UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

SUPREMACÍA CUÁNTICA: ¿EL FIN DE LA SEGURIDAD CLÁSICA?

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Garrigós Candela, Elena

Tutor/a: Andrés Martínez, David de

Cotutor/a: Escobar Román, Santiago

CURSO ACADÉMICO: 2021/2022

Agradecimientos

A mis padres, por haber hecho posible que haya aprendido tanto en este grado. A veces no soy consciente de lo afortunada que soy de teneros.

A Rosa, por ser paciente y un apoyo en el proceso de desarrollo de este trabajo. Por aguantarme en todo tipo de situaciones y ser la fuente de distracción necesaria para respirar hondo y proseguir. Y a mis amigas, entre todas somos cada día más fuertes.

Y por último, me gustaría agradecer a mis tutores en este trabajo, David de Andrés y Santiago Escobar, por vivir todo este largo proceso conmigo y haber confiado en mi idea desde el primer minuto. Por toda la profesionalidad y la tranquilidad que me habéis mostrado. Con vosotros me he sentido muy cómoda.

Resum

La computació quàntica pretén donar un gir a tota la computació tal i como la coneixem. En els últims anys s'han realitzat grans avanços que inclús han aconseguit el desenvolupament de llenguatges d'alt nivell per a aquestes revolucionàries computadores. Aquestes revolucionàries arquitectures faciliten la solució de determinats tipus de problemes com la factorització de nombres enters, el logaritme discret o el logaritme discret de curva elíptica, entre altres, que són la base dels algoritmes criptogràfics actuals. Llavors, a pesar de què encara no existeixen computadores quàntiques tan potents com per a trencar les claus generades pels algoritmes criptogràfics actuals, s'està treballant ja en la criptografia postquàntica per a assegurar que els futurs algoritmes criptogràfics siguen robustos per afrontar possibles atacs quàntics en el futur.

D'altra banda, hi ha una forta postura que assegura que mai s'aconseguirà construir una computadora quàntica competent. No obstant, la major part del esforç dedicat a la computació quàntica s'invirteix en desenvolupar mètodes de tolerància quàntica a fallades.

En aquest treball es donarà a conèixer les bases de la computació quàntica, es presentaran diversos llenguatges quàntics d'alt nivell i les plataformes de simul·lació existents, i es realitzaran experiments per a determinar la capacitat de les computadores quàntiques actuals per a trencar l'encriptació RSA mitjançant l'algoritme quàntic de factorització Shor, analitzant així quánt prop- o lluny -queda la revolució computacional en termes de seguretat, i debatint -en contraposició- si en volta de parlar de supremacia quàntica hauríem de parlar d'utopia quàntica.

Paraules clau: Física quàntica, qubits, RSA SHA, Algoritme Shor, llenguatges quàntics, circuïts, simul·lador, computadora quàntica, entrellaçament, coherència quàntica, Qiskit

---

Resumen

La computación cuántica pretende dar un giro a toda la computación tal y como la conocemos. En los últimos años se han realizado grandes avances que incluso han llevado al desarrollo de lenguajes de alto nivel para estas revolucionarias computadoras. Estas revolucionarias arquitecturas permitirán la solución de determinados tipos de problemas como la factorización de números enteros, el logaritmo discreto o el logaritmo discreto de curva elíptica, entre otros, que son la base de los algoritmos criptográficos actuales. Por ello, a pesar de que no existen todavía computadoras cuánticas lo suficientemente potentes como para romper las claves generadas por los algoritmos criptográficos actuales, se está trabajando ya en la criptografía post-cuántica para asegurar que los futuros algoritmos criptográficos sean robustos frente a posibles ataques cuánticos en el futuro.

Por otro lado, hay una fuerte postura que asegura que nunca se va a llegar a construir una computadora cuántica competente. Sin embargo, la mayor parte del esfuerzo dedicado a la computación cuántica se invierte en desarrollar métodos de tolerancia cuántica a fallos.

En este trabajo se dará a conocer las bases de la computación cuántica, se presentarán diversos lenguajes cuánticos de alto nivel y las plataformas de simulación existentes, y se realizarán experimentos para determinar la capacidad de los computadores cuánticos actuales para romper la encriptación RSA mediante el algoritmo cuántico de factorización Shor, analizando así cuán cerca -o lejos- queda la revolución computacional en temas de seguridad, y debatiendo, en contraposición, si en vez de hablar de supremacía cuántica debemos hablar de utopía cuántica.

Palabras clave: Física cuántica, qubits, RSA, SHA, Algoritmo Shor, lenguajes cuánticos, circuitos, simulador, computadora cuántica, entrelazamiento, coherencia cuántica, Qiskit

---

Abstract

Quantum computing aims to give a turn to all computing as we know it. In recent years, great advances have been made that have even led to the development of high-level languages for these revolutionary computers. These revolutionary architectures will allow the solution of certain types of problems such as the factorization of integers, the discrete logarithm or the discrete logarithm of elliptic curve, among others, which are the basis of current cryptographic algorithms. Therefore, despite the fact that there are still no quantum computers powerful enough to break the keys generated by current cryptographic algorithms, work is already under way on post-quantum cryptography to ensure that future cryptographic algorithms are robust against possible quantum attacks in the future.

On the other hand, there is a strong position that ensures that a competent quantum computer will never be built. However, most of the effort dedicated to quantum computing is invested in developing quantum fault-tolerance methods.

In this work, the bases of quantum computing, various high-level quantum languages and existing simulation platforms will be presented, and experiments will be carried out to determine the ability of current quantum computers to break RSA encryption using the quantum factorization Shor's algorithm, thus analyzing how close - or far - is the computational revolution in terms of security, and debating, in contrast, if instead of talking about quantum supremacy we should talk about quantum utopia.

Key words: Quantum physics, qubits, RSA, SHA, Shor's algorithm, quantum languajes, circuits, simulator, quantum computer, entanglement, quantum coherence, Qiskit

---

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

1.1 Motivación

Hoy en día, el mundo de la informática nos rodea en todos los ámbitos de la vida. A lo largo de los años, lo que comenzó siendo una máquina para realizar cálculos en un laboratorio ha pasado a formar parte del día a día de la sociedad. Primero la industria empezó a beneficiarse de los ordenadores, con la automatización de la administración de recursos. Más tarde pusieron esta tecnología al alcance de la gente de a pie con los primeros ordenadores personales -PCs-. Lo que supuso la creación de videojuegos y aplicaciones de escritorio, que hizo que la gente introdujese el uso del mundo digital en sus vidas. Con el nacimiento de Internet se dio la oportunidad de construir un mundo comunicado y sin limitaciones físicas, que ha hecho posible el avance sociocultural y tecnológico. Desde entonces, se han desarrollado infraestructuras computacionales y dispositivos de todo tipo repartidos por todo el mundo para ofrecer servicios fiables y potentes. Mensajería, redes sociales, pero también sistemas críticos como la banca, aeroespaciales, transporte o energía. Toda la información se mantiene incorrupta gracias a la ciberseguridad y las técnicas que protegen los datos, como la criptografía, que se aplican tanto en la comunicación como en el almacenamiento de datos. Si no se encriptasen los datos, cualquiera podría leer y alterarlos a su antojo.

En la búsqueda de sobrepasar las limitaciones que las tecnologías y arquitecