

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего образования

Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Квантовые вычисления

Выполнил:

студент группы 382003-1

Ивлев А.Д.

Научный руководитель:

Линёв А.В.

Консультировал:

Ведруков П. Е.

Нижний Новгород

2022

Содержание

1. Введение
2. Постановка задач
3. Кубит и однокубитные квантовые гейты
4. Система кубитов и многокубитные квантовые гейты
5. Квантовые схемы
6. Квантовое преобразование Фурье
7. Основные логические и арифметические алгоритмы
8. Арифметические алгоритмы по модулю
9. Алгоритм Шора
10. Реализация системы кубитов (квантового регистра) и основных гейтов. Общие принципы работы с системой.
11. Результаты экспериментов
12. Заключение
13. Приложение
14. Список литературы

Введение

Квантовый компьютер - вычислительное устройство, которое использует для вычислений различные свойства квантовых состояний, например суперпозицию или квантовую запутанность.

Современные квантовые компьютеры пока что достаточно малы и далеки от совершенства, чтобы превзойти классические компьютеры на практике, но из-за своих свойств они способны решать определенные вычислительные задачи (например, целочисленная факторизация, которая лежит в основе шифрования RSA) асимптотически быстрее, чем классические компьютеры, часто сводя экспоненциальную сложность алгоритмов к полиномиальной.

Разработка квантовых алгоритмов для таких задач ведётся параллельно разработкам по созданию и совершенствованию квантовых компьютеров. Но для проверки работоспособности данных алгоритмов пока недостаточно доступных квантовых компьютеров по нескольким причинам: малое количество кубитов (квантовых битов), топология их связей представляет собой не полный граф, достаточная большая погрешность при вычислениях (Пример топологии реального квантового компьютера IBM в приложении). Конечно, над решением этих проблем ведутся работы, например в современных моделях имеется некоторый максимально допустимый относительный размер ошибки, до которого погрешности корректируются.

Постановка задач

Модель идеального квантового компьютера представляет собой систему кубитов, где каждый из них связан со всеми другим и все гейты (квантовые вентили) выполняются без ошибок. Именно такая модель лучше всего подходит для первоначальной проверки квантовых алгоритмов. Второй этап тестирования, алгоритма представляет собой проверку его устойчивости к ошибкам, возникающим во время вычислений на реальном квантовом компьютере. На третьем этапе алгоритм нужно проверить на работоспособность при топологии связей кубитов приближенной к реальной.

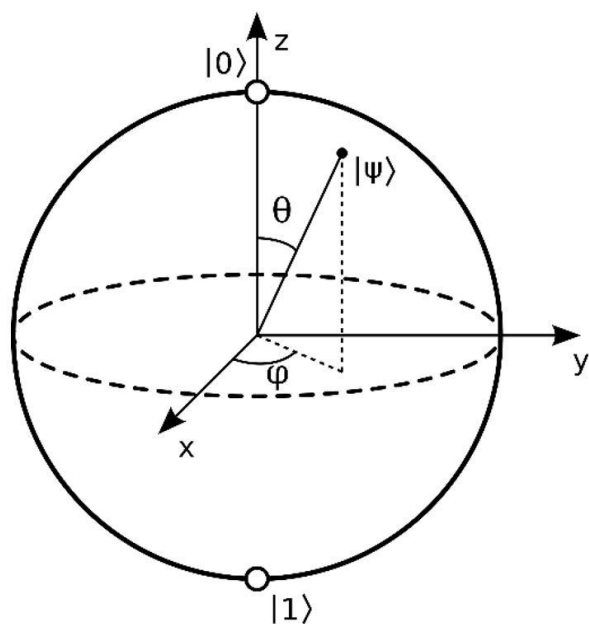
Задача состоит в создании на обычном компьютере модели квантового компьютера, на которой возможно составлять из базовых гейтов квантовые алгоритмы и проверять их работоспособность при наличии ошибок выполнения базовых гейтов. В данной работе за генерацию ошибок будет отвечать один из самых часто используемых гейтов, гейт фазы (gate P - гейт фазового сдвига), который будет описан ниже.

Также будут продемонстрированы и описаны некоторые квантовые алгоритмы, а классическое представление квантовой схемы алгоритма Шора будет протестировано на устойчивость к ошибкам.

Кубит и однокубитные квантовые гейты

Кубит - наименьшая единица измерения количества информации в квантовом компьютере. В отличие от классического бита может находиться в базисных состояниях $|0\rangle$ и $|1\rangle$, а также их суперпозиции $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$, где $\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1$, то есть состояние кубита описывается нормированным вектором $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$. Условие нормировки должно сохраняться при всех действиях с кубитом.

Также исходное состояние кубита может быть эквивалентным образом представлено с помощью двух вещественных параметров — углов φ и θ : $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle$. Таким образом можно отобразить состояние кубита на сфере Блоха.



https://en.wikipedia.org/wiki/File:Bloch_sphere.svg

Базовая операция над кубитом это его измерение, но невозможно измерить состояние кубита, не повлияв на него (особенность квантовой механики, не имеющая аналогов в классической физике). Измеряя кубит мы переводим его в одно из базисных состояний $|0\rangle$ или $|1\rangle$ с вероятностями $|\alpha|^2 = |\langle 0|\phi\rangle|^2$, $|\beta|^2 = |\langle 1|\phi\rangle|^2$ соответственно. Зная это можно заметить, что за вероятности при измерении состояния кубита отвечает только угол θ .

Любая логическая операция с кубитами называется гейтом (от английского gate – вентиль). Они (по аналогии с классическими логическими элементами) являются базовыми блоками для построения квантовых схем. По числу задействованных кубитов гейты делятся на однокубитные и многокубитные. Для демонстрации действия гейта на кубиты используют его матричную запись.

Применение произвольного однокубитного гейта u , который задаётся матрицей $u = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$, к произвольному кубиту $|\phi_0\rangle = \begin{pmatrix} \phi_{10} \\ \phi_{20} \end{pmatrix}$ можно описать $|\phi_1\rangle = u * |\phi_0\rangle$. При этом матрица, задающая гейт такая, что после его применения не нарушается условие нормировки. Далее кратко описаны основные однокубитные гейты.

Единичный гейт (Identity gate) I

Гейт I описывается единичной матрицей $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Легко заметить, что при применении данного гейта состояние кубита не меняется и его реализация не имеет смысла, но он важен при математическом описании квантовых вычислений.

Гейты Паули (Pauli gates) X (NOT), Y, Z

Данные гейты задаются унитарными матрицами $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$. Можно заметить, что с помощью гейтов Паули можно отобразить поворот состояния кубита на сфере Блоха вокруг осей X, Y, Z соответственно.

Также гейт X называю гейтом NOT, потому что после применения данного гейта меняются местами весовые коэффициенты состояний $|0\rangle$ и $|1\rangle$. $(\alpha |0\rangle + \beta |1\rangle \rightarrow \beta |0\rangle + \alpha |1\rangle)$

Ещё можно заметить, что Z не меняет вероятности $|0\rangle$ и $|1\rangle$. На сфере Блоха меняется только угол φ (перемещение вокруг оси Z в плоскости XY). Можно сказать, что гейт Z меняет фазу состояния кубита.

Для кубита можно построить неограниченное число гейтов. Однако, в силу полноты системы состоящей из матриц Паули и единичной матрицы I, любая матрица 2 на 2 может быть разложена на комбинацию этих матриц. Поэтому для использования представляют интерес сами матрицы Паули и

некоторые их специальные (часто использующиеся) комбинации, описанные ниже.

Гейт Адамара (Hadamard gate) H

Матрица данного гейта: $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ ($H = \frac{x+z}{\sqrt{2}}$). На сфере Блоха данный гейт поворачивает состояние на π вокруг оси $\frac{(x+z)}{\sqrt{2}}$.

Гейт фазового сдвига (Phase shift gate) P

Матрица: $P = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$ ($P\left(\frac{\pi}{\psi}\right) = \sqrt[\psi]{Z}$, $P(0) = I$). Данный гейт не изменяет вероятности $|0\rangle$ и $|1\rangle$, но изменяет состояние кубита (то есть меняет фазу). Данное изменение можно описать на сфере Блоха, как поворот на ϕ радиан в плоскости XY.

Система кубитов и многокубитные квантовые гейты

Набор из n кубитов составляет систему кубитов. Важно, что эта система также подчиняется принципу суперпозиции и находится одновременно сразу во всех своих базовых классических состояниях, число которых равно 2^n . Произвольное состояние системы записывается в виде:

$|\phi\rangle = \sum_{i=0}^{2^n-1} a_i |i\rangle$, $a_i \in \mathbb{C}$ (в правой части выражения i записывается в двоичном виде справа налево). Для системы также должно выполняться условие нормировки. Значит состояние системы можно описать нормированным вектором, состоящим из a_i .

Чтобы объединить состояния двух квантовых систем нужно взять их тензорное произведение $|\phi\psi\rangle = |\phi\rangle \otimes |\psi\rangle$. Но не всегда можно выделить состояния отдельных кубитов (групп кубитов) из системы обратив эту процедуру. Говорят, что кубиты сцеплены или запутаны (entangled), если их совместное состояние невозможно разложить в произведение индивидуальных состояний. Например, состояние Белла, которое можно задать с помощью гейтов H и CNOT (подробнее ниже).

