

Вычисление алгоритма Берштейн-Вазирани в Qiskit

Computing the Bernstein-Vazirani algorithm in Qiskit

И.А.Юхновский

ноябрь 2020

Аннотация

В статье приведен практический пример использования фреймворка для квантовых вычислений на примере вычисления алгоритма Берштейн-Вазирани в Qiskit.

Abstract

The article provides a practical example of using a framework for quantum computing using the example of computing the Bernstein-Vazirani algorithm in Qiskit.

1 Введение

Qiskit - это платформа с открытым исходным кодом для работы с квантовыми компьютерами на уровне схем, импульсов и алгоритмов.

Основная цель Qiskit - создать программный стек, упрощающий использование квантовых компьютеров для всех. Однако Qiskit также стремится облегчить исследования по наиболее важным открытым вопросам, с которыми сегодня сталкиваются квантовые вычисления.

Продемонстрируем на практике как использовать Qiskit для реализации алгоритма Берштейн-Вазирани с возможностью запускать на симуляторах и реальных квантовых компьютерах. [1]

2 Задача Бернштейна — Вазирани

Алгоритм Бернштейна — Вазирани является расширенной версией алгоритма Дойча — Йожи (уже рассмотренный нами в предыдущей работе «Теоретическое исследование возможности построения квантового компьютера на источниках ионизирующего излучения»), поскольку вместо определения принадлежности функции к определённому классу — сбалансированная или постоянная (то есть принимает либо значение 0, либо 1 при любых аргументах) — алгоритм находит «спрятанный» вектор, позволяющий однозначно определить значение функции в любой точке cite{coles}.

Алгоритм Бернштейна — Вазирани демонстрирует в решаемой им задаче зазор между классическими и квантовыми алгоритмами по наименьшему требуемому количеству запросов к оракулу (чёрному ящику). Даже если разрешить использование вероятностных алгоритмов (с заранее ограниченной вероятностью ошибки), решение классической задачи потребует $O(n)$ обращений к оракулу, в то время как в квантовом алгоритме достаточно $O(1)$ обращений к нему [3].

Далее рассмотрим постановку задачи и алгоритмы решения предложенные в [4]

2.1 Классическая задача

Пусть существует оракул, преобразующий n -битное число в один бит:

$$f : \{0, 1\}^n \rightarrow \{0, 1\} \quad (1)$$

такой, что:

$$f(x) = a \cdot x \quad (2)$$

, где \cdot скалярное произведение вида $x \cdot y = x_1y_1 \oplus x_2y_2 \oplus \dots \oplus x_ny_n$.

Считаем, что один вызов функции f осуществляется за константное время. Требуется найти a .

2.2 Квантовая задача

Постановка задачи в квантовой модели похожая, но доступ к оракулу в ней осуществляется не напрямую через функцию f , а через линейный оператор U_f , действующий на систему из $n + 1$ кубита:

$$U_f = \sum_{x \in \{0,1\}^n, y \in \{0,1\}} |x\rangle\langle x| \otimes |y \oplus f(x)\rangle\langle y| \quad (3)$$

, где:

- $|x\rangle$ — кет-вектор, соответствующий квантовому состоянию x ,
- $\langle x|$ — бра-вектор, соответствующий квантовому состоянию x ,
- \otimes — произведение Кронекера,
- \oplus — сложение по модулю 2.

Квантовым состояниям 0 и 1 соответствуют векторы $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ и $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Вектор для совместного состояния $x_1 x_2 \dots x_n$ может быть представлен как произведение $|x_1 x_2 \dots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \dots \otimes |x_n\rangle$.

Аналогично классическому случаю, предполагается, что обращение к оракулу, вычисляющему результат применения оператора U_f к входящей системе из $n + 1$ кубита, выполняется за константное время.

Предполагается, что:

$$f(x) = a \cdot x \quad (4)$$

Требуется найти a .

2.3 Алгоритм

2.3.1 Классическая задача

В классическом случае при каждом вызове оракула возвращается один бит числа a , поэтому чтобы найти n -битное число a , нужно вызывать оракул n раз. Ниже приведён вариант n обращений к оракулу, позволяющих целиком восстановить a :

$$f(1000\dots 0_n) = a_1$$

$$f(0100\dots 0_n) = a_2$$

\vdots

$$f(0000\dots 1_n) = a_n$$

Количество обращений к оракулу в классическом случае равно $O(n)$, где n — количество бит числа a . Несложными теоретико-информационными рассуждениями можно показать, что эта оценка не улучшаема даже в рамках класса BPP ¹.

