RESEARCH ON APPLICATION OF GROVER'S QUANTUM ALGORITHM TO DNA SEQUENCING

Dung Van Lu, Huynh Phuong Anh*, Huynh Bao Nguyen,
Nguyen Ngoc Minh Tri, Cao Thi My Hao, Nguyen Thi Hong
The University of Danang - University of Science and Education

ARTICLE INFO

Received: **20/3/2024**
Revised: **23/5/2024**
Published: **24/5/2024**

KEYWORDS

Quantum Algorithm
Quantum computing
Grover's algorithm
Unstructured search
DNA sequencing

ABSTRACT

In order to search through an unstructured database of N elements, Grover's quantum search algorithm has O ($\sqrt{N}$) time complexity and uses O (log N) storage space thanks to the application of quantum mechanical properties (such as, superposition, entanglement,...) in each step of this algorithm. In this article, we studied the Grover quantum algorithm and clarified the quantum supremacy of the algorithm by analyzing the "behavior" of quantum mechanical properties in each step of this algorithm. In addition, we calculated and implemented on IBM quantum computers through the Qiskit platform in applicating for the problem of DNA sequencing with a string of length N = 8. The results showed that the Grover's quantum search algorithm has superior search capabilities compared to existing algorithms as the quantum computer has a sufficient number of qubits, which helps to reduce the time and resources needed to determine the gene sequence and provides more effective methods for molecular biology.

NGHIÊN CỨU ỨNG DỤNG THUẬT TOÁN LƯỢNG TỬ GROVER TRONG GIẢI TRÌNH TỰ DNA

Dụng Văn Lữ, Huỳnh Phương Anh*, Huỳnh Bảo Nguyên,
Nguyễn Ngọc Minh Trí, Cao Thị Mỹ Hảo, Nguyễn Thị Hồng
Trường Đại học Sư phạm – Đại học Đà Nẵng

THÔNG TIN BÀI BÁO

Ngày nhận bài: **20/3/2024**
Ngày hoàn thiện: **23/5/2024**
Ngày đăng: **24/5/2024**

TỪ KHÓA

Thuật toán lượng tử
Máy tính lượng tử
Thuật toán Grover
Tìm kiếm phi cấu trúc
Giải trình tự DNA

TÓM TẮT

Để tìm kiếm trên một cơ sở dữ liệu phi cấu trúc gồm N phần tử, thuật toán lượng tử Grover có độ phức tạp về thời gian O($\sqrt{N}$) và sử dụng O(log N) không gian lưu trữ nhờ ứng dụng của các tính chất cơ học lượng tử. Trong bài báo này, chúng tôi nghiên cứu thuật toán lượng tử Grover và làm rõ tính ưu việt của thuật toán bằng cách phân tích các "hành xử" của tính chất cơ học lượng tử (như chồng chất, vướng víu,...) trong từng bước của thuật toán. Đồng thời, chúng tôi tính toán và triển khai trên máy tính lượng tử IBM thông qua nền tảng Qiskit trong bài toán giải trình tự DNA với chuỗi có độ dài N = 8. Kết quả cho thấy thuật toán lượng tử Grover có khả năng tìm kiếm ưu việt hơn các thuật toán hiện có khi máy tính lượng tử có số qubit đủ dùng, điều này giúp giảm thời gian và nguồn lực cần thiết để xác định chuỗi gen và đồng thời cung cấp phương pháp hiệu quả hơn cho nghiên cứu sinh học phân tử.

DOI: https://doi.org/10.34238/tnu-jst.9932

* Corresponding author. *Email: panh26072002@gmail.com*

1. Giới thiệu

Đối với bài toán tìm kiếm từ dữ liệu gồm *N* phần tử, đặc biệt là một cơ sở dữ liệu phi cấu trúc, thời gian cần thiết đối với các giải pháp cổ điển hiện nay là tuyến tính, tức là bậc O(*N*) về thời gian. Trong khi đó, thuật toán lượng tử Grover (GA) chỉ mất thời gian O(√*N*) và sử dụng O(log *N*) không gian lưu trữ [1]. Nó là thuật toán lượng tử tìm kiếm cơ sở dữ liệu phi cấu trúc nhanh nhất với khả năng tăng tốc bậc hai, đây là điểm khác biệt với các thuật toán lượng tử khác. Khi *N* càng lớn thì GA càng thể hiện sự tăng tốc rõ hơn và tiết kiệm được tài nguyên hơn. GA là một thuật toán lượng tử, nó sử dụng một vài thuộc tính cốt lõi của tính toán lượng tử như chồng chất lượng tử và vướng víu lượng tử [2]. Đơn vị thông tin cơ bản trong tính toán lượng tử là bit lượng tử (qubit) được mô tả bởi trạng thái lượng tử là $|0\rangle$ và $|1\rangle$. Một trạng thái bất kì trong hệ một qubit là chồng chất của các qubit cơ bản $\psi = a|0\rangle + b|1\rangle$, trong đó a, b là những số phức thoả mãn $|a|^2 + |b|^2 = 1$ và $|a|^2, |b|^2$ là xác xuất mà ψ ở trạng thái $|0\rangle, |1\rangle$ tương ứng. Khi hệ có n qubit thì trạng thái tổng quát là chồng chất và vướng víu từ 2^n trạng thái cơ bản. Nhờ các tính chất lượng tử này mà các thuật toán lượng tử có khả năng xử lý nhanh hơn các thuật toán cổ điển.

Trong những nghiên cứu gần đây, GA được ứng dụng rộng rãi trong các vấn đề liên quan tìm kiếm dữ liệu phi cấu trúc hay tối ưu hoá. GA làm tăng biên độ xác suất của trạng thái được đánh dấu để tăng tốc nhiều loại thuật toán khác để giải bài toán 3SAT [3]. GA giúp tăng tốc khả thi cho các bài toán hộp đen về độ phức tạp của truy vấn lượng tử, bao gồm tính khác biệt của phần tử [4]. Người ta cũng áp dụng GA trong hệ thống điện và năng lượng để giải quyết ba vấn đề cốt lõi: độ tin cậy, tối ưu hóa và kiểm soát [5]. Hoặc tiềm năng ứng dụng GA trong máy học lượng tử [6]. Bên cạnh đó, nó cũng được cải tiến và kết hợp với các thuật toán khác để giải những vấn đề phức tạp hơn, điển hình như việc đưa ra GA phân tán [7], hay triển khai kết hợp với các hàm băm cổ điển MD5, SHA-1, SHA-2 và SHA-3 [8], hoặc triển khai cho AES-128, -192 và -256 trong phần mềm lượng tử [9].

Giải trình tự DNA là một lĩnh vực nghiên cứu quan trọng trong ngành sinh học phân tử, nhằm xác định chuỗi gen và thông tin mã hóa của các phân tử. Việc nghiên cứu trình tự DNA được coi là một yếu tố quan trọng vì nó mang các chi tiết về bộ gen có thể được các nhà nghiên cứu sử dụng để dự đoán sớm bệnh tật. Công cụ giải trình tự DNA được dùng phổ biến là Sanger và các thế hệ mới [10]. Tuy nhiên, khi kích thước dữ liệu lớn và nhiều thông tin trong xử lý thì thuật toán và máy tính cổ điển sẽ có những hạn chế nhất định.

Gần đây, Marina và cộng sự đã kết hợp thuật toán tìm đường lượng tử và khuếch đại biên độ Gaussian để giải trình tự DNA [11]. Sarkar và cộng sự đã đề xuất một thuật toán lượng tử để giải quyết lĩnh vực xử lý dữ liệu đầy thách thức để tái cấu trúc trình tự bộ gen [12]. Tuy nhiên, việc áp dụng thuật toán lượng tử trong giải trình tự DNA vẫn đang trong quá trình phát triển và đòi hỏi sự tích hợp chặt chẽ giữa kiến thức lượng tử và sinh học. Điều này đặt ra nhiều thách thức và cơ hội nghiên cứu mới, mở rộng phạm vi của việc áp dụng công nghệ lượng tử trong thời kỳ hiện đại của khoa học [13].

Trong bài báo này, chúng tôi phân tích ở mục 2 các bước triển khai thuật toán lượng tử Grover từ góc nhìn cơ học lượng tử. Ở mục 3 trình bày tính toán mạch lượng tử và triển khai thuật toán trên máy tính lượng tử thực thông qua nền tảng Qiskit đối với bài toán giải trình tự DNA của một đoạn axit nucleic có độ dài $N = 8$ với mẫu cần tìm có độ dài $M = 2$. Cuối cùng là phần kết luận về tính ưu việt của thuật toán lượng tử Grover và tiềm năng ứng dụng của nó để giải quyết các bài toán tối ưu trong tương lai.