Применение однокубитного гейта U к произвольному j -му кубиту в системе из n кубитов можно описать матрицей, полученной из тензорного произведения: $I \otimes \dots \otimes I \otimes U \otimes I \otimes \dots \otimes I$, где U стоит на j месте, а единичных матриц $n-1$. Также можно объединять применение нескольких однокубитных гейтов к разным кубитам в системе в одну

матрицу преобразования, например, для двух кубитной системы $X \otimes H$, где к 0 кубиту мы применили X, а к 1 H.

Основными многокубитными гейтами являются контролирующие (управляющие) гейты, которые применяют однокубитный гейт U к j-му кубиту, если все контролирующие кубиты находятся в состоянии $|1\rangle$. Например, для 2-х кубитной системы матрица преобразования, где 1 управляющий гейт, а 0 управляемый можно записать в виде:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{pmatrix}$$
, если же наоборот, то
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & u_{11} & 0 & u_{12} \\ 0 & 0 & 1 & 0 \\ 0 & u_{21} & 0 & u_{22} \end{pmatrix}$$
. В первом случае состояния $|00\rangle$ и $|01\rangle$ останутся без изменений, а к паре состояний $|10\rangle$ и $|11\rangle$ применится гейт U. Во втором $|00\rangle$ и $|10\rangle$ без изменений, к $|01\rangle$ и $|11\rangle$ применятся данный гейт. По аналогии строятся матрицы преобразований для систем большей размерности и большего количества контролирующих гейтов. Далее покажем основные гейты с контролем CNOT и CCNOT.

Гейт контролируемого отрицания CNOT (XOR)

Данный двухкубитный гейт применяет гейт X (NOT), если управляющий гейт находится в состоянии $|1\rangle$. Легко по описанным выше правилам построить матрицу данного преобразования, где 0 кубит

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

управляемый, а 1 управляющий:

Если внимательней посмотреть на результат применения данного гейта, то можно заметить, что он эквивалентен логической операции XOR над 2 кубитами, где результат записывается в управляемый кубит.

Гейт дважды контролируемого отрицания CCNOT (Toffoli gate)

Данный трёхкубитный гейт применяет гейт X (NOT), если управляющие гейты находятся в состоянии $|1\rangle$. По аналогии его матрица:

$$CCNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

В общем случае, гейт n раз контролируемого отрицания задаёт операцию $u = u \oplus (c_1 \wedge \dots \wedge c_n)$, где u - управляемый кубит, а c_i управляющие.

В теории дискретной математики доказано, что любую логическую операцию можно представить как совокупность некоторого числа стандартных, базовых операций, например системы AND, XOR и 1 (базис Жегалкина). В квантовых вычислениях аналогично можно любую обратимую унитарную операцию на кубитах можно представить как совокупность некоторых базовых операций. Таким базисом может служить один трехкубитный гейт (например CCNOT или CSWAP (гейт Фредкина)) или парв из одного однокубитного и одного двухкубитного гейта (например, X (NOT) и CNOT (XOR)). На практике обычно рассматривают последний вариант базиса: X и CNOT.

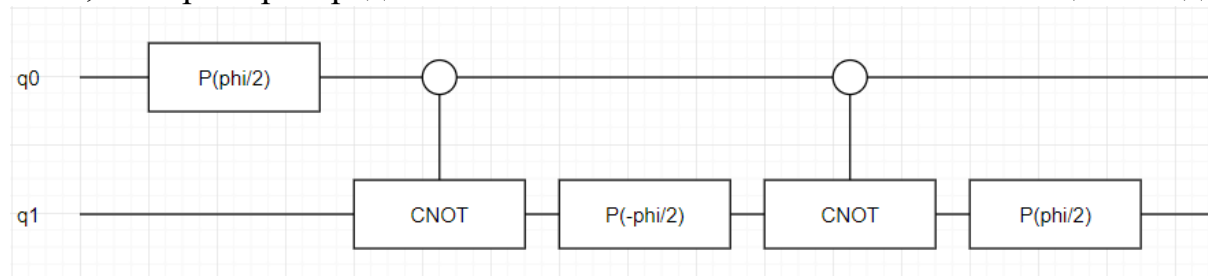
Учитывая всё выше сказанное описанных в данной работе гейтов более чем достаточно, чтобы составлять любые квантовые схемы. И большинство из них описаны только потому, что они часто используются при составлении алгоритмов.

Другие гейты, которые будут использованы ниже можно составить из комбинации данных выше гейтов, например гейт контролируемого

$$CP(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix}$$

фазового сдвига, где 0 кубит управляемый, а 1 управляющий можно представить, как последовательное применение гейтов $P(0, \frac{\phi}{2})$, CNOT (0, 1), $P(1, -\frac{\phi}{2})$, CNOT (0, 1), $P(1, \frac{\phi}{2})$. Данную запись можно считать простым квантовым алгоритмом.

Любые квантовые алгоритмы принято представлять в виде квантовых схем, например представление гейта CP на схеме в общем вид:






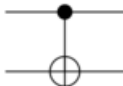


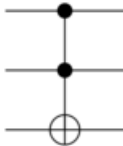


Подробнее про алгоритмы и квантовые схемы их изображающие будет рассказано далее.

Квантовые схемы

Квантовая схема - модель квантовых вычислений, где, аналогично классическим схемам, входные значения проходят через последовательность элементов (гейтов, измерений и т.д.) вычисляя определённую функцию.

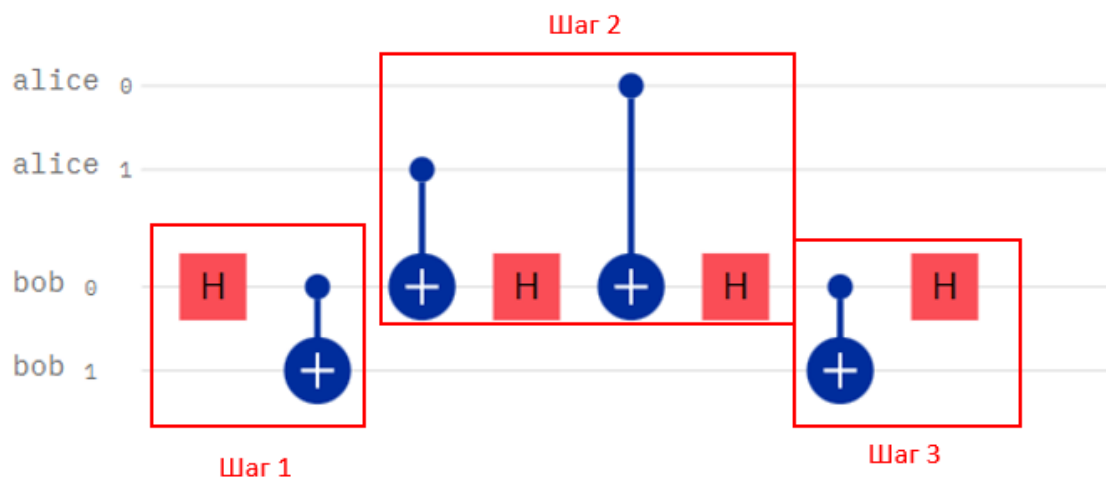
Обычно квантовые схемы изображаются, как строки, идущие вниз от младшего кубита к старшему, на которых последовательно в прямоугольниках или кругах изображаются гейты, применённые к данному кубиту. Многокубитные гейты затрагивают сразу несколько строк, а гейты с контролем указывают прямой с точками на линии соответствующих управляющих кубитов. Пример изображения гейтов:

Operator	Gate(s)	Matrix
Pauli-X (X)	 	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
SWAP	 	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

https://en.wikipedia.org/wiki/File:Quantum_Logic_Gates.png

Далее для примера рассмотрим квантовую схему, реализующую плотное кодирование (dense coding). Данный алгоритм, позволяет передать два классических бита информации используя один кубит в качестве канала связи (стандартное обозначение канала в криптографии: Alice – Bob). Данный способ передачи информации является безопасным, так как нельзя

извлечь информацию из одного кубита, не имея запутанного с ним кубита.
Схема алгоритма:



Так как данная работа направлена на описание квантовых вычислений и симуляцию квантового компьютера, и не включает в себя составление и оптимизацию квантовых алгоритмов, то все представленные стандартные алгоритмы будут рассмотрены с практической стороны, без доказательств. Подробнее про приведённые квантовые алгоритмы можно прочитать в литературе, указанной в конце доклада.

Шаг 1: создаём в кубитах Боба запутанное состояние Белла, в котором нельзя извлечь информацию из одного кубита, не имея запутанного с ним кубита.

Шаг 2: кубит `Bob0` передаётся Алисе, где он с помощью двух кубитов кодируется двумя битами.

Шаг 3: выход из состояния Белла, после чего мы можем измерить полученную от Алисы информацию.