¹ [https://en.wikipedia.org/wiki/BPP_\(complexity\)](https://en.wikipedia.org/wiki/BPP_(complexity))

2.3.2 Квантовый алгоритм

Определяется n -кубитный оператор Адамара:

$$H^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x,y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle \langle x| \quad (5)$$

и учитывается, что применение оператора U_f к состоянию вида:

$$|x\rangle |- \rangle = |x\rangle \otimes |- \rangle \quad (6)$$

, где

$$|- \rangle = \frac{|0\rangle \otimes |-1\rangle}{\sqrt{2}} \quad (7)$$

дает в результате величину:

$$U_f(|x\rangle |- \rangle) = (-1)^{f(x)} |x\rangle |- \rangle \quad (8)$$

2.3.3 Пошаговая работа алгоритма

На первом шаге оператор Адамара $H^{\otimes(n+1)}$ применяется к $(n+1)$ -кубитному состоянию $|0\rangle^n |1\rangle$, состоящего из основного состояния $|0\rangle^n$ и вспомогательного бита $|1\rangle$:

$$|0\rangle^n |1\rangle \xrightarrow{H^{\otimes(n+1)}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |- \rangle \quad (9)$$

На втором шаге к предыдущему результату применяем оператор U_f :

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |- \rangle \xrightarrow{U_f} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle |- \rangle \quad (10)$$

На третьем шаге к первым n кубитам результата применяется оператор Адамара $H^{\otimes(n+1)}$ и с учетом 4 получаем:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle |- \rangle \xrightarrow{H^{\otimes(n+1)}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle |- \rangle \quad (11)$$

В результате измерение входного регистра даст значение $|a\rangle$. Таким образом, в квантовой постановке задачи достаточно $O(1)$ обращений к оракулу. В общем случае построение и использование оракула требует

$O(4^n)$ логических элементов, но это не учитывается при анализе алгоритма в данной модели, так как значимым для неё является только число обращений к оракулу.

Алгоритм в таком виде был реализован на 5- и 16-кубитных компьютерах IBM, также возможно собрать оптическую систему, как было описано в предыдущей работе для алгоритма Дойча.

В любой практической реализации алгоритма Бернштейна — Вазирани основную сложность составляет создание оракула, так как построение и использование оракула требует $O(4^n)$ логических элемента.

Кроме сложности построения оракула, практической реализации сопутствуют проблемы с точностью. Тестирование системы проводилось на 1-, 2- и 3-битных строках, на которых симулятор IBM-Qiskit (англ.) выдавал правильный ответ со 100 % точностью. Затем было проведено тестирование 1- и 2-битных строк на 5-кубитной машине ibmqx4 и 16-кубитной ibmqx5, где были зафиксированы ошибки вычислений и сильное отклонение от ожидаемого результата.

3 Реализация задачи Бернштейна — Вазирани на Qiskit в Jupyter Notebook

Qiskit - Python фреймворк. Его можно использовать как в отдельных приложениях, так и в симуляторах. Одним из подобных популярных симуляторов является Jupyter Notebook [2].

quantum_calc

13 ноября 2020 г.

1 Моделирование квантового компьютера

2 Квантовые вычисления

2.1 Базовые арифметические операторы

[1]: `3*5`

[1]: `15`

[2]: `8/4`

[2]: `2.0`

2.1.1 Комплексные числа

Комплексное число всегда $c = a + bj$

[2]: `import numpy as np`

[3]: `1j*1j`

[3]: `(-1+0j)`

[4]: `z=4+8j`

[5]: `w=5-6j`

[6]: `print("Real of Z:", np.real(z))`

Real of Z: 4.0

[7]: `print("Imag of Z:", np.imag(z))`

Imag of Z: 8.0

[8]: `z+w`

[8]: $(9+2j)$

2.1.2 Комплексное сопряжение

[9]: `np.conj(w)`

[9]: $(5+6j)$

[10]: `np.conj(z)`

[10]: $(4-8j)$

2.1.3 Норма/Модуль/Абсолютное значение

[11]: `np.abs(z)`

[11]: 8.944271909999916

[12]: `np.abs(w)`

[12]: 7.810249675906654

2.1.4 Строки и колонки

[13]: `row_vec = np.array([1, 2+2j, 3])`

[14]: `row_vec`

[14]: `array([1.+0.j, 2.+2.j, 3.+0.j])`

[15]: `col_vec = np.array([[1], [2+2j], [3]])`

[16]: `col_vec`

[16]: `array([[1.+0.j], [2.+2.j], [3.+0.j]])`

Строки в квантовой механике соответствуют $\langle A |$ Колонки соответствуют $| B \rangle$

2.1.5 Скалярное произведение двух векторов

[17]: `A = np.array([[1], [4-5j], [5], [-3]])`

[18]: `A`

```
[18]: array([[ 1.+0.j,
   [ 4.-5.j],
   [ 5.+0.j],
  [-3.+0.j]])
```

```
[19]: B = np.array([1, 5, -4j, -1j])
```

```
[20]: B
```

```
[20]: array([ 1.+0.j,  5.+0.j, -0.-4.j, -0.-1.j])
```

$\langle B | A \rangle$ будет:

```
[21]: np.dot(B,A)
```

```
[21]: array([21.-42.j])
```

2.1.6 Матрицы

```
[22]: M=np.array([[2-1j, -3],[-5j, 2]])
```

```
[23]: M
```

```
[23]: array([[ 2.-1.j, -3.+0.j],
   [-0.-5.j,  2.+0.j]])
```

```
[24]: M=np.matrix([[2-1j, -3],[-5j, 2]])
```

```
[25]: M
```

```
[25]: matrix([[ 2.-1.j, -3.+0.j],
   [-0.-5.j,  2.+0.j]])
```

2.1.7 Эрмитово сопряжение

```
[26]: M.H
```

```
[26]: matrix([[ 2.+1.j, -0.+5.j],
   [-3.-0.j,  2.-0.j]])
```

2.1.8 Тензорное произведение

```
[27]: np.kron(M,M)
```

```
[27]: matrix([[ 3. -4.j,  -6. +3.j,  -6. +3.j,   9. -0.j],
   [-5.-10.j,    4. -2.j,    0.+15.j,  -6. +0.j],
   [-5.-10.j,    0.+15.j,   4. -2.j,  -6. +0.j],
```

```
[ -25. +0.j,   0.-10.j,   0.-10.j,   4. +0.j]])
```

2.2 Кубиты, Сфера Блоха

pip install qiskit

```
[2]: import qiskit
```

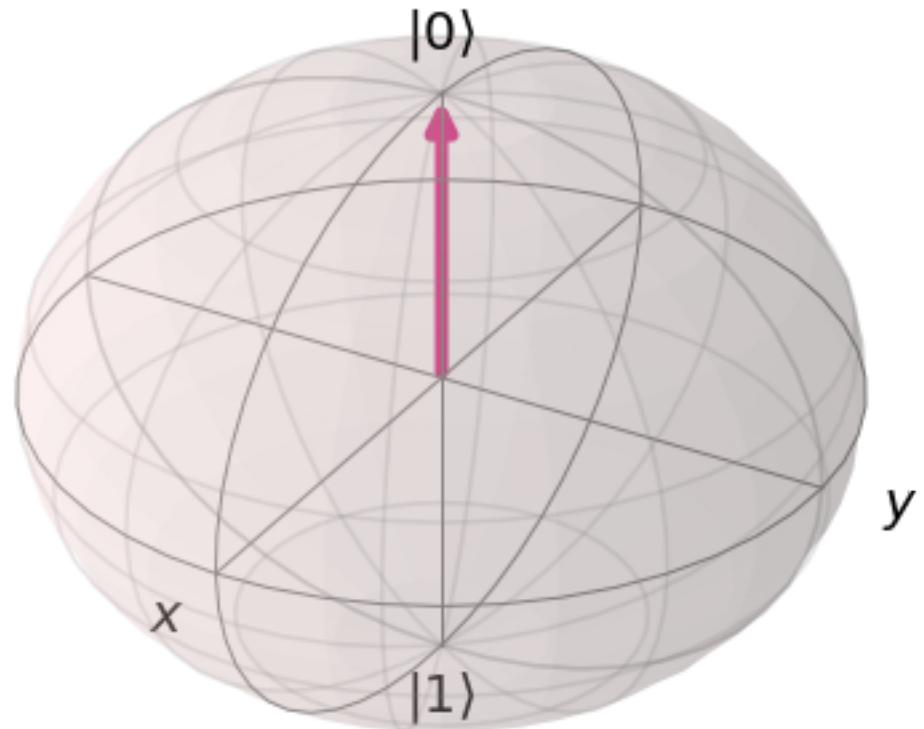
```
[3]: qiskit.__qiskit_version__
```

```
[3]: {'qiskit-terra': '0.15.2',
      'qiskit-aer': '0.6.1',
      'qiskit-ignis': '0.4.0',
      'qiskit-ibmq-provider': '0.9.0',
      'qiskit-aqua': '0.7.5',
      'qiskit': '0.21.0'}
```

```
[4]: from qiskit import *
from qiskit.visualization import plot_bloch_vector
plot_bloch_vector([0,0,1], title = 'Спин вверх')
```