2. Thuật toán lượng tử Grover

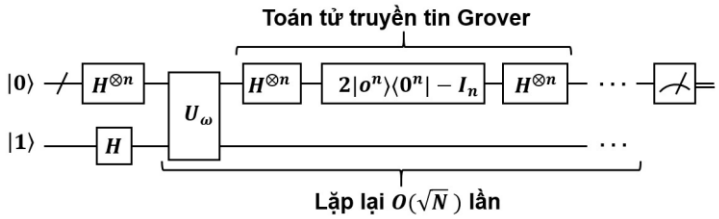
2.1. Vấn đề cần tìm kiếm

Giả sử ta có một ô cơ sở dữ liệu gồm $N = 2^n$ phần tử tạo thành tập hợp $X = \{X_1, X_2, X_3, \dots, X_N\}$ và cho trước một hàm Boolean $f = X \to \{0,1\}$. Ta sử dụng toán tử U_ω, sao cho:

$$U_\omega|x\rangle = \begin{cases} |x\rangle \text{ nếu } x \neq \omega \\ -|x\rangle \text{ nếu } x = \omega \end{cases} \#(1)$$

Mục tiêu của GA là tìm ra chỉ số $|\omega\rangle$ thoả mãn điều kiện trên [1].

2.2. Biểu diễn mạch lượng tử của thuật toán Grover



Hình 1. *Mạch lượng tử biểu diễn thuật toán lượng tử Grover*

GA [1] có thể mô tả ngắn gọn như Hình 1, các bước của thuật toán được diễn ra như sau:

Bước 1: Khởi tạo trạng thái. Thanh ghi thứ nhất gồm n qubit được nhập vào từ trạng thái $|0\rangle$. Số n qubit được chọn tối thiểu sao cho $n = [\log_2 N]$, với N là số phần tử của dữ liệu cần tìm. Thanh ghi thứ hai gồm một qubit và được thiết lập từ trạng thái $|1\rangle$:

$$|\psi_o\rangle = |0\rangle^{\otimes n}|1\rangle \#(2)$$

Mỗi cổng Hadamard đơn tác dụng lên mỗi qubit đơn, tạo ra trạng thái chồng chất. Như vậy, sau bước 1, tức là đầu vào hộp đen (*oracle*) U_ω, ta có trạng thái lượng tử:

$$|\psi_1\rangle = H^{\otimes n+1}|\psi_o\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\right)^{\otimes n+1}\begin{bmatrix}1\\0\end{bmatrix}^{\otimes n}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{N}}\sum_{x=1}^{N}|x\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |s\rangle|-\rangle \#(3)$$

Bước 2: Thực hiện vòng lặp Grover $r(N)$ lần. Hàm $r(N)$, có độ phức tạp $O(\sqrt{N})$, được miêu tả như sau:

(a) Thực hiện toán tử U_ω (còn gọi là *hàm oracle, hàm trong hộp đen*)

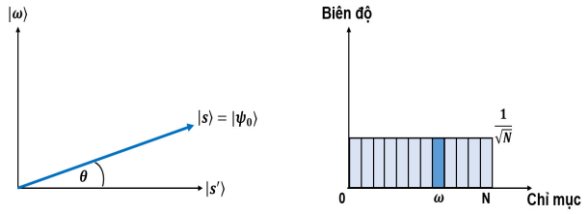
$$|\psi_2\rangle = U_\omega|\psi_1\rangle = \frac{1}{\sqrt{N}}\sum_{x=1}^{N}|x\rangle\frac{|0\oplus f(x)\rangle - |1\oplus f(x)\rangle}{\sqrt{2}} = \frac{1}{\sqrt{N}}\sum_{x=1}^{N}|x\rangle(-1)^{f(x)}|-\rangle \#(4)$$

Đối với mọi $|x\rangle$, với $f(x) = 0$ thì biên độ của nó không thay đổi. Ngược lại, biên độ thành âm.

(b) Thực hiện toán tử U_s. Sau mỗi lần lặp U_s, trạng thái được xoay về gần với trạng thái $|i\rangle = |\omega\rangle$ hơn, với góc xoay θ_G: $\theta_G = 2\arcsin(1/\sqrt{N})$

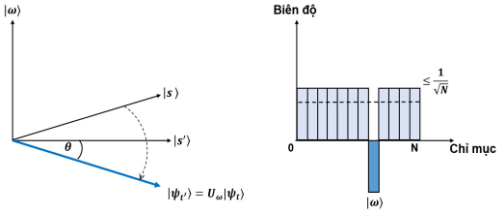
Bước 3: Thực hiện phép đo. Kết quả của phép đo sẽ là λ_ω với xác suất tiến tới 1 khi $N \gg 1$. Từ λ_ω ta có thể tìm thấy $|\omega\rangle$. Sau k lần lặp, trạng thái của hệ n qubit về gần với trạng thái $|\omega\rangle$, số k nhỏ nhất để xác suất trên gần với 1 là $k \approx \pi\sqrt{N}/4$. Như vậy, thuật toán tìm kiếm Grover có độ phức tạp $O(\sqrt{N})$ về thời gian nên sẽ tìm kiếm nhanh hơn khi dùng thuật toán toán cổ điển.

2.3. Biểu diễn vector quay của thuật toán Grover



(a) *(b)*

Hình 2. *Trạng thái ban đầu: (a) biểu diễn vector $|s\rangle$ và (b) biên độ của phần tử cần tìm*



(a) *(b)*

Hình 3. *Lần quay đầu tiên: (a) vector $|s\rangle$ với góc quay θ và (b) biên độ của phần tử cần tìm*

***Bước 1*:** Khởi tạo vector $|s\rangle$ (Hình 2). Hình 2(a) mô tả trạng thái $|\omega\rangle$ cần tìm bằng cách tạo vector $|s'\rangle$ trực giao với nó, tức là $\langle\omega|s'\rangle = 0$, và vị trí của $|s\rangle$ trên đồ thị tạo bởi hệ trục vuông góc $|\omega\rangle$ và $|s'\rangle$ được xác định là chồng chất của hai trạng thái cơ sở này:

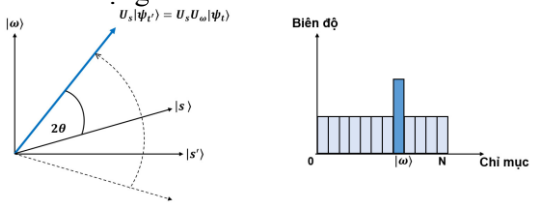
$$|s\rangle = \sin\theta|\omega\rangle + \cos|s'\rangle; \qquad \theta = \arcsin\langle s|\omega\rangle = \arcsin\frac{1}{\sqrt{N}} \#(5)$$

Trong bước này, biên độ của $|\omega\rangle$ chưa phân biệt với các phần tử còn lại (Hình 2(b)).

***Bước 2*:** Áp dụng toán tử U_ω (*oracle*) lên trạng thái $|s\rangle$ và làm nó quay 1 góc θ. Lúc nào, biên độ xác xuất của $|\omega\rangle$ bị lật ngược về âm như được mô tả ở Hình 3.

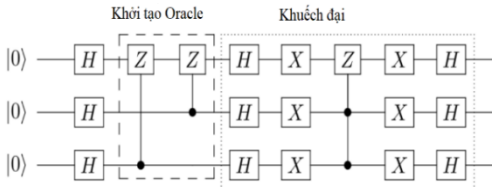
***Bước 3*:** Thực hiện toán tử U_s cho trạng thái $|s\rangle$ trong đó: $U_s = 2|s\rangle\langle s| - I$ khuếch đại vector màu xanh $|s\rangle$ gần hơn với $|\omega\rangle$. Hình 4 cho thấy phép biến đổi làm tăng biên độ của $|\omega\rangle$ cần tìm lên khoảng 3 lần giá trị ban đầu, đồng thời giảm các biên độ của các trạng thái khác.

Cuối cùng, ta thực hiện phép đo, trạng thái nào cho xác suất (bình phương biên độ) cao nhất chính là trạng thái cần tìm.