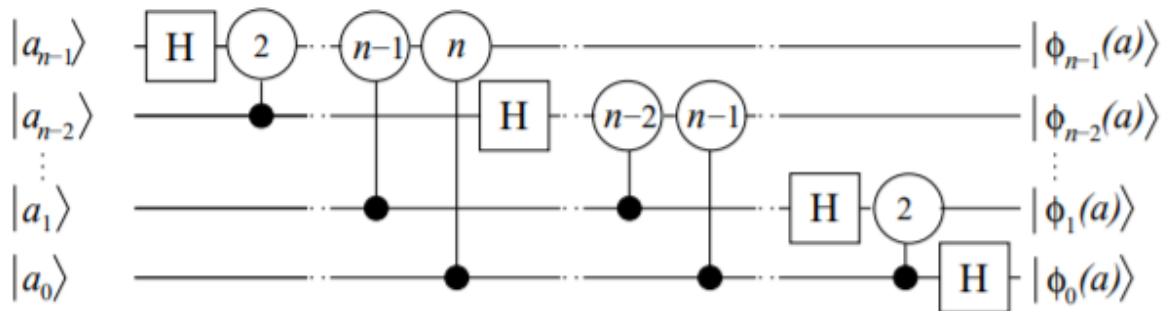
Стоит заметить, что квантовые схемы должны состояться таким образом, что кубиты в которые не записывается результат должны по окончании алгоритма иметь те же состояния, что и в начале. Также квантовая схема должна быть обратима, то есть если выполнить её с конца (если есть гейты фазы, то поворот на тоже значение, но в обратную сторону), то она вернёт изначальные состояния системы.

Так плотное кодирование удовлетворяет этим условиям. Состояния кубитов Алисы не изменились после работы алгоритма и если мы исполним “отзеркаленный” алгоритм, то вернём Бобу его изначальные состояния.

Квантовое преобразование Фурье (QFT)

Первым алгоритмом для рассмотрения взято квантовое преобразование Фурье (QFT), которое будет использоваться в большинстве описанных далее алгоритмах. Используемая квантовая схема реализует на квантовом компьютере преобразование Фурье, которое выполняется за полиномиальное время.

Если обозначить состояние n кубитов как $|X\rangle = \sum_{j=0}^{n-1} x_j * |j\rangle$, то конечное состояние можно описать как $|Y\rangle = \sum_{k=0}^{n-1} y_k * |k\rangle$, где $y_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \omega_n^{jk}$ ($\omega_n^{jk} = e^{2\pi \frac{jk}{n}}$). То есть преобразование и его матрица аналогичны дискретному преобразованию Фурье. Для реализации данной схемы нам понадобятся два гейта: H и CP. На схеме обозначим CP ($\frac{\pi}{2^{k-1}}$) за кружок с числом k внутри.

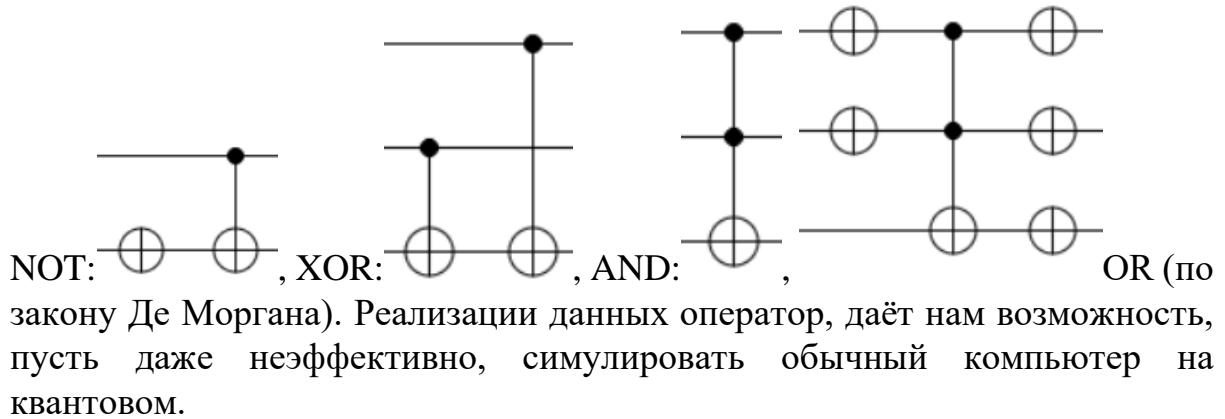


Данная схема реализует QFT и не трудно заметить, что её сложность $\frac{1}{2}n(n+1)$. После применения QFT мы переводим систему в пространство Фурье, для выхода из которого необходимо совершить обратное, "отзеркаленное" QFT (вспомним про обратимость квантовых схем), где нужно совершить обратные повороты, то есть CP ($-\frac{\pi}{2^{k-1}}$).

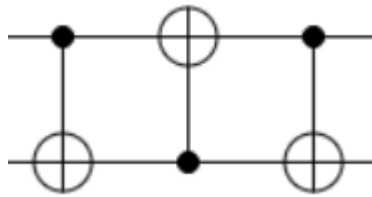
Так как с увеличением n , уменьшается минимально возможный угол $\frac{\pi}{2^{k-1}}$, то на реальном квантовом компьютере с некоторого достаточно большого k , CP ($\frac{\pi}{2^{k-1}}$) даёт настолько ощутимую ошибку, что, отбрасывая повороты на достаточно малые углы, мы уменьшим ошибку и уменьшим сложность алгоритма приблизительно к $O(n \log(n))$. Такой подход называется приближенный (Approximate) QFT. Подробнее про это можно узнать в книге A. Barenco, A. Ekert, K. Suominen, P. Torma, "Approximate quantum Fourier transform and decoherence"

Основные логические и арифметические алгоритмы

Как было сказано выше гейты $X(r)$, $CNOT(c, r)$, $CCNOT(c1, c2, r)$ реализуют функции \bar{r} (NOT), $r \oplus c$ (XOR), $r \oplus (c1 \wedge c2)$, где результат записывается в управляемый кубит r . Реализуем с помощью данных гейтов стандартные логические операторы (NOT, AND, OR, XOR), где результат будет записываться в отдельный кубит l , который по умолчанию находится в состоянии $|0\rangle$. Схемы, на которых l старший кубит:



Также представим схему двухкубитного гейта SWAP, который, как можно понять из названия, меняет между собой состояния двух кубитов:



. Работоспособность данных схем можно доказать, построив их таблицы истинности.

Рассмотрим схему квантового сумматора. Для реализации данной схемы нам понадобится гейт CP и $2n$ кубитов, которые мы условно разделим на 2 регистра $|a\rangle$ и $|b\rangle$ длиной n каждый. Регистр $|b\rangle$ перед началом алгоритма находится в пространстве Фурье. В данных регистрах лежат числа a и b соответственно, такие что $0 \leq a, b < 2^n$. Результатом выполнения схемы будет перевод регистров $|a\rangle \rightarrow |a\rangle, |b\rangle \rightarrow |(a + b) \bmod 2^n\rangle$. Регистр $|b\rangle$ после выполнения схемы всё ещё в пространстве Фурье и для измерений результата нужно сначала совершить обратное преобразование Фурье.

Схема

сумматора:

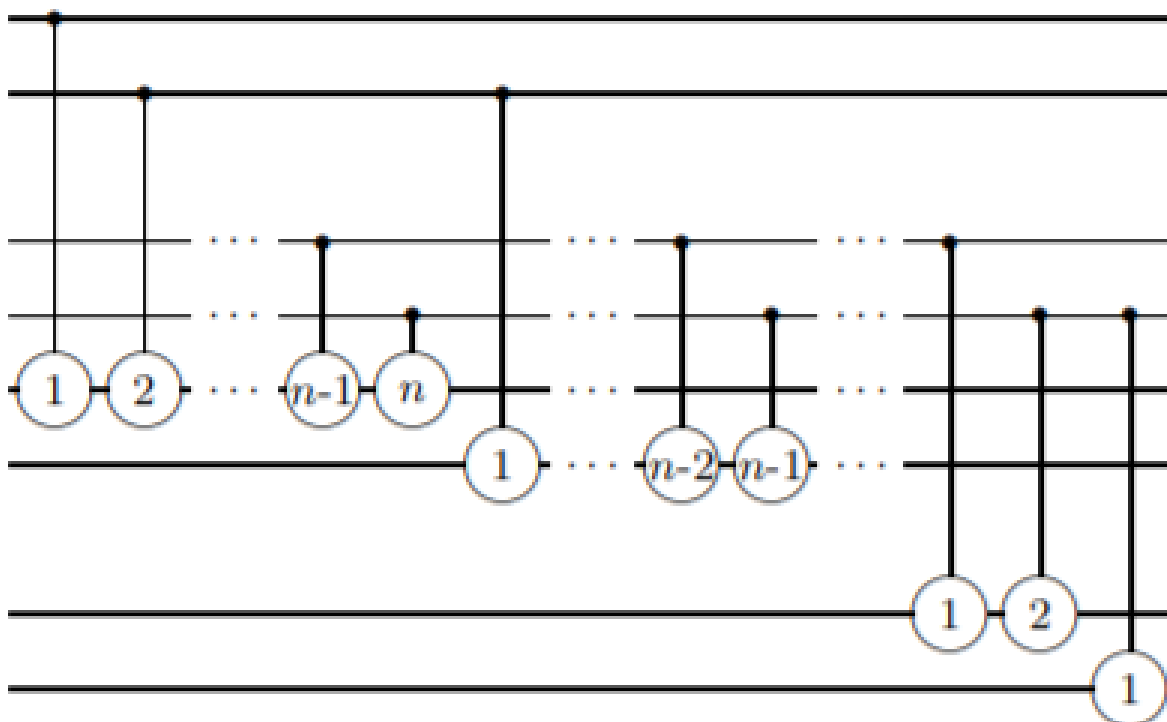
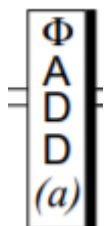
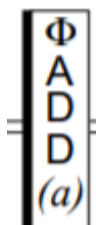