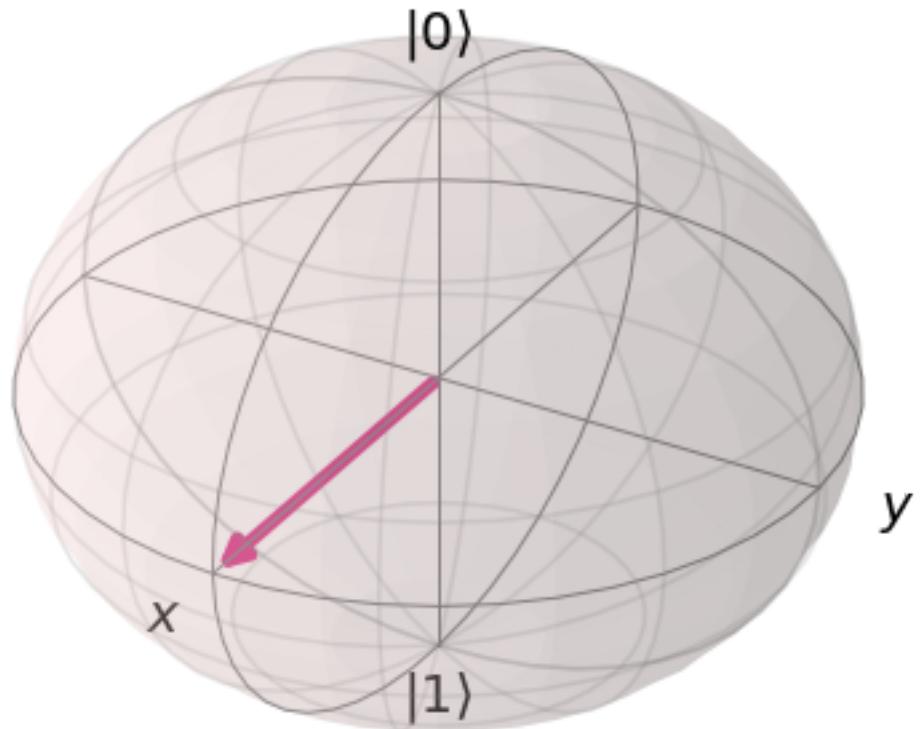
```
[4]:
```

Спин вверх



```
[31]: plot_bloch_vector([1,0,0])
```

```
[31]:
```



2.3 Базисные состояния

```
[32]: ket_zero = np.array([[1],[0]])
ket_one = np.array([[0],[1]])
```

```
[33]: np.kron(ket_zero, ket_zero)
```

```
[33]: array([[1],
           [0],
           [0],
           [0]])
```

```
[34]: np.kron(ket_zero, ket_one)
```

```
[34]: array([[0],
           [1],
```

```
[0],  
[0])
```

```
[35]: np.kron(ket_one, ket_zero)
```

```
[35]: array([[0],  
           [0],  
           [1],  
           [0]])
```

```
[36]: np.kron(ket_one, ket_one)
```

```
[36]: array([[0],  
           [0],  
           [0],  
           [1]])
```

2.4 Квантовые операторы

2.4.1 X на $|0\rangle$

```
[6]: qc = QuantumCircuit(1)  
qc.x(0)  
qc.draw('mpl')
```

```
[6]:
```



```
[7]: backend = Aer.get_backend('statevector_simulator')  
out = execute(qc, backend).result().get_statevector()  
print(out)
```

```
[0.+0.j 1.+0.j]
```

2.5 Z and Y операторы

```
[8]: qc.y(0)  
qc.z(0)  
qc.draw('mpl')
```

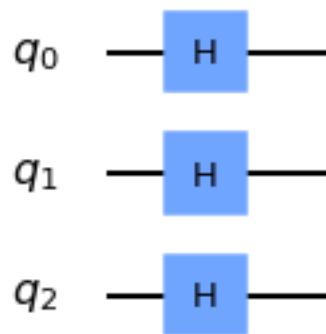
```
[8]:
```



2.6 Оператор Адамара

```
[9]: qc = QuantumCircuit(3)
for qubit in range(3):
    qc.h(qubit)
qc.draw('mpl')
```

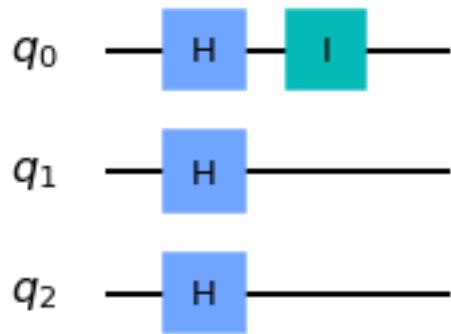
[9]:



2.7 Единичный оператор

```
[10]: qc.i(0)
qc.draw('mpl')
```

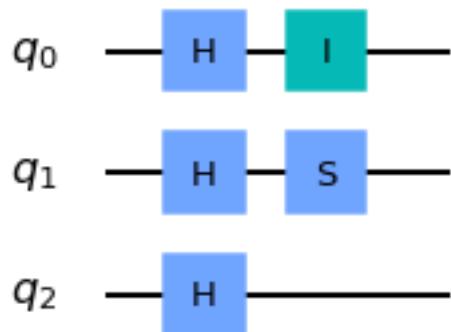
[10]:



2.8 S - Оператор

```
[11]: qc.s(1)  
qc.draw('mpl')
```

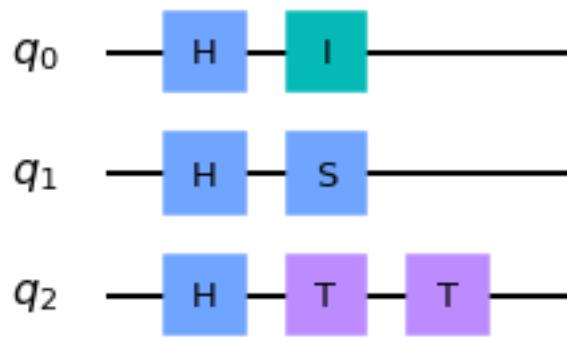
[11]:



2.9 T - Оператор

```
[13]: qc.t(2)  
qc.draw('mpl')
```

[13]:



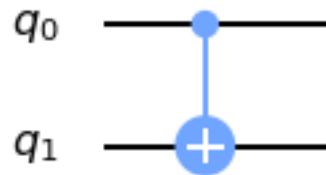
2.10 U - Оператор

...

