решении определенного класса проблем, которых не могут решить классические компьютеры за приемлемое время.

* 1. Преимущества и слабости квантовых вычислений.

Важнейшим преимуществом квантовой системы является квантовый параллелизм – возможность регистра находиться одновременно во всех своих состояниях. Классический n-битный регистр может находиться только в одном состоянии, в то время как n-битный квантовый регистр находится сразу во всех 2N базисных состояниях. Для получения однозначного ответа для квантовых вычислений необходимо провести преобразование состояния таким образом, чтобы нужный ответ получил бы большую амплитуду, то есть проявился при измерении с большей вероятностью.

Слабости квантовых вычислений являются продолжением их сильных сторон: ответ можно получить, лишь в результате измерения, которое является вероятностным процессом и приводит к безвозвратной потере информации об амплитудах полученных базисных состояний. Кроме того, в классических алгоритмах можно прервать вычисления, если ответ уже получен. Квантовые алгоритмы всегда выполняются до конца, что также требует специальной организации. Еще одна особенность квантовых вычислений – обратимость используемых преобразований – не представляет проблемы, если нет ограничений на размер квантового регистра. Однако при довольно умеренных ограничениях вычислений многих функций оказывается затруднено, а соответствующие задачи в таких моделях с ограничениями могут иметь экспоненциальную сложность.

Перечислим 3 основных мотиваций изучения квантовых компьютеров:

Процесс миниатюризации, который сделал современные компьютеры мощными и дешевыми, практически достиг микро-уровней, на которых проявляются квантовые эффекты. Производители чипов склонны перейти к большим размерам, чтобы подавить эти эффекты, но стоит попробовать сработаться с ними, открывая путь к дальнейшей миниатюризации.

Использование квантовых эффектов позволяет значительно ускорить некоторые вычисления и даже реализовать некоторые вещи, недоступные классическим компьютерам. Основная цель данной работы – демонстрация этих идей.

Одна из задач теории компьютерной науки звучит как «выявить возможности и ограничения самого допустимо-сильного вычислительного устройства, которое может позволить нам природа».

* 1. Когда же будет построен квантовый компьютер?

Первый 2х-кубитный компьютер был построен в 1997, а в 2001 5ти-кубитный компьютер успешно разложил число 15. На текущий момент, самый большой квантовый компьютер имеет несколько десятков кубит.

Хотя, уже доступны компьютеры с небольшим количеством кубит, практическая реализация более мощного квантового компьютера выглядит достаточно трудной задачей. Квантовая система всегда взаимодействует со своим окружением. Это взаимодействие влияет на состояние системы, вызывая потерю данных. Процесс влияния окружения на квантовую систему называется декогерированием. Проблема шумов и декогерирования в теории может быть решена с помощью квантового исправления ошибок и устойчивых алгоритмов вычисления, но эти проблемы все еще не решены на практике. Давид П. ДиВинцензо предложил список необходимых условий (критерии ДиВинцензо) для идентификации системы как квантового компьютера. (DiVincenzo, David P. (2000-04-13). "The Physical Implementation of Quantum Computation").

Критерии ДиВинцензо:

1. Масштабируемая физическая система с четко выраженными кубитами. Нам необходим квантовый регистр из кубитов для хранения информации, так же как и классическому компьютеру требуется память. Самый простой способ физически реализовать кубит – использовать 2х уровневую квантовую систему. Для примера, электрон или фотон могут быть кубитами.
2. Возможность инициализировать состояние кубитов как начальное |00…0>. При отсутствии возможности сбросить состояние классического компьютера, нельзя доверять выводам вычислений, даже если процесс прошел корректно.
3. Время на декогерирование значительно длиннее, чем время на простейшую операцию. Декогерирование возможно наибольшая помеха к строительству жизнеспособного квантового компьютера. Декогерирование ведет к деградации многих возможностей квантового состояния из-за взаимодействия с окружающей средой, а также увеличивает время квантовых вычислений. Само по себе декогерирование не так важно. Гораздо важнее соотношение между временем декогерирования и времен на выполнение вентиля.
4. Универсальный набор квантовых вентилей. Допустим, есть классический компьютер с большой памятью. Необходимо управлять данными в этой памяти через различные логические вентили. Должна быть возможность применить независимый логический оператор на эту память, чтобы осуществить информационный процесс.
5. Возможность измерения кубитов. Результат классических вычислений должен быть выведен на экран или распечатан. В то время как процесс вывода на классическом компьютере достаточно тривиальная часть вычислений, это критически важная часть квантовых вычислений. После выполнения квантового алгоритма, состояние должно быть измерено, чтобы получить результат.
6. Возможность взаимопревращать стационарные и изменяющиеся кубиты. Некоторые реализации отличны в хранении квантовой информации, в то время как продолжительные трансформации квантовой информации могут потребовать дополнительных физических ресурсов. Это можно сравнить с обычным компьютером, в котором CPU и память сделаны из полупроводников, в то время как жесткий диск используется как устройство для хранения. Также и рабочий квантовый компьютер может потребовать несколько видов кубитов для представления квантовых вычислений.
7. Возможность надежно передавать изменяющиеся кубиты между разными локациями. Не приходится говорить, что это необходимое требование квантовой коммуникации, такой как распределенный квантовый ключ. Это условие также важно в распределенных квантовых вычислениях.
   1. Кубиты и квантовая память

В классических вычислениях единицей информации является бит, которые может быть 0 или 1. В квантовых вычислениях используются квантовые биты (кубиты), которые могут быть в суперпозиции 0 и 1. Рассмотрим систему, которая может быть в двух базовых состояниях, назовем их |0> и |1>. Будем идентифицировать эти состояния как векторы (1, 0) и (0, 1) соответственно. Представим кубит как α0|0> + α1|1>, |α0|^2+ |α1|^2= 1. Квантовому компьютеру необходимо как минимум 102 – 103 кубит, для выполнения алгоритмов, более эффективных, чем их классические аналоги.

* 1. Измерение по вычислительному базису

Допустим, мы измеряем состояние |φ>, в таком случае, мы увидим только одно классическое состояние |j>. Конкретное |j> неизвестно заранее, оно проявится с вероятностью |αj|2, что представляет собой квадратичную норму соответствующей частоты aj (|a + ib| =√a2+ b2). При этом, после измерения квантовое состояние |φ> пропадает и остается классическое состояние |j>. Другими словами, измерение |φ> разрушает квантовую суперпозицию |φ> до классического состояния |j>, а вся остальная информация из αi пропадает.

* 1. Унитарные преобразования.

Вместо измерения |φ>, также можно применить к нему некоторый оператор, то есть поменять состояние на |ψ> = β1|1> + β2|2> +…+ βN|N>. Квантовая механика допускает применение к квантовым состояниям только линейных операторов. Это значит, что если мы рассматриваем состояние |φ> как N-мерный вектор (α1, . . . , αN)T, то применение оператора, изменяющего |φ> на |ψ> соответствует умножению |φ> на NxN комплексную матрицу U:

* 1. Квантовые вычислительные операции.

Рассмотрим, как квантовый компьютер может совершать вычисления над регистром кубитов. Рассмотрим метод, под названием квантовая модель цепи(quantum circuit model). В классической теории, логическая цепь (Boolean circuit) представлена в виде конечного направленного незамкнутого графа с вентилями(gates) AND, OR и NOT. Входные биты проходят через эти вентили и, согласно цепи, принимают некоторые выходные значения. Будем считать, что цепь вычисляет некоторую булевскую функцию   
f : {0, 1}n→ {0, 1}m если итоговый узел содержит подходящее значение f(x) для каждого Х∈ {0, 1}n.

Семейство цепей - это набор C = {Cn} цепей, для каждого входного набора размером n. Каждая цепь имеет один выходной бит. Такая семья определяет язык L ⊆ {0, 1}∗= ∪n≥0{0, 1}n если, для каждого n и каждого ввода x∈ {0, 1}n, цепь n возвращает 1 если x ∈ L и возвращает 0 в противоположном случае. Семейство цепей называется равномерно полиномиальной, если есть детерминированная машина Тьюринга, которая возвращает цепь Cn в ответ на ввод n, используя пространство логарифмов из n. Заметим, что размер (количество вентилей) цепи Сn может расти полиноминально из n. Известно, что равномерно полиномиальное семейство цепей будет эквивалентно полиномиальной детерминированной машине Тьюринга: язык L может быть описан как равномерно полиномиальное семейство цепей, если L ∈ P, где P – класс языков, допустимых для полиномиальных машин Тьюринга.