Схема взята из Addition on a Quantum Computer, Thomas G. Draper

Для использования в описанных далее схемах обозначим сумматор,



как $\frac{1}{2}n(n+1)$ гейтов СР. Так как все квантовые схемы обратимы, то обратная схема к данной будет являться вычитателем, где в $|b\rangle$ будет записано $|b-a\rangle$, если $b \geq a$, а если $b < a$, то $|2^{n+1} - (a-b)\rangle$. Все обратные алгоритмы будем обозначать чертой слева (или степенью -1),



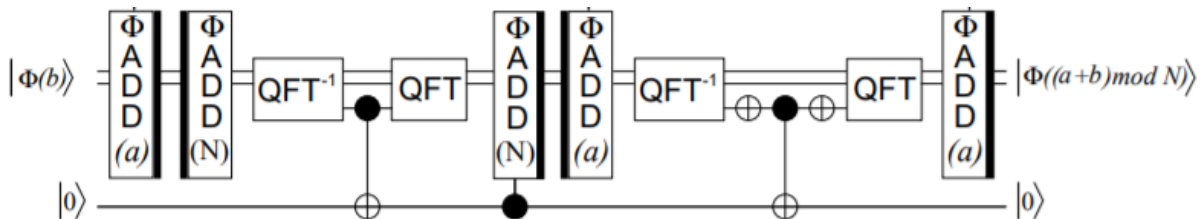
следовательно вычитатель:

Также можно модифицировать данную выше схему заменив квантовый регистр $|a\rangle$ на обычный, тогда нам будет необходимо совершать гейты Р только когда соответствующий бит числа a равен 1. Это позволит оптимизировать алгоритм по памяти и количеству исполняемых гейтов.

Арифметические алгоритмы по модулю

Сложение по модулю $(a+b) \bmod N$

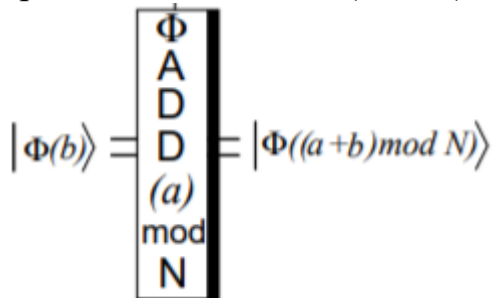
Рассмотрим по шагам схему сложения по модулю N . Всего для работы схемы необходимо $n + 2$ кубитов, где 2 кубита служат для отслеживания переполнения и изначально находятся в состоянии $|0\rangle$. Где a задаётся с классического регистра, а b находится в квантовом регистре длиной $n+1$, который находится в пространстве Фурье, причём $0 \leq a, b < N < 2^n$.



Эта и все последующие схемы взяты из Circuit for Shor's algorithm using $2n+3$ qubits, Stéphane Beauregard, 2003

1. На первом шаге мы складываем a и b .
2. Далее вычитаем N и получаем, $(a + b - N) \bmod 2^{n+1}$
 - a. Если $(a + b - N) \geq 0$, тогда n кубит будет в состоянии $|0\rangle$
 - b. Если $(a + b - N) < 0$, то в состоянии $|1\rangle$.
3. На третьем шаге нам нужно отследить в $n+1$ кубит состояние n кубита. Для этого сначала нам необходимо вывести систему из пространства Фурье и только затем применять CNOT. После вернём состояние системы сделав QFT.
4. Если $n+1$ кубит в состоянии $|1\rangle$ (произошло переполнение из-за того, что $(a + b - N) < 0$), то нам необходимо добавить N . То есть:
 - a. Если $n+1$ кубит в состоянии $|0\rangle$, то ничего не изменится и в первых n кубитах лежит $(a + b - N)$
 - b. Если $n+1$ кубит в состоянии $|1\rangle$, то первых n кубитах лежит $(a + b)$
5. Вычтем a и получим:
 - a. $(b - N) < 0$, тогда в n кубит будет в $|1\rangle$
 - b. $(b) > 0$, тогда n кубит $|0\rangle$
6. Дополнительный $n+1$ кубит после выполнения алгоритма должен находиться в состоянии $|0\rangle$. Тогда если n кубит в состоянии $|0\rangle$, то мы знаем, что $n+1$ кубит в состоянии $|1\rangle$ и мы, выходя из пространства Фурье, меняем его с помощью двух отрицаний (второе чтобы оставить n кубит в состоянии $|0\rangle$) и CNOT. После снова делаем QFT
7. Прибавляем a и получаем:
 - a. $(a + b - N)$, если $(a + b) \geq N$
 - b. $(a + b)$, если $(a + b) < N$

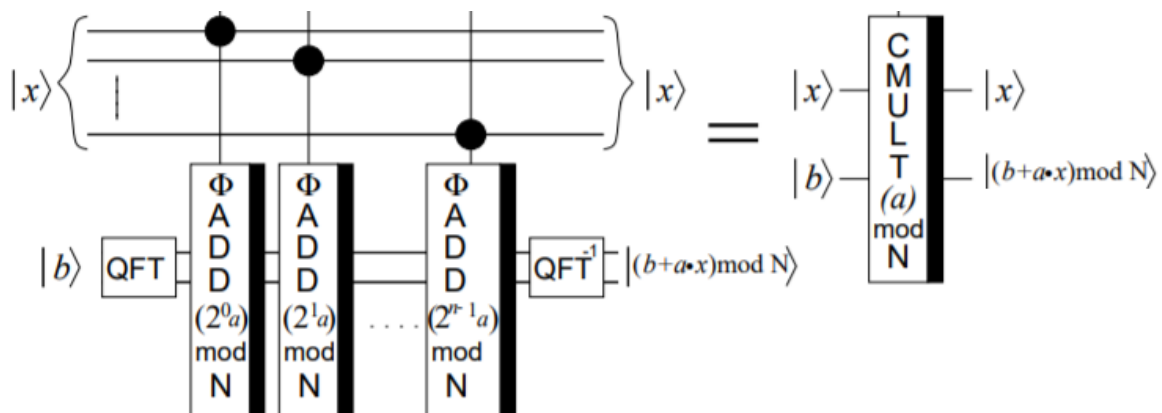
Что соответствует $(a+b) \bmod N$ (в пространстве Фурье). Вычитание $(b - a) \bmod N$, можно получить аналогично обратив данную схему или провести сложение $(b + (N - a)) \bmod N$. Обозначим сложение как:



. В худшем случае сложность (глубина) алгоритма $\frac{9}{2}n(n+1) + 4$ исполняемых гейтов.

Умножение по модулю $(b+ax) \bmod N$

Дальше рассмотрим алгоритм умножения по модулю N . Заметим, что ax можно представить в виде $ax = a(x_0 2^0 + \dots + x_{n-1} 2^{n-1})$, то есть разложением x в двоичное представление, сведём умножение к n контролируемым сложениям. Тогда при $0 \leq a, x, b < N < 2^n$ нам понадобится $2n+2$ кубитов (n для представления x и $n+2$ для сложений, где изначально может храниться некоторое b) и схема с применением QFT и обратного QFT примет вид:

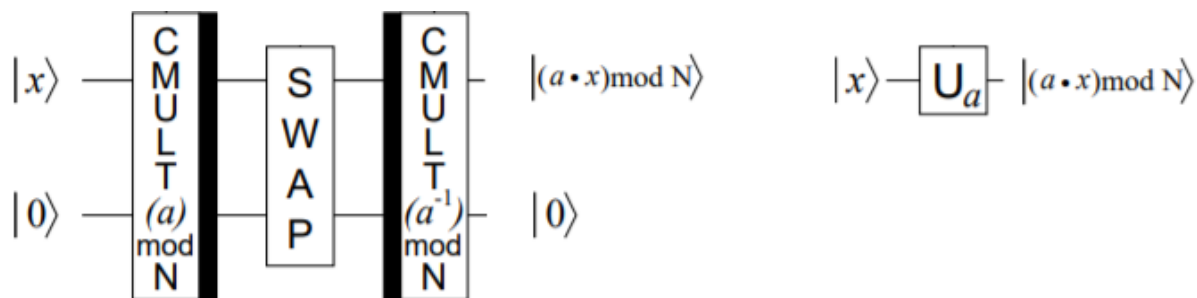


Сложность (глубина) алгоритма в худшем случае (без учёта QFT): $\frac{9}{2}n^2(n+1) + 4n$.

Обратная к данной схема даст результат $(b - ax) \bmod N$.