2.11 C-Not Оператор

```
[17]: qc = QuantumCircuit(2)
qc.cx(0,1)
qc.draw('mpl')
```

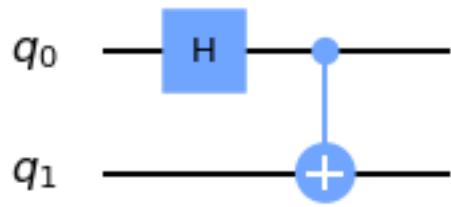
[17] :



2.12 Запутанность

```
[18]: qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0,1)
qc.draw('mpl')
```

[18] :



```
[19]: final_state = execute(qc,backend).result().get_statevector()
print(final_state)
```

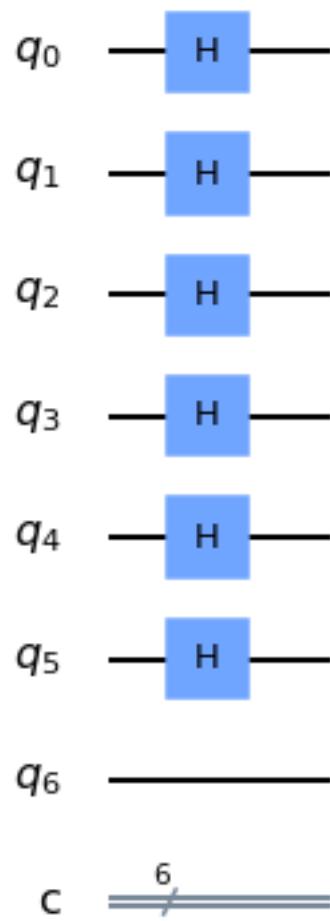
```
[0.70710678+0.j 0.           +0.j 0.           +0.j 0.70710678+0.j]
```

2.13 Берштейн-Вазирани Алгоритм

```
[ ]: s = 101011
```

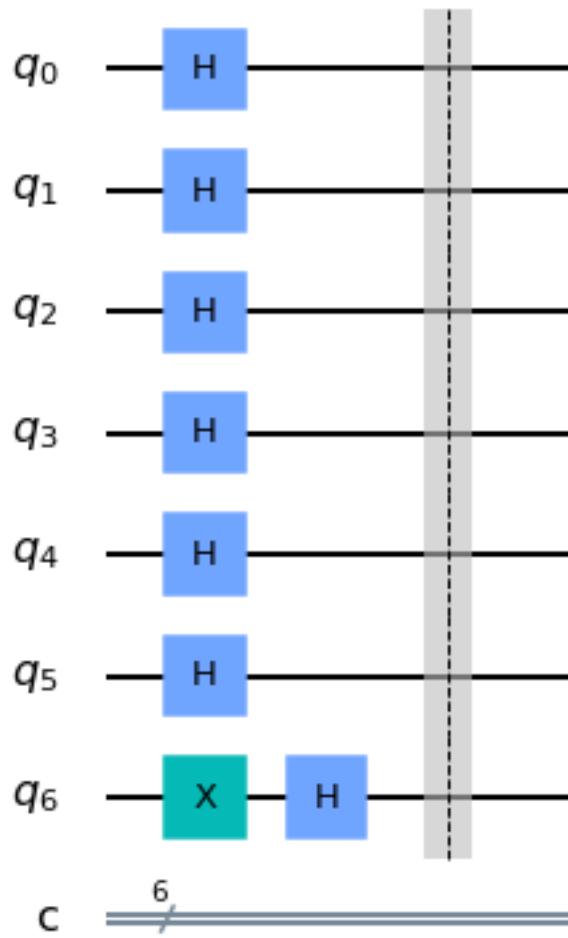
```
[20]: qc = QuantumCircuit(6+1, 6)
qc.h([0,1,2,3,4,5])
qc.draw('mpl')
```

```
[20]:
```



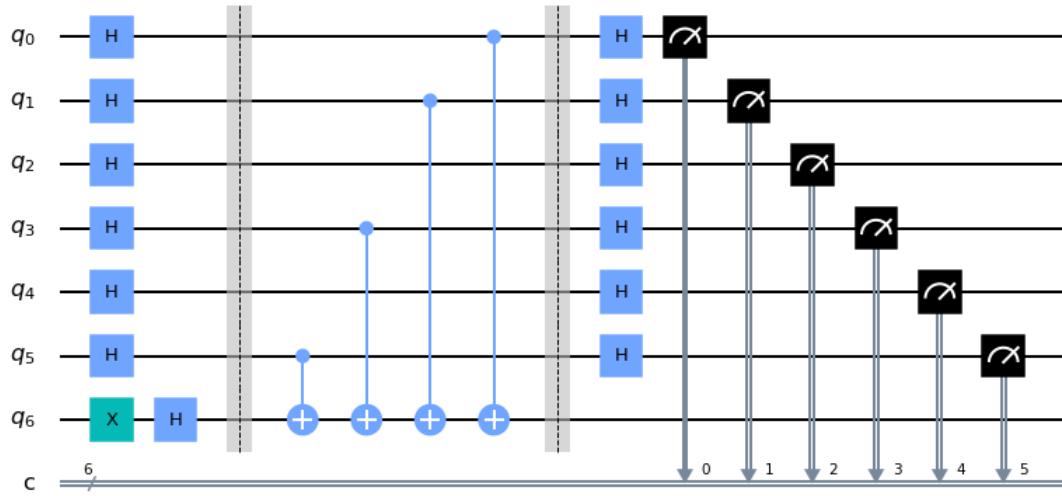
```
[21]: qc.x(6)
qc.h(6)
qc.barrier()
qc.draw('mpl')
```