Таким же способом можем описать вероятностные цепи, которые на вход, в дополнение к n входных бит получат несколько случайных битов(“coin flips”). Вероятностна цепь вычисляет функцию f если она успешно возвращает правильный ответ f(x) с вероятностью не менее 2\3 для каждого x. Вероятностные цепи эквиваленты вероятностной машине Тьюринга: язык L может быть описан равномерно полиномиальной вероятностным семейством цепей, если L ∈ BPP, где BPP (“Bounded-error Probabilistic Polynomial time”) класс языков, которые могут быть эффективно обработаны вероятностной машиной Тьюринга с шансом на успех не менее 2\3.

Глава 2. Описание задачи

2.1. Постановка задачи

Данная работа имеет несколько целей, связанных с теорией квантовых компьютеров. Исследовательская цель состоит в изучении принципов работы квантовых компьютеров, алгоритмов квантовых вычислений и их реализации. После рассмотрения всех аспектов использования алгоритмов квантовых вычислений для различных целей, необходимо провести сравнительный анализ с классическими способами решения этих же задач. На основании сравнений, необходимо сделать вывод о преимуществах и недостатках алгоритмов квантовых вычислений. В ходе работы будут рассмотрены следующие задачи и алгоритмы их решения.

* Алгоритм Дейча-Джозы для определения сбалансированности функции
* Алгоритм Шора для факторизации числа
* Квантовый обход графа
* Квантовый алгоритм для систем линейных равенств (Алгоритм ХХЛ)

Практическая часть работы заключается в непосредственной реализации вышеупомянутых алгоритмов на симуляторе квантового компьютера.

2.2. Ранние алгоритмы.

Два наиболее крупных достижения квантовых алгоритмов — это алгоритм Шора для факторизации числа и алгоритм поиска Гровера. Для начала, опишем некоторые идеи, предшествующие им. Все квантовые алгоритмы работают с очередями в той или иной форме. Представим N-битный ввод x = (x0,…,xn-1) ∈ {0, 1}N. Обычно мы имеем N=2n, поэтому мы можем получить доступ к биту xi, используя n-битный индекс i. В данном случае ввод можно представить как N-битную память, в которой мы можем получить доступ к любому элементу по его индексу. Представим в виде квантовой операции Ox: |i, 0> → |i, xi>. Первые n кубитов в состоянии также называются адресными кубитами или адресными регистрами, в то время как следующий n+1 кубит называется целевым кубитом.

2.3. Алгоритм Дейча.

Данный алгоритм один из первых, показавших превосходство квантовых алгоритмов над классическими. Невзирая на простоту, в алгоритме используется на полную принцип суперпозиции.

Пусть f: {0, 1} => {0, 1} бинарная функция. Заметим, что возможно только четыре значения для f

f1:0→0, 1→0,f2:0→1, 1→1,

f3:0→0, 1→1,f4:0→1, 1→0.

В первых двух случая, f1 и f2 назовем постоянными, а функции f3 и f4 сбалансированными. Для классического случая, для определения сбалансированности f, необходимо вычислить ее дважды, в то время как с использованием алгоритма Дейча хватит одного вычисления.

Пусть |0> и |1> соответствует классическим битам 0 и 1, определим состояние |ψ0>=(|00>−|01>+|10>−|11>). Применим f на это состояние в качестве унитарного оператора Uf : |x, y> →|x, y⊕f(x)>, где ⊕ сложение по модулю два(XOR). Для точности, мы имеем:

|ψ1>=Uf|ψ0>=(|0,f(0)>−|0,1⊕f(0)>+|1,f(1)>−|1,1⊕f(1)>)=

(|0,f(0)>−|0,¬f(0)>+|1>,f(1)>−|1,¬f(1)>),

где ¬ соответствует отрицанию. Таким образом, данная операция не что иное, как вентиль CNOT с контрольным битом f(x); целевой бит y только в том случае, когда f(x)=1 и остается прежним во всех остальных случаях. В результате, мы применяем вентиль Адамара на первый кубит, чтобы получить   
|ψ2>=(UH⊗I)|ψ1>=|√2[(|0>+|1>)(|f(0)>−|¬f(0)>)+(|0>−|1>)(|f(1)>−|¬f(1)>)].