(a) *(b)*

Hình 4. *(a) Trạng thái $|s\rangle$ gần hơn về phía $|\omega\rangle$; (b) biên độ của $|\omega\rangle$ tăng dần*



Hình 5. *Sơ đồ mạch GA với 3 qubit*

3. Ứng dụng thuật toán lượng tử Grover để giải trình tự DNA

DNA là chuỗi polyme dài, dạng sợi, chứa gen di truyền quy định mọi hoạt động sống của các sinh vật và nhiều loài virus. Nó có bốn phân tử nucleic: adenine (A), cytosine (C), guanine (G) và thymine (T). Các cặp adenine với thymine và guanine với cytosine, được biểu thị bằng A-T và G-C, được gọi là cặp bazơ (bp) trong DNA [10].

Trong ví dụ của nghiên cứu này, chúng tôi lấy một đoạn chuỗi DNA loại hCoV-19 có đoạn gen như sau: ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGAT… Giả sử một đoạn bị lỗi thymine "A*T*AC" (được đặt là chuỗi F) và cần dò T ở vị trí nào trong chuỗi này.

Để giải trình tự DNA bằng GA, các nucleotide được mã hóa thành các trạng thái lượng tử sau:

$$|A\rangle \to |00\rangle; |T\rangle \to |11\rangle; |C\rangle \to |01\rangle; |G\rangle \to |10\rangle \#(6)$$

Khi đó, chuỗi cần tham chiếu $F = \{ATAC\}$ được mã hóa thành $|F\rangle = |00110001\rangle$ có độ dài mẫu $N = 8$. Chuỗi cần tìm $P = \{T\}$ được mã hoá thành $|P\rangle = |11\rangle$ có độ dài $M = 2$. Ta cần tìm vị trí xuất hiện của mẫu $|P\rangle$ trong chuỗi $|F\rangle$. Dưới đây, chúng tôi tính toán và triển khai để giải quyết vấn đề này trên nền tảng Qiskit.

3.1. Tính toán theo mô hình mạch

Hình 5 mô tả mạch lượng tử GA cho mạch 3 qubit được sử dụng để tính toán trong trường hợp này. Trạng thái từ đầu vào được khởi tạo $|\psi_o\rangle = |0\rangle^{\otimes n}|1\rangle$, sau khi áp dụng cổng Hadamard [2] tạo ra sự chồng chất của tất cả các trạng thái có biên độ bằng nhau:

$$|\psi_1\rangle = \frac{1}{2\sqrt{2}}(|0,1\rangle + |1,2\rangle + |2,3\rangle + |3,4\rangle + |4,5\rangle + |5,6\rangle + |6,7\rangle + |7,0\rangle) \#(7)$$

Trong đó, mỗi trạng thái chỉ mục $|i,j\rangle$ ứng với i, j là vị trí trong chuỗi $|F\rangle$, nghĩa là, trạng thái $|0,1\rangle$ biểu diễn cho "00", $|1,2\rangle$ biểu diễn cho "01", …, $|7,0\rangle$ biểu diễn cho "10" trong chuỗi $|F\rangle$.

Với mẫu $|P\rangle = |11\rangle$ nên ta cần áp dụng U_1 hai lần. Bắt đầu với kí hiệu đầu tiên nếu chúng bằng nhau thì trạng thái tương ứng với chỉ số đầu tiên sẽ bị đảo ngược, nếu không thì *oracle* sẽ giữ nguyên trạng thái. Trạng thái thu được sau khi áp dụng U_1 như sau:

$$U_1|\psi_1\rangle = \frac{1}{2\sqrt{2}}(|0,1\rangle + |1,2\rangle - |2,3\rangle - |3,4\rangle + |4,5\rangle + |5,6\rangle + |6,7\rangle - |7,0\rangle)\#(8)$$

Sau đó, sau khi áp dụng Oracle tương tự cho kí hiệu thứ hai:

$$|\psi\rangle = U_1U_1|\psi_1\rangle = \frac{1}{2\sqrt{2}}(|0,1\rangle - |1,2\rangle + |2,3\rangle - |3,4\rangle + |4,5\rangle + |5,6\rangle - |6,7\rangle - |7,0\rangle)\#(9)$$

Các trạng thái có biên độ dương (dấu '+') sẽ là mẫu cần tìm. Nhưng ngoài trạng thái $|2,3\rangle$ phù hợp với kết quả cần tìm, còn có các trạng thái khác cũng có dấu dương. Vấn đề này được giải quyết bằng cách áp dụng toán tử khuếch đại $U_s = 2|s\rangle\langle s| - I$, trạng thái cần tìm sẽ được khuếch đại nhiều hơn các trạng thái khác. Kết quả đo sẽ cho ở dạng xác suất, đây là điểm đặc biệt so với thuật toán cổ điển. Tiếp theo, chúng ta sẽ thu được kết quả chính xác khi triển khai trên Qiskit.

3.2. Triển khai thuật toán trên nền tảng Qiskit

Sau khi nhập lệnh 'import' từ các thư viện và module cần thiết để thực hiện trong môi trường lượng tử trên nền tảng Qiskit [14]. Chúng tôi chèn thêm thư viện Qibo và lệnh khởi tạo mạch lượng tử cho thuật toán:

```
def initialize( N, M, s, string_w, pattern_p):
    for i in range(N):
        if string_w[i] == 1:
            c.add(gates.X(i))
    for i in range(M):
        if pattern_p[i] == 1:
            c.add(gates.X(i + N))
    for i in range(int(s/2)):
        c.add(gates.H(i + N + M))
    for i in range(int(s/2)):
        c.add(gates.CNOT(i + N + M,i + int(s/2) + N + M))
    for i in range(int(s/2)):
        if i == (int(s/2)-1):
            c.add(gates.X(N + M + s - 1))
        else:
            c.add(gates.X(int(s/2) + i +  M).controlled_by(*range(int(s/2) + i + N + M + 1, N + M + s)))
    return c
```

Đoạn code sau đây được viết để thực hiện việc khuếch đại trạng thái cần tìm bằng toán tử phản xạ tức là đảo trạng thái có biên độ cao hơn các trạng thái khác:

```
def diffusion(N, M, s):
    c.add([gates.H(i) for i in range(N + M, N + M + s)])
    c.add([gates.X(i) for i in range(N + M, N + M + s)])
    c.add(gates.Z(N + M + s-1).controlled_by(*range(N + M, N + M + s-1)))
    c.add([gates.X(i) for i in range(N + M, N + M + s)])
    c.add([gates.H(i) for i in range(N + M, N + M + s)])
    return c
```

Tiếp theo là đoạn code để xác định trạng thái cần tìm nếu chuỗi và mẫu khớp nhau. Sau đó sẽ trả về đầu ra là mạch lượng tử với các biên độ của trạng thái trên trung bình bị đảo.

```
def oracle(N,M,s,p,equal):
    controls = [3, 5, 6, 1, 2, 4, 0]
    inp_list = [*range(N + M, N + M + int(s/2))]
    combi = [ ]
    for i in range(1, len(inp_list) + 1):
        combi.extend(list(itertools.combinations(inp_list, r = i)))
```

```
    for i in range(len(combi)+1):
        if  i == len(combi):
            if p == 0:
                c.add(gates.X(N - 1))
                c.add(gates.X(N))
            c.add(gates.Z(N + M + 2).controlled_by(N-1, N, N + M, N + M + 1))
            if p == 0:
                c.add(gates.X(N - 1))
                c.add(gates.X(N))
        else:
            c.add([gates.X(j) for j in list(combi[i])])
            if p == 0:
                c.add(gates.X(controls[i]))
                c.add(gates.X(N))
            c.add(gates.Z(N + M + 2).controlled_by(controls[i], N, N + M, N + M + 1))
            if p == 0:
                c.add(gates.X(controls[i]))
                c.add(gates.X(N))
            c.add([gates.X(j) for j in list(combi[i])])
    if equal is True:
        diffusion(N, M, s)
    inp_list = [*range(N + M + int(s/2), N + M + s)]
    combi = [ ]
    for i in range(1, len(inp_list) + 1):
        combi.extend(list(itertools.combinations(inp_list, r = i)))
    for i in range(len(combi)+1):
        if  i == len(combi):
            if p == 0:
                c.add(gates.X(N - 1))
                c.add(gates.X(N + 1))
            c.add(gates.Z(N + M + 5).controlled_by(N-1, N + 1, N + M + 3, N + M + 4))
            if p == 0:
                c.add(gates.X(N - 1))
                c.add(gates.X(N + 1))
        else:
            c.add([gates.X(j) for j in list(combi[i])])
            if p == 0:
                c.add(gates.X(controls[i]))
                c.add(gates.X(N + 1))
            c.add(gates.Z(N + M + 5).controlled_by(controls[i] , N + 1, N + M + 3, N + M + 4))
            if p == 0:
                c.add(gates.X(controls[i]))
                c.add(gates.X(N + 1))
            c.add([gates.X(j) for j in list(combi[i])])
    if equal is True:
        diffusion(N, M, s)
    return c
```

Tiếp theo ta dùng các lệch code sử dụng để tạo một danh sách các chữ số nhị phân từ một số kết quả thập phân là trả về một danh sách chứa các chữ số nhị phân tương ứng.

```
def bin_list(dec,nqubits):
    binary = format(dec,'0{}b'.format(nqubits))
    return list(map(int, str(binary)))
```

Tiếp theo sử dụng code có khả năng sử dụng trọng số để chuyển đổi số nhị phân thành số thập phân. Hàm trả về một số nguyên là số thập phân tương ứng với số nhị phân được đưa vào.

```
def bin_to_dec(binary_number):
    decimal_number = 0
    for position, string_digit in enumerate(binary_number[::-1]):
        decimal_number += int(string_digit) * 2 ** position
    return decimal_number
```

Tiếp theo là đoạn code được sử dụng để tìm kiếm một mẫu trong một chuỗi. Mã cũng tính toán vị trí của mẫu trong chuỗi bằng cách tìm giá trị lớn nhất của xác suất và hiển thị kết quả như Hình 6 (với chuỗi $N = 8$ và mẫu $M = 2$). Kết quả cho thấy trạng thái $|010\rangle$ với xác suất cao nhất 0,15011.

```
string = input("Given string: " )
pattern = input("Pattern:  " )
string_w  = [int(x) for x in str(string)]
pattern_p   = [int(x) for x in str(pattern)]
n_shots = int(input("Number of shots: " ))
N = len(string_w)
M = len(pattern_p)
s = int(m.ceil(m.log2(N - M)))*M
c = Circuit(N + M + s)
p = pattern
initialize( N, M, s, string_w, pattern_p)
if pattern_p[0] == pattern_p[1]:
      oracle(N,M,s,p,True)
else:
    for p in pattern_p:
        oracle(N,M,s,p,False)
        diffusion(N,M,s)
c.add(gates.M(*range(N + M, N + M + int(s/2)), register_name="A"))
result = c(nshots = n_shots)
y = result.frequencies(binary=True, registers=True)
print('State', '\t\tOccurance','\tProbability')
suma =0
for q_state in y['A']:
    prob = y['A'][q_state]/100000.
    print(q_state, '\t\t', y['A'][q_state],'\t\t',prob)
print()
max_key = max(y['A'].items(), key = operator.itemgetter(1))[0]
print("The pattern has been found at index i =",bin_to_dec(max_key))
```

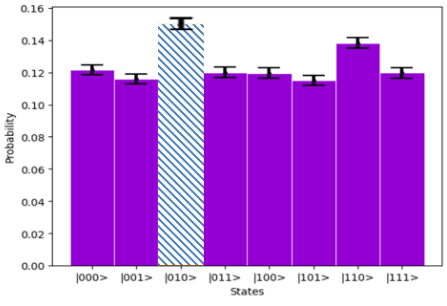
```
Given string:  00110001
Pattern:   11
Number of shots:  100000

[Qibo 0.2.4|INFO|2024-02-29 15:04:47]: Using numpy backend on /CPU:0

State       Occurance   Probability
000          12149       0.12149
001          11591       0.11591
010          15011       0.15011
011          11996       0.11996
100          11962       0.11962
101          11495       0.11495
110          13826       0.13826
111          11970       0.1197

The pattern has been found at index i = 2
```

Hình 6. *Kết quả ở dạng xác suất của các trạng thái*



Hình 7. *Kết quả hiển thị dạng biểu đồ*

Để tường minh hơn, chúng tôi code vẽ biểu đồ và kết quả được làm nổi bật bằng bằng nền gạch chéo như ở Hình 7: trạng thái cần tìm là $|010\rangle$ với xác suất cao nhất 0.15011.

Sau khi thực hiện thuật toán ta thu được là trạng thái $|010\rangle$ tương ứng với trạng thái $|2,3\rangle$ - mẫu "11" khớp với chuỗi "00110001" tại vị trí $i = 2$. Tổng số qubits sử dụng là 16 và *nshots* = 100000. Kết quả này rõ ràng hơn nhờ việc chèn thêm thư viện Qibo vì không có sẵn trên nền tảng Qiskit.

4. Kết luận

Thuật toán Grover áp dụng tính chất của cơ học lượng tử nên tốc độ xử lý tăng bậc hai so với giải pháp cổ điển. Chúng tôi đã tiến hành ứng dụng GA vào việc giải trình tự DNA, trong khi thuật toán cổ điển sử dụng ít nhất $N + M$ truy vấn, thì vị trí chỉ mục đã được tìm thấy chỉ với một/hai truy vấn khi sử dụng GA. Điều này cho thấy sự ưu việt về thời gian chạy và lợi thế của việc sử dụng thuật toán lượng tử so với thuật toán cổ điển. Việc tăng tốc bậc hai của GA vẫn còn khiêm tốn với máy tính lượng tử hiện nay, nó càng tối ưu khi máy tính lượng tử có số qubit lớn. Trong tương lai nhóm sẽ tiếp tục triển khai vấn đề giải trình tự DNA thông qua GA với số qubit cao hơn để xử lý chuỗi có độ dài lớn cũng như của mẫu phức tạp hơn.

TÀI LIỆU THAM KHẢO/ REFERENCES

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* (STOC), May 1996, pp. 212-219, doi: 10.48550/arXiv.quant-ph/9605043.

[2] G. Nannicini, "An introduction to quantum computing, without the physics," *SIAM Review*, vol. 62, no. 4, pp. 936-981, 2020, doi: 10.48550/arXiv.1708.03684.

[3] Z. Zhang, *et al.*, "Grover-QAOA for 3-SAT: Quadratic speedup, fair-sampling, and parameter clustering," *arXiv preprint*, 2024, doi: 10.48550/arXiv.2402.02585.

[4] S. Aaronson, D. Grier, and L. Schaeffer, "A quantum query complexity trichotomy for regular languages," *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, Baltimore, MD, USA, 2019, pp. 942-965, doi: 10.48550/arXiv.1812.04219.

[5] M. R. Habibi, S. Golestan, A. Soltanmanesh, J. M. Guerrero, and J. C. Vasquez, "Power energy applications based on quantum computing: The possible potentials of grover's algorithm," *Electronics*, vol. 11, no. 18, 2022, doi: 10.3390/electronics11182919.

[6] B. Khanal, P. Rivas, J. Orduz, and A. Zhakubayev, "Quantum machine learning: A case study of grover's algorithm," *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2021, pp. 79-84, doi: 10.1109/CSCI54926.2021.00088.

[7] D. Qiu, L.Luo, and L. Xiao, "Distributed Grover's algorithm," *Theoretical Computer Science*, vol. 993, p. 114461, 2024, doi: 10.48550/arXiv.2204.10487.

[8] R. H. Preston, "Applying Grover's algorithm to hash functions: a software perspective," *IEEE Transactions on Quantum Engineering*, vol. 3, no. 2500710, pp. 1-10, 2022, doi: 10.1109/TQE.2022.3233526.

[9] S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia, "Implementing Grover oracles for quantum key search on AES and LowMC," *Proc. 39th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2020, pp. 280–310, doi: 10.48550/arXiv.1910.01700.

[10] J. M. Heather and B. Chain, "The sequence of sequencers: The history of sequencing DNA," *Genomics*, vol. 107, no.1, pp. 1-8, 2016, doi: 10.1016/j.ygeno.2015.11.003.

[11] R. Marina and C. Baldo III, "Quantum DNA sequencing using Gaussian amplitude amplification," *arXiv preprint*, 31 Oct. 2023, doi: 10.48550/arXiv.2310.20466.

[12] A. Sarkar, Z. Al-Ars, C. G. Almudever, and K. L. Bertels, "QiBAM: approximate sub-string index search on quantum accelerators applied to DNA read alignment," *Electronics*, vol. 10, no. 19, p. 2433, 2021, doi: 10.3390/electronics10192433.

[13] J. Leng, F. Yang, and X.B. Wang, "Quantum advantage of noisy Grover's algorithm," *arXiv preprint*, 19 Jun. 2023, doi: 10.48550/arXiv.2306.10855.

[14] J. Barberà, "Quantum string matching," *github.com*, May 26, 2022. [Online]. Available: https://github.com/juliabarbera/TFG-quantum-string-matching-/blob/main/Quantum_string_matching.Ipynb. [Accessed January 15, 2023].