Умножение по модулю $(b+ax) \bmod N$ с записью в входной регистр

Введём в рассмотрение алгоритм умножения по модулю, который сохраняет результат в регистр, где находился x , что позволит нам выполнять умножения друг за другом:



Во-первых, регистр для сложения изначально должен быть в состоянии $|0\rangle$.

1. После выполнения первого умножения в первом регистре останется x , а во втором запишется результат ax .
2. Меняем значения регистров с помощью гейтов SWAP.
3. Затем нам нужно вернуть второй регистр из состояния $|x\rangle$ в состояние $|0\rangle$. Для этого можно сделать обратное умножение на число a^{-1} обратное по модулю N к a . После чего получим $(x - aa^{-1}x) \bmod N$, то есть 0. Но стоит учесть, что мы можем найти обратное по модулю число, только если $\text{НОД}(a, N) = 1$.

Сложность (глубина) данного алгоритма без учёта двух внешних QFT в CMULT в худшем случае равна $9n^3 + 10n^2 + 12n$.

Алгоритм Шора

Прежде чем переходить к разбору квантовой части алгоритма Шора углубимся в классическую его часть.

Нетривиальным квадратным корнем от N назовём число x такое, что $x^2 \equiv 1 \bmod N$, но $x \not\equiv \pm 1 \bmod N$. Для такого числа известно, что $\text{НОД}(x + 1, N)$ является делителем числа N отличным от 1 и N .

Порядком числа x по модулю N назовём наименьшее r такое, что $x^r \equiv 1 \bmod N$. Задача отыскания порядка случайного остатка по модулю N , как мы увидим, связана с отысканием нетривиального квадратного корня из N (и тем самым с разложением на множители). Это видно из следующей леммы:

Пусть N — нечётное составное число, имеющее как минимум два разных простых множителя. Выберем случайное число a от 0 до $N - 1$, взаимно простое с N , считая все такие числа равновероятными. Тогда с вероятностью $1/2$ или больше порядок r числа a по модулю N окажется чётным, а число $x^{\frac{r}{2}}$ будет нетривиальным квадратным корнем из N .

Квантовая часть алгоритма Шора отвечает за нахождение порядка r для заданной пары (x, N) . Для этого используется периодичность функции

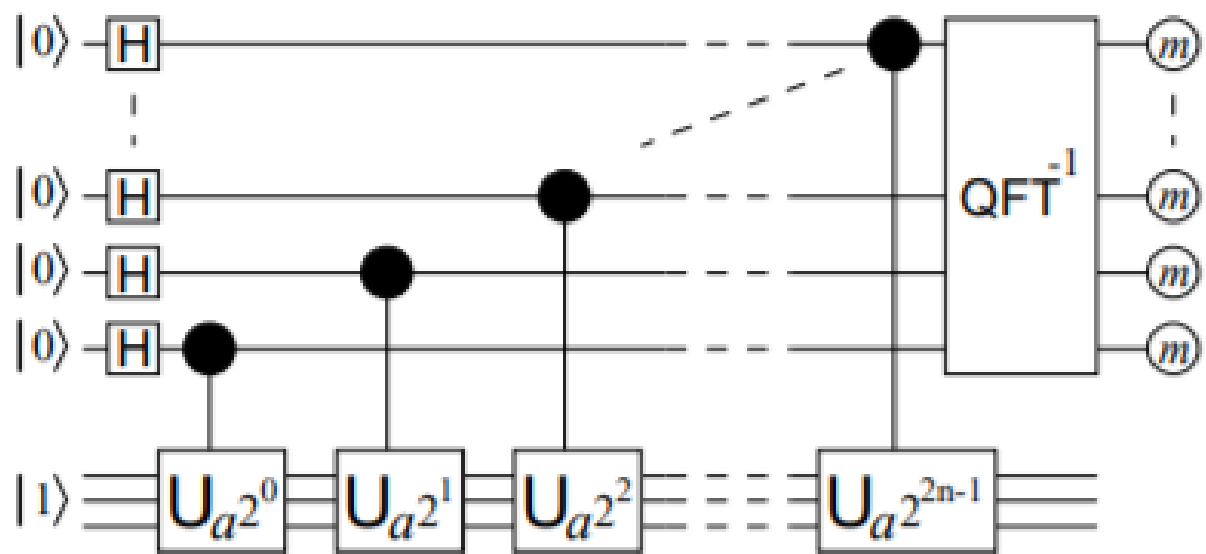
$f(a) = x^a \bmod N$ для получения суперпозиции с периодическими коэффициентами.

Возьмём два квантовых регистра длинами $2n$ и $2n+2$. В которых изначально все кубиты, кроме $2n$ -ого находятся в нулевом состоянии. Применим преобразование Адамара к каждому кубиту первого регистра. В результате в нём появится суперпозиция всех значений.

Применяем квантовую схему, соответствующую классическому вычислению функции $f(a) = x^a \bmod N$. В результате получаем состояние двух регистров, равное $\sum_{a=0}^{2n-1} \frac{1}{\sqrt{2n}} |a, f(a)\rangle$.

Раз функция f имеет период r , то (поскольку в последовательности $f(0), f(1), f(2), \dots$ повторения случаются только через r шагов) в первом регистре останется периодическая последовательность, и квантовая выборка из преобразования Фурье и измерение первого регистра позволит найти период r , что нам и требовалось.

Если представить $x^a = x^{(a_0 2^0 + \dots + a_{2n-1} 2^{2n-1})}$, то квантовый алгоритм вычисления $x^a \bmod N$ можно свести к последовательности из $2n$ умножений. Тогда квантовая схема алгоритма Шора примет вид:



После алгоритма Шора от (a, N) ($\text{НОД}(a, N) = 1$) мы знаем период r и для нахождения делителя числа N необходимо вычислить $\text{НОД}\left(a^{\frac{r}{2}} + 1, N\right)$.

Итоговая сложность (глубина) алгоритма Шора $18n^4 + 20n^3 + 26n^2 + 3n$.

Реализация системы кубитов (квантового регистра) и основных гейтов. Общие принципы работы с системой.

Для моделирования системы кубитов нам необходимо хранить информацию о всех состояниях системы, то есть основу реализации будет составлять $\text{vector} \langle \text{complex} \langle T \rangle \rangle$ длины 2^n , где T - тип с плавающей точкой, а n - число кубитов в системе. В данной реализации в i элементе вектора должен храниться весовой коэффициент состояния $|i\rangle$, где i представляется в двоичном виде, записанном слева направо, например для системы из 3-х кубитов 3-ий элемент хранит информацию о $|110\rangle$ (в теории запись идёт справа налево), то есть младший кубит хранит информацию о младшем бите представления числа i .

Данный метод хранения приводит к экспоненциальному росту памяти, а также, применяя гейты (изменяя состояния системы), мы должны изменить состояния экспоненциально большого числа коэффициентов в векторе, что приводит к экспоненциальному росту числа операций, то есть нельзя моделировать квантовую вычислительную систему за полиномиальное время. Это не позволяет нам моделировать большие системы кубитов на домашних компьютерах, но для этого мы можем использовать суперкомпьютеры. Пока не существует известного способа эффективного моделирования квантового компьютера с помощью классического компьютера.

В данной реализации при составлении квантовых алгоритмов подразумевается, что после задания начального состояния квантового регистра мы работаем с ним, только посредством гейтов, то есть перед работой с вектором пользователь должен, либо задать его весовые коэффициенты так, чтобы вектор был нормализован, либо после выставления коэффициентов вызвать для системы функцию `normalization`, которая нормализует вектор за него.

После задания начального состояния система готова, для применения к ней гейтов. Если мы вернёмся к теории, то для применения однокубитного гейта U к j кубиту нам необходимо умножить вектор состояний на матрицу преобразования, полученной из тензорного произведения: $I \otimes \dots \otimes I \otimes U \otimes I \otimes \dots \otimes I$, но если мы проделаем данные вычисления несколько раз, то заметим, что данное преобразование сводится к применению гейта U ко всем парам состояний, где отличается только состояние j -го кубита. Например, для системы из 3-х кубитов применение

гейта U к 1 кубиту сведется к применению его к 4 парам состояний: $\begin{pmatrix} |000\rangle \\ |010\rangle \end{pmatrix}$

, $\begin{pmatrix} |100\rangle \\ |110\rangle \end{pmatrix}$, $\begin{pmatrix} |001\rangle \\ |011\rangle \end{pmatrix}$, $\begin{pmatrix} |101\rangle \\ |111\rangle \end{pmatrix}$. То есть нам нет необходимости проводить тензорные произведения и составлять матрицу преобразования достаточно,

только находить данные пары и напрямую применять к ним данный гейт. Поскольку данные пары не пересекаются это позволяет нам провести параллизацию применения однокубитных гейтов. Для параллизации вычислений была выбрана библиотека openMP. Так, например, выглядит реализация гейта X (NOT):

```
void X(size_t n) // (NOT)
{
    int nbit = (1 << n);

#pragma omp parallel for
    for (int i = 0; i < data.size(); i++)
    {
        if (!(i & nbit))
        {
            swap(data[i], data[i + nbit]);
        }
    }
}
```

Для применения произвольного n-кубитного гейта аналогично нужно искать группы из 2^n состояний, где отличаются, только состояния кубитов к которым данный гейт применён. Данные группы не пересекаются, что снова даёт возможность для параллельных вычислений. Для контролируемых гейтов достаточно искать пары состояний, где управляющие кубиты в состоянии $|1\rangle$ и отличается, только управляемый гейт и к данным парам непосредственно применять контролируемый гейт. Например, для применения в трёх кубитной системе гейта CU (0 (управляющий), 1 (управляемый)), сведётся к применению гейта U к парам: $\begin{pmatrix} |100\rangle \\ |110\rangle \end{pmatrix}$ и $\begin{pmatrix} |101\rangle \\ |111\rangle \end{pmatrix}$. Таким образом, реализация гейта CNOT:

```
void CNOT(size_t h, size_t l) // (XOR) h - control
{
    int nbith = (1 << h);
    int nbitl = (1 << l);

#pragma omp parallel for
    for (int i = 0; i < data.size(); i++)
    {
        if ((i & nbith) && !(i & nbitl))
        {
            swap(data[i], data[i + nbitl]);
        }
    }
}
```

После всех вычислений пользователь должен сделать серию измерений для получения результата (функция `condition_exp`). Если измеряются состояния n кубитов для получения относительно точного результата необходимо провести в несколько раз больше, чем 2^n экспериментов.