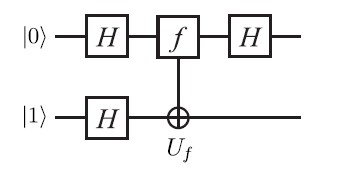
Волновая функция упрощается до:

|ψ2>=1/√2|0>(|f(0)>−|¬f(0)>)

В случае если f постоянная, для которой |f(0)>=|f(1)>, и

|ψ2>=1/√2|1>(|f(0)>−|¬f(0)>), для |¬f(0)>=|f(1)>. Таким образом, значение первого кубита покажет является ли функция f постоянной или сбалансированной.

Составим квантовую цепь для имплементации алгоритма Дейча. Для начала применим трансформацию Адамара W2 =UH⊗UH на |01> для получения |ψ0>. Определим условный вентиль Uf, то есть управляемый вентиль NOT с контрольным битом f(x) и действием Uf :|x, y>→|x, y⊕f(x)>. Затем вентиль Адамара применяется к первому кубиту, до его измерения.



В квантовой цепи мы считаем вентиль Uf своеобразным черным ящиком, т.е. алгоритмом, для которого известны лишь входные и выходные данные. Такие структуры иногда называются оракулом, а вентиль Uf называется оракулом Дейча. Таким образом, имея матрицу Uf, можно составить цепь на рисунке 1 и применить ее на входное состояние |01>. После этого, можно сказать является f постоянной или сбалансированной за одно применение Uf.

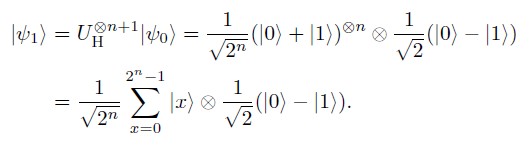
2.4. Алгоритм Дейча-Джоза

Алгоритма Дейча может быть обобщен в виде алгоритма Дейча-Джоза. Допустим, есть некая бинарная функция f:Sn≡{0,1,...,2n−1}→{0,1}. Будем считать, что f может быть либо постоянной, либо сбалансированной. Когда f постоянная, она принимает значение 0 или 1 соответственно входному значению x. Сбалансированная функция принимает значение f(x)=0 для половины x ∈Sn и f(x)=1 для остальных. Другими словами,   
|f−1(0)|=|f−1(1)| =2n−1, где |A| указывает на количество элементов во множестве A, известное как мощность множества А, Хотя существуют функции, которые не являются ни постоянными, ни сбалансированными, мы не будем рассматривать здесь эти случаи. Задача найти алгоритм, который определит, является ли f постоянной или сбалансированной за наименьшее число вычислений f.

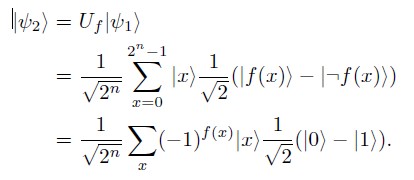
Очевидно, что необходимо как минимум 2n-1 + 1 шагов в худшем случае с классическими манипуляциями, для утверждения что f(x) постоянная или сбалансированная со стопроцентной вероятностью. Покажем, что количество шагов сократится до одного, при использовании квантового алгоритма.

Подготовить n+1 кубитный регистр в состоянии *|ψ*0>= *|*0>*⊗n ⊗ |*1*>.* Первые n кубит работают как входные кубиты, в то время как n+1й кубит будет служить своеобразной «временной памятью», такие кубиты для хранения временной информации назовем служебными кубитами.