[21]:



```
[22]: qc.cx(5,6)
qc.cx(3,6)
qc.cx(1,6)
qc.cx(0,6)
qc.barrier()
qc.h([0,1,2,3,4,5])
qc.measure([0,1,2,3,4,5],[0,1,2,3,4,5])
qc.draw('mpl')
```

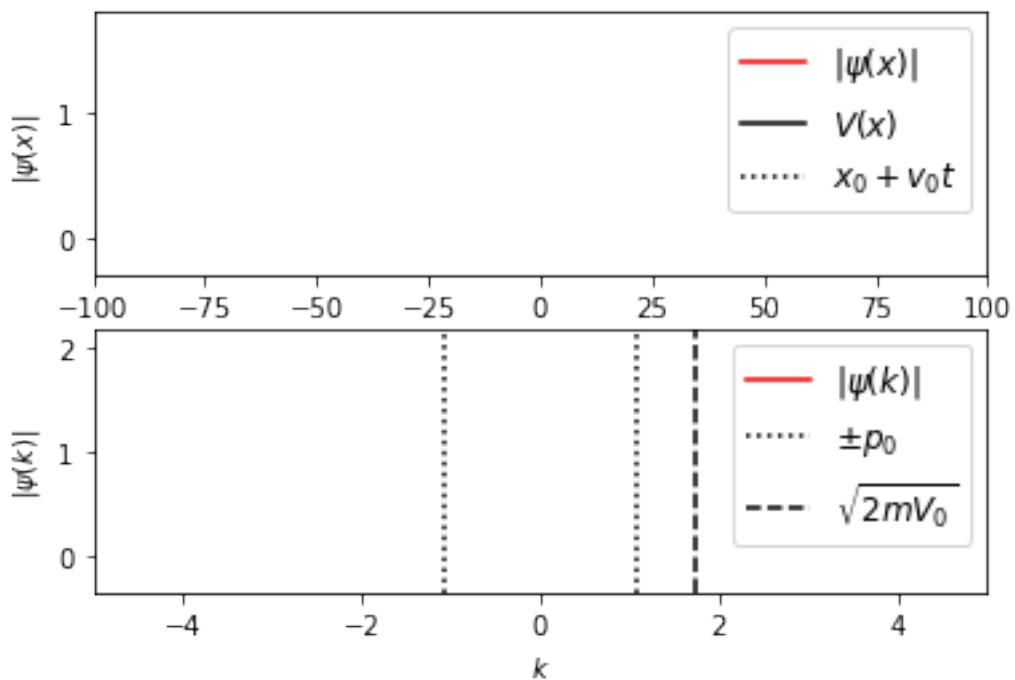
[22]:



```
[24]: simulator= Aer.get_backend('qasm_simulator')
result = execute(qc,backend=simulator, shots = 1).result()
counts=result.get_counts()
print(counts)
```

```
{'101011': 1}
```

[27] :



4 Заключение

В статье было продемонстрирована легкость разработки квантового алгоритма, возможность проводить численные квантовые эксперименты и запускать их на симуляторах и реальных квантовых компьютерах при использовании Qiskit. В отличии от реализаций на языках программирования, фреймворк предоставляет абстрактный уровень для интеграции с квантовыми компьютерами и их симуляторами, тем самым, является является удобным фронтендом в исследовании квантовых компьютеров и квантовых алгоритмов. Также, к неоспоримым преимуществам относятся возможности визуализации на любом из алгоритмических этапов.

В ноябре был анонсирован CAD система по хардварному моделированию квантовых устройств от концепта до реализации:

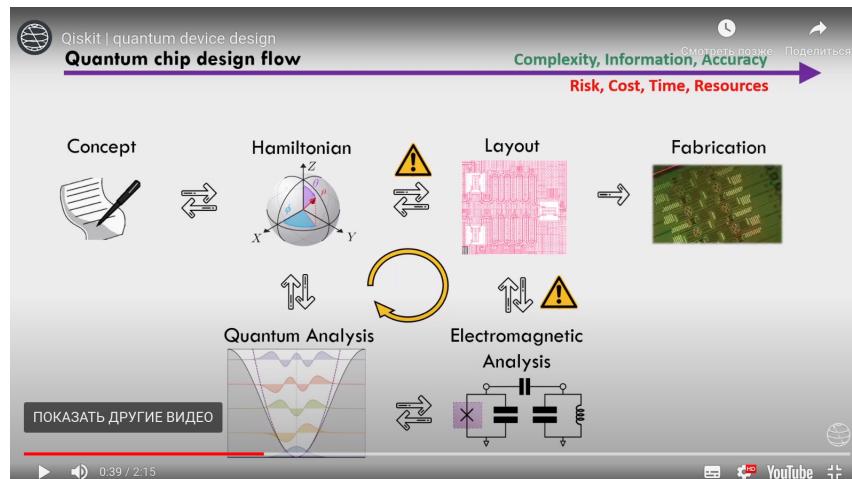


Рис. 1: Qiskit этапы проектирования

Позволяет проектировать устройства, выполнять необходимы расчеты и проверки, производить компановку на подложке.