Результаты экспериментов

Проверка работоспособности алгоритма Шора

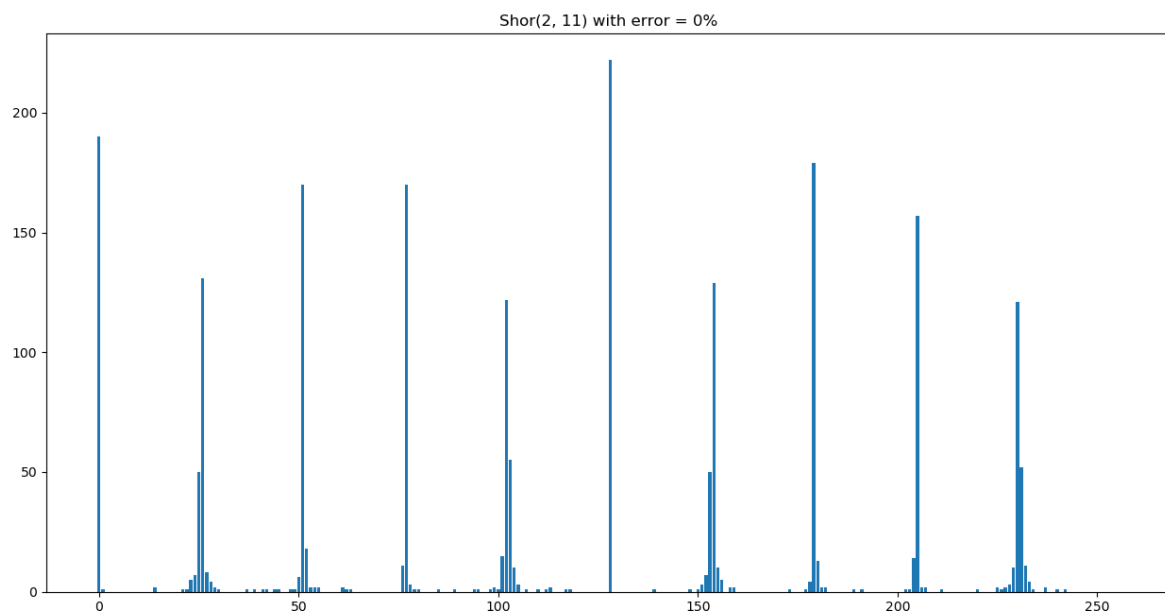
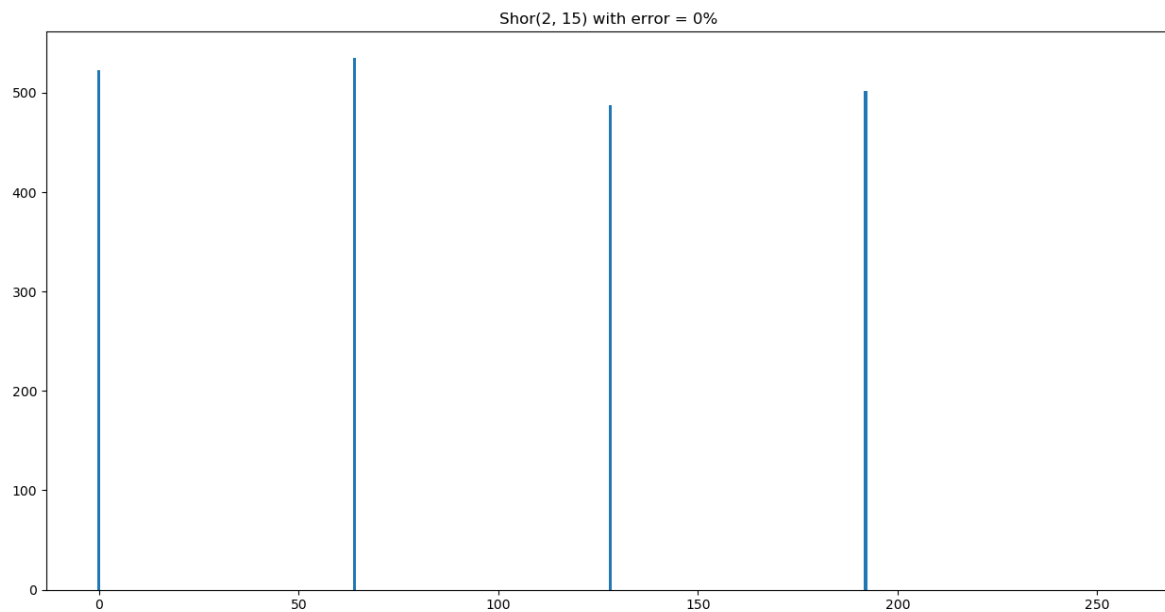
Был реализован алгоритм Шора. Для его исполнения должна быть система кубитов `qb` длиной, как минимум $4n+2$, где есть $4n+2$ стоящих подряд кубита в $|0\rangle$ состоянии. Запуск алгоритма происходит, как `qb.Shor(a, N, start, end, error)`, где a, N - целые положительные числа такие, что $0 \leq a < N < 2^n$, `start` и `end` обозначают область запуска на системе, то есть `end - start = 4n + 2`, `error` ошибка гейта `P` в % (по умолчанию ноль).

Для проверки работоспособности алгоритма проведём серию экспериментов, где `error = 0`, n минимально возможное для, конкретного N , размер системы $4n+2$. После применения алгоритма Шора, из результатов измерения нам необходимо выделить количество пиковых значений r (одно из которых всегда в 0). Будем считать, что алгоритм отработал правильно, если $\text{НОД}(a^r - 1, N) = N$.

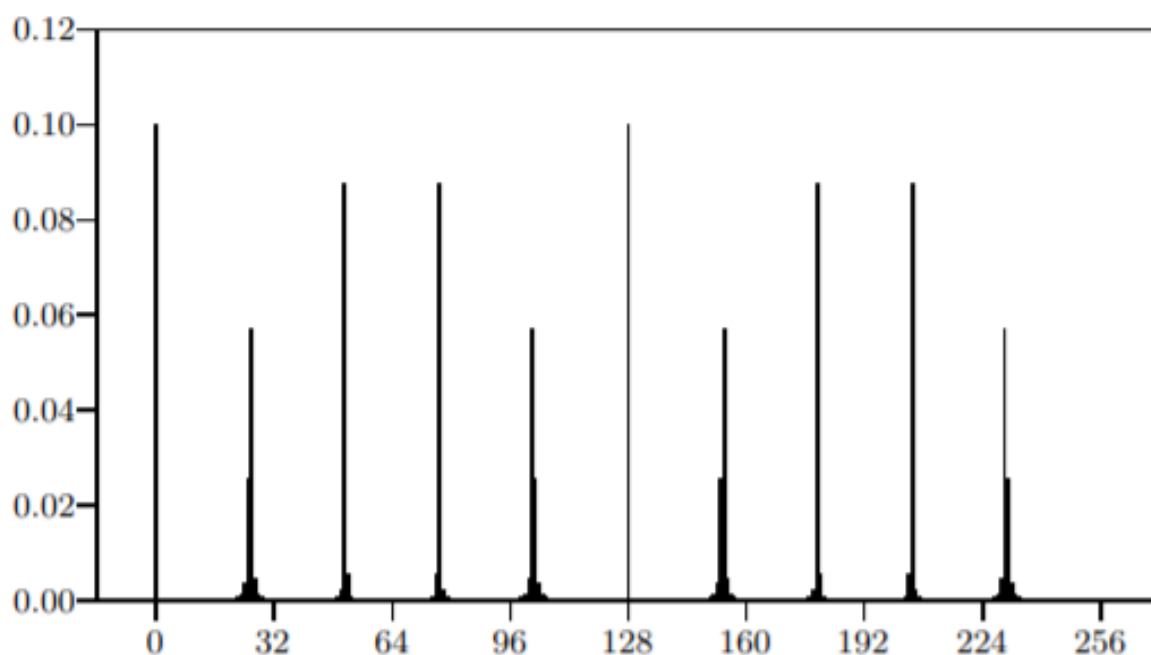
Пара $\{a, N\}$	Полученное r	$\text{НОД}(a^r, N)$
$\{2, 3\}$	2	3
$\{3, 7\}$	6	7
$\{2, 11\}$	10	11
$\{2, 15\}$	4	15
$\{3, 28\}$	6	28
$\{2, 63\}$	6	63

Записи и графическое представление всех результатов можно найти в папке `results`.