Применим трансформация Уэлша-Адамара к регистру. Получим состояние:

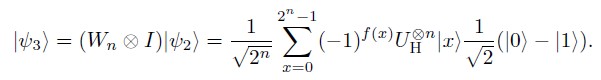


Применим f(x)-управляемый вентиль NOT к регистру, который изменит n+1 кубит только в том случае, если для входного кубита f(x) = 1. Следовательно, необходим вентиль Uf, который посчитает f(x) и повлияет на регистр как Uf |x>|c>= |x>|c⊕f(x)>, , где |c> однокубитное состояние n+1го кубита. Заметим, что |c> обратилось только тогда, когда f(x) = 1 и осталось неизменным во всех остальных случаях. Получим состояние:



Хотя вентиль Uf применен однажды для всех, он одновременно применен для всех n-кубитных состояний.

Трансформация Уэлша-Адамара применяется к первым n кубит. Получим:



Для наглядности, запишем применение одно-кубитного вентиля Адамара в следующем виде: //КАРТИНКА, где x*∈* {0,1}, для нахождения результата. Применение трансформации Уелша-Адамара на |x>=|xn-1…x1x0> даст: //КАРТИНКА где x\*y=xn-1yn-1 *⊕* xn-2yn-2 *⊕* … *⊕* x0y0 . Заменив результат по 5.6 получим//КАРТИНКА 5.8

Первые n кубит измерены. Допустим f(x) постоянная. Тогда //КАРТИНКА . В итоге получаем: //КАРТИНКА с фиксированным y*∈ Sn.* Это выражение сводится к нулю, при x\*y=0 для половины от всех х и 1 для другой половины х, если у не равен 0.Тогда сумма возвращает *δy*0. Теперь состояние сводится к ФОРМУЛА и измерение выходных первых n кубитов всегда 00…0. Теперь допустим что f(x) сбалансированная. Вероятность |y=0> в |фи3> пропорциональна ФОРМУЛА. Таким образом, пропадает вероятность в результате измерения получить 00…0 для первых n кубитов. В результате, функция f постоянная, если мы получили 00…0 при измерении первых n кубитов состояния |фи3>, и сбалансированная в другом случае.

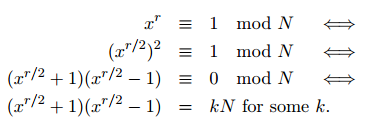
2.5. Алгоритм Шора

Возможно наиболее важный квантовый алгоритм на данный момент – это алгоритм факторизации Шора. Он способен найти делители составного числа N за грубо говоря (log N)2 шагов. С другой стороны, на сегодняшний день нет ни одного детерминистического или случайного алгоритма, который может факторизовать N за полиномиальное время. Лучший известный классический алгоритм по времени работает примерно 2(log N)α, где α = 1/3 для эвристической верхней границы и α = ½ для точной верхней границы. По факту, большинство современной криптографии основывается на предположении о том, что не существует классического алгоритма факторизации. Вся криптография, к примеру, RSA будет взломана, если будет реализован.

Крайне важное наблюдение Шора в том, что есть эффективный квантовый алгоритм для проблемы нахождения периода, и что это может быть сведено к факторизации. Допустим, мы хотим найти делители составного числа N > 1. Предположим, что N нечетное и не простое число, так как это может быть легко отсеяно классическим алгоритмом. Случайно выберем некоторое целое число x ∈ {2, . . . , N − 1}, которое взаимно простое с N. Если x не взаимно простое N, тогда наибольший общий делитель для x и N это ненулевой делитель числа N. Теперь будем считать x и N взаимно простыми, поэтому x – элемент мультипликативной группы ZN. Рассмотрим последовательность

1 = x0(mod N ), x1(mod N ), x2(mod N ), . . .

Последовательность зациклится через некоторое время: есть как минимум 0<r<=N, таких что xr = 1 (mod N). Назовем r периодом последовательности. Допустив что N нечетное и не простое, можем увидеть, что с вероятностью больше или равной 1/2, период r четный и xr/2 + 1 и xr/2 – 1 не множители N. В этом случае имеем:



Глава 3. Реализация задачи.

3.1.Средства реализации

3.2. Требования к программному и аппаратному обеспечению

3.3. Реализация