Поскольку на текущий момент есть только регистрация, без доступа к продукту, то сложно сказать, возможна ли интеграция кастомизированных устройств. Если это будет предусмотрено, то видится перспективным разработка программных квантовых компонентов по открытым API этой CAD.

В заключении определим место проводимого исследования в современном технологическом мире - это рисунки 6 и 7 - построение квантовых устройств с использованием ионизирующего излучения.

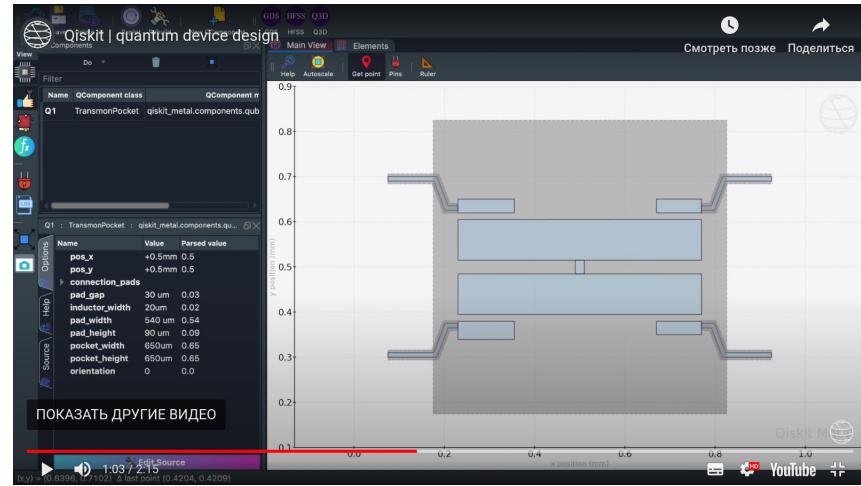


Рис. 2: Qiskit проектирование кубита

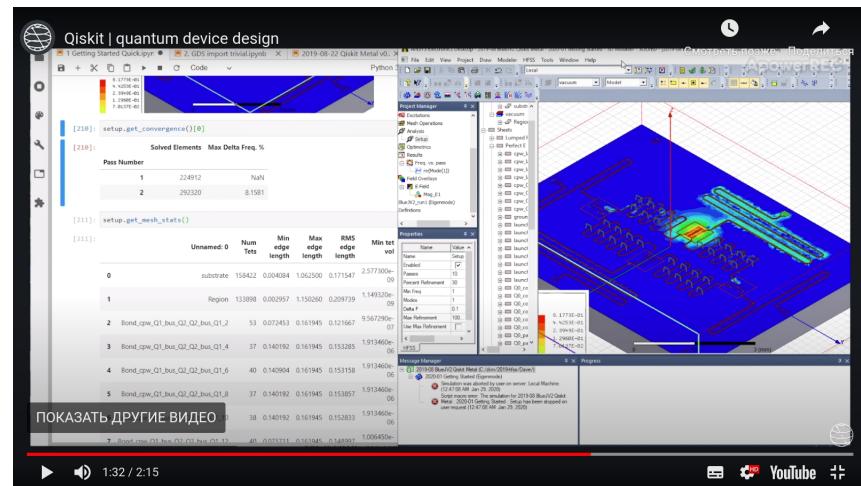


Рис. 3: Qiskit 3D-компьюнавка

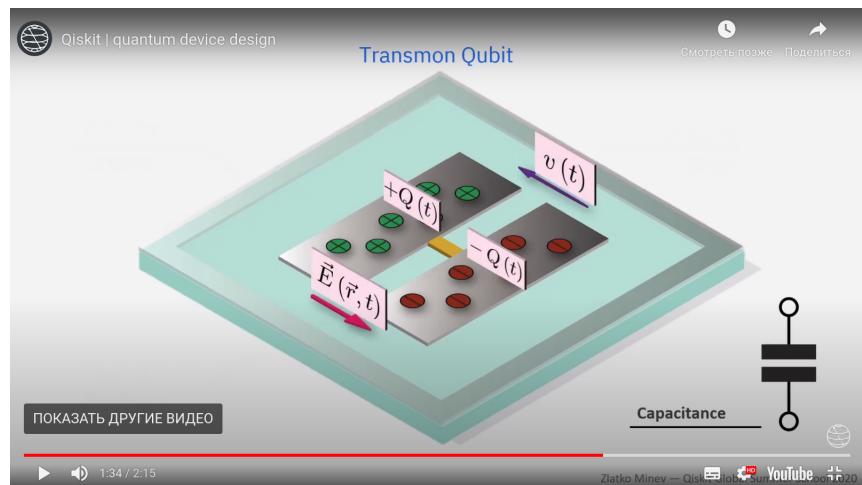


Рис. 4: Квантовый анализ конденсатора