Графическое представление результатов:



Как видно алгоритм работает верно. Также мне удалось найти в работе Шора визуализацию теоретически полученного им результата, которая крайне похожа на результат, полученный мной.



Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, Peter W. Shor, 1996, page 18

Зависимость времени исполнения алгоритма Шора от длины системы

Моделирование квантовых вычислений на классическом компьютере имеет экспоненциальный рост времени выполнения алгоритмов от длины системы, на которой они исполняются. Покажем это на примере алгоритма Шора.

Аналогично прошлому эксперименту проведём серию запусков, где $\text{error} = 0$, n минимально возможное для, конкретного N , размер системы $4n+2$. Также включено распараллеливание с помощью openMP.

n	Среднее время (в секундах)
2	0.007
3	0.11
4	3.5
5	90
6	3300

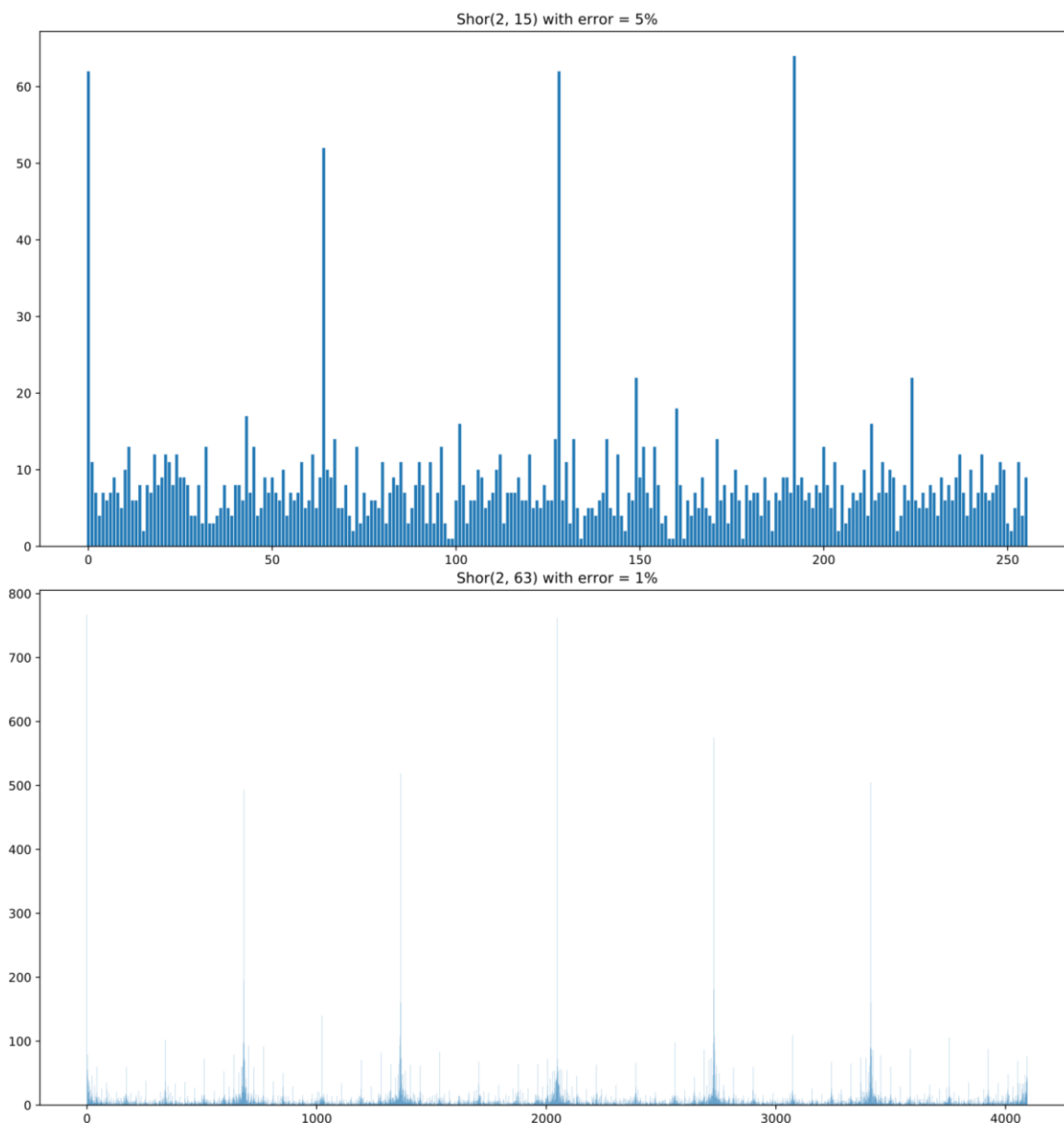
Аппроксимация данных значений дала примерно: $t = \frac{1}{4}e^{(3.5n - 11)}$, что подтверждает экспоненциальную сложность вычислений.

Параметры системы, на которой проводились эксперименты:

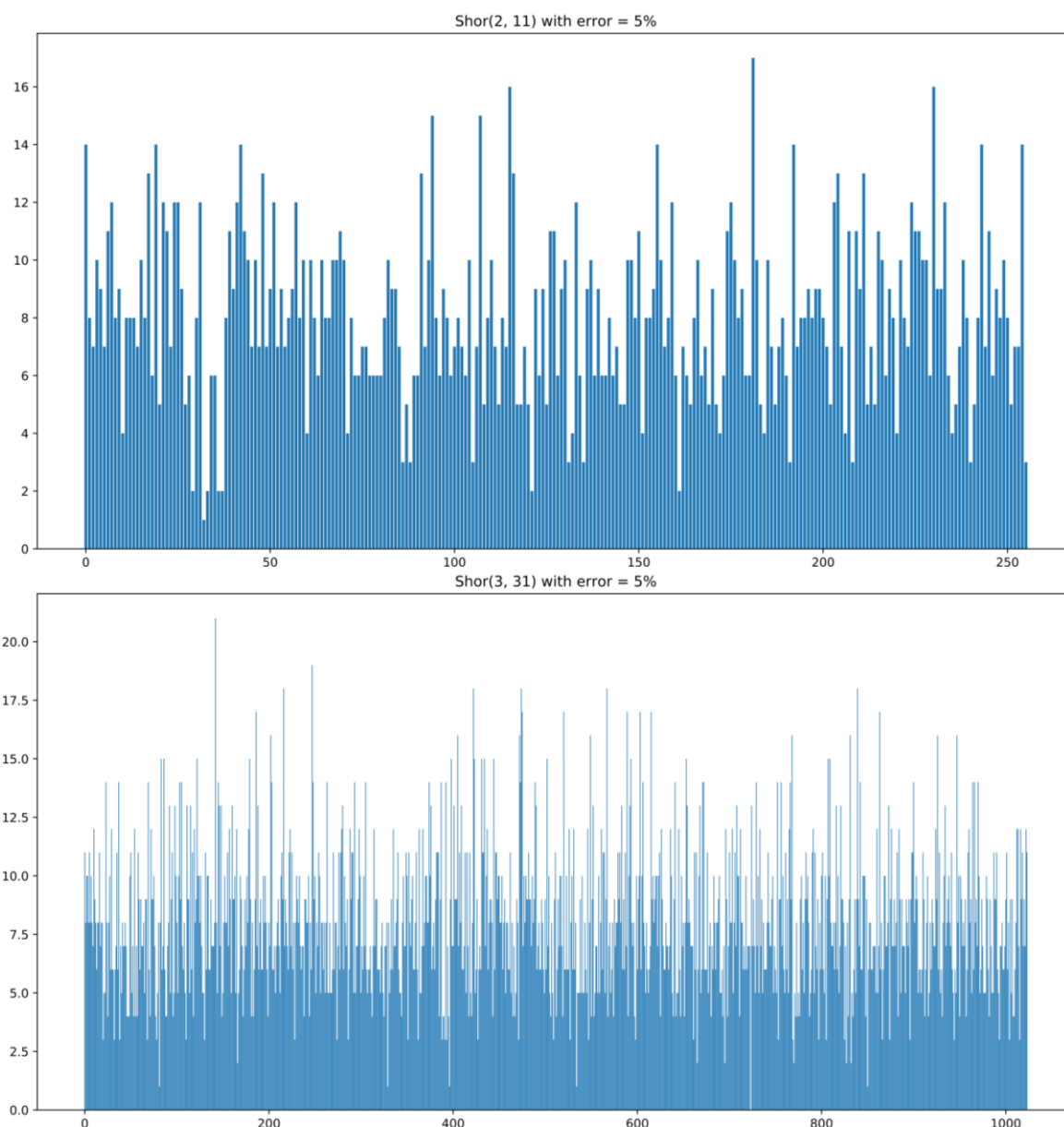
- CPU: AMD Ryzen 5 Mobile 3550H
- RAM: 16 GB, DDR 4, 1330 MHz
- ОС: Windows 10
- Compiler: Visual Studio 2019

Проверка алгоритма Шора на устойчивость к ошибке гейта Р

В данных экспериментах мы введём в гейт Р ошибку в % от угла поворота. Например, для $\text{Shor}(2, 15)$ на 18 кубитах ($n = 4$) с ошибкой в 5% и $\text{Shor}(2, 63)$ на 26 кубитах ($n = 6$) с ошибкой в 1% результат ещё различим:



А для $\text{Shor}(2, 11)$ на 18 кубитах ($n = 4$) с ошибкой в 5% и $\text{Shor}(3, 31)$ на 22 кубитах ($n = 5$) с ошибкой в 5% результат уже нельзя различить:

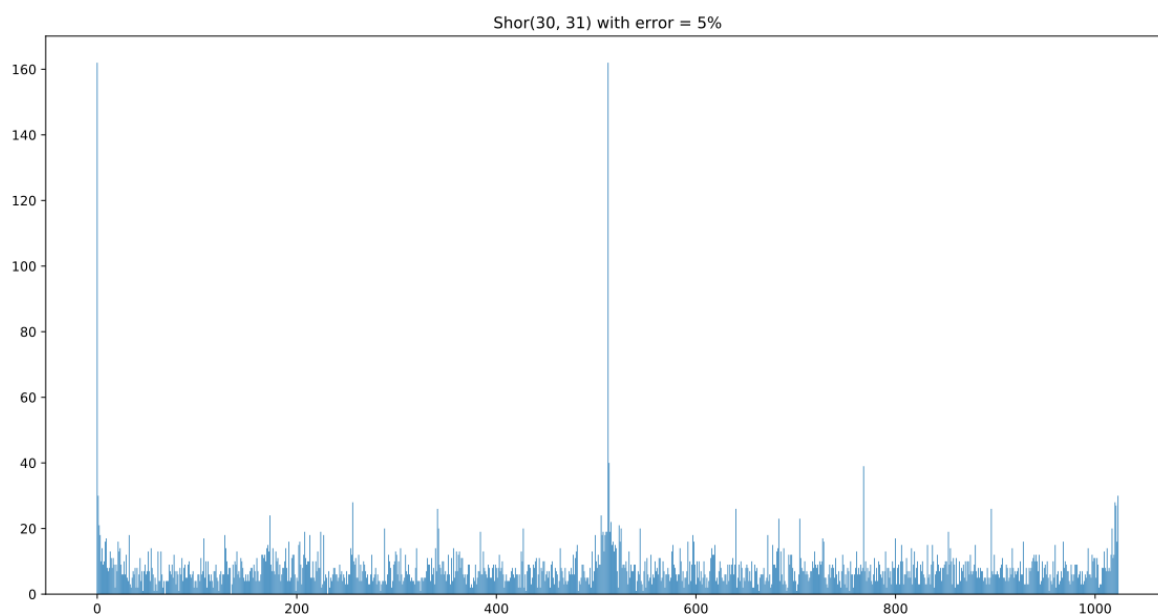


В ходе экспериментов удалось выяснить, что для увеличения точности необходимо проводить в несколько раз больше измерений, чем 2^{2n} . Также при ошибке меньше 1% результат различим во всех проведённых экспериментах.

На устойчивость результата к ошибкам влияет, количество кубитов в системе, чем больше кубитов, тем больше накапливается ошибка. Также влияют входные данные: большей устойчивостью обладают пары (a, N) , такие что результат вычисления алгоритма Шора на идеальной модели даст степень двойки (например, 2, 15, где результат 4), а худшей по устойчивости парой (a, N) является такая, что в результате на идеальной модели получим результат приближенный к n , отличный от степени 2 (например, 2, 11, где результат 10 или 3, 31, где результат 30).

Значит, чтобы получить наиболее корректный результат для конкретного n при достаточно большом уровне ошибки мы можем провести эксперименты с различными a на модели с минимально возможным количеством кубитов и выбрать из этих экспериментов один с наиболее различимым результатом.

Например, для $N = 31$, можно выбрать $a = 30$ и при тех же 5% ошибки получить различимый результат:



Часть экспериментов можно посмотреть в репозитории с кодом (ссылка в приложении). Также вы можете провести свои эксперименты.

Взлом RSA с помощью алгоритма Шора

RSA - криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

В RSA открытый ключ $\{e, n\}$ и закрытый ключ $\{d, n\}$ генерируются следующим образом:

1. Выбираются 2 случайных различных простых числа p и q (размер p и q задан)
2. Вычисляется $n = pq$ (модуль)
3. Вычисляется значение функции Эйлера от n : $\varphi(n) = (p - 1) \cdot (q - 1)$
4. Выбирается открытая экспонента e такая, что $1 < e < \varphi(n)$ и $\text{НОД}(e, \varphi(n)) = 1$
5. Вычисляется закрытая экспонента d такая, что $d \cdot e \equiv 1 \pmod{\varphi(n)}$

Чтобы закодировать сообщение m необходимо вычислить $c = m^e \bmod n$. А чтобы его расшифровать: $m = c^d \bmod n$.

Взлом происходит разложением n на p и q , для этого с помощью алгоритма Шора от (a, n) ($\text{НОД}(a, n) = 1$) находим период r , после мы можем найти делитель числа n посчитав $\text{НОД}(a^{\frac{r}{2}} + 1, n)$. Далее вычисляем $\varphi(n)$ и d , после чего можно произвести декодирование сообщения.

Цель эксперимента, зная открытый ключ $\{e, n\}$ и зашифрованное сообщение c , расшифровать закодированное в c сообщение m . В данном эксперименте $m = 3$. Далее будут представлены результаты с учётом 1% ошибки алгоритма Шора.

Открытый ключ $\{e, n\}$	Полученное после взлома сообщение	Время (в секундах)
$\{3, 10\}$	3	3.13
$\{7, 15\}$	3	3.83
$\{5, 26\}$	3	105

Как видно из экспериментов алгоритм Шора позволяет взламывать RSA. Вы также можете сами провести данные эксперименты всё для этого подготовлено в `nQbit.cpp`, достаточно только задать e в переменную e , n в переменную pq (e и n должны удовлетворять всем условиям открытого ключа) и длину двоичного представления n в переменную $n1$. Также можете изменить кодируемое сообщение m .

Заключение

В данной программной реализации симулятора квантового компьютера мною были явно реализованы гейты: H , X , $CNOT$, $CCNOT$, Y , Z , R , CP , CPP (гейты R , CP , CPP обладают возможностью генерации погрешности). Также, алгоритмически реализованы гейты $SWAP$, $CSWAP$ (гейт Фредкина). Проведено базовое тестирование. На примере реализации описанных выше квантовых гейтов показана универсальность системы для составления и тестирования алгоритмов. Результатом реализованных схем стал алгоритм Шора, который был протестирован на устойчивость к ошибкам (результаты прилагаются). Приведены примеры работы алгоритмов.

Также реализованы RSA, алгоритм проверки числа на простоту Рабина-Миллера модифицированный для факторизации чисел и ряд математических функций необходимых для их работы.

Следующими шагами в моей дальнейшей работе могут быть:

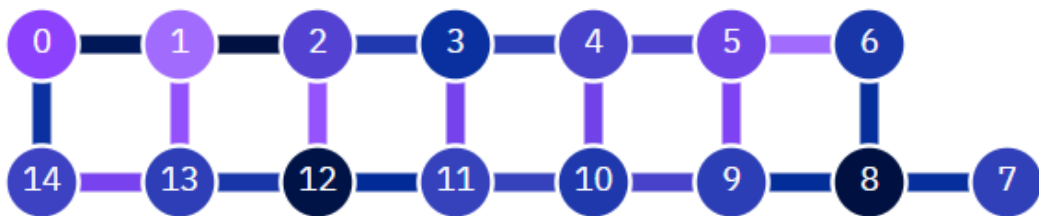
- Более объёмное тестирование, создание более подробной документации, подготовка учебных материалов, примеров и тестов для будущих участников данного проекта.

- Расширение возможностей генерации ошибок и проверки алгоритмов на устойчивость.
- Симуляция реальной топологии связей кубитов.
- Обработчик ошибок составления алгоритмов.
- Оптимизация.

Приложение

Ссылка на репозиторий с кодом и результатами тестирования алгоритма Шора: <https://github.com/Faert/ITLab>.

Пример топологии реального квантового компьютера IBM, где чем чернее кубит тем больше ошибка исполнения однокубитных гейтов и чем чернее связь между кубитами тем больше ошибка гейта CNOT, который используется для их связи:



Список литературы

- Algorithms for Quantum Computation: Discrete Logarithms and Factoring, Peter W. Shor, 1994
- Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, Peter W. Shor, 1996
- Circuit for Shor's algorithm using $2n+3$ qubits, St'ephane Beauregard, 2003
- Addition on a Quantum Computer, Thomas G. Draper, 2000
- Algorithms, S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, 2006
- Approximate quantum Fourier transform and decoherence, A. Barenco, A. Ekert, K. Suominen, P. Torma, 2008