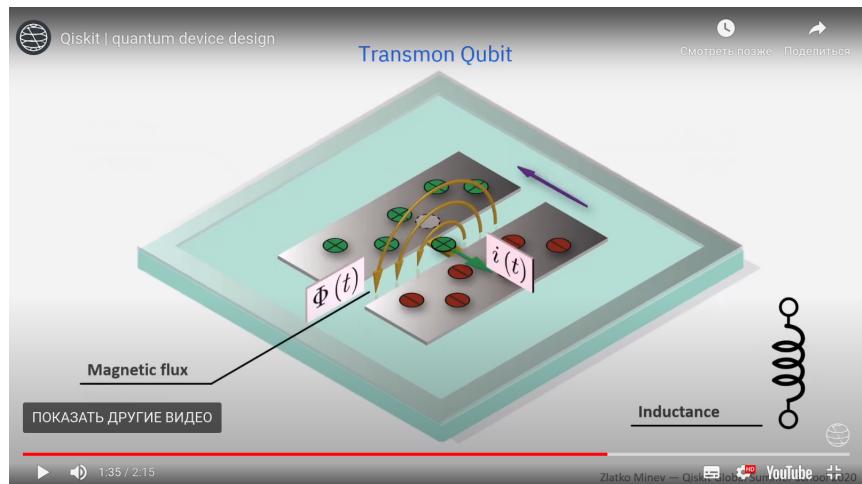


Рис. 5: Квантовый анализ индуктивности

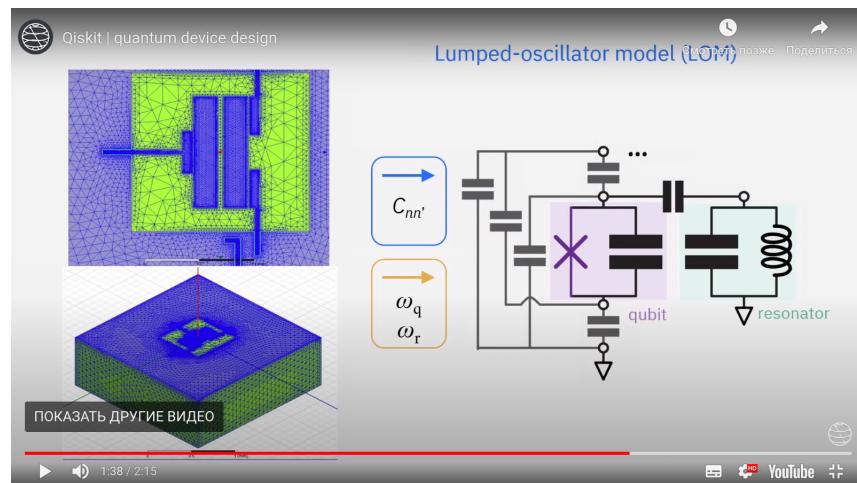


Рис. 6: Принципиальное моделирование систем

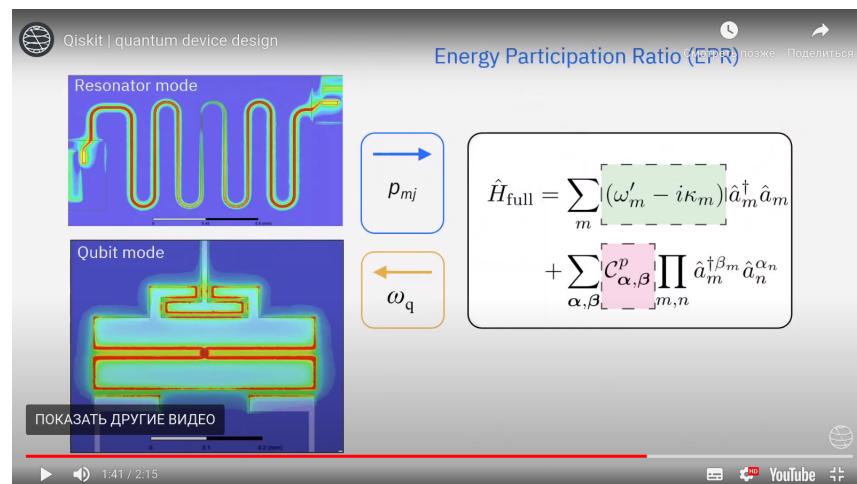


Рис. 7: Квантомеханический расчет систем

Список литературы

- [1] QISKit 0.23.1 DOCUMENTATION <https://qiskit.org/documentation/>
- [2] Jupiter <https://jupyter.org/>
- [3] Hidary, J. D. (2019). Quantum Computing: An Applied Approach.
doi: <https://doi.org/10.1007/978-3-030-23922-0>
- [4] Patrick J. Coles, Stephan Eidenbenz, Scott Pakin, Adetokunbo Adedoyin,
John Ambrosiano. Quantum Algorithm Implementations for Beginners
// arXiv:1804.03719 [quant-ph]. — 2018.
- [5] QISKit METAL <https://qiskit.org/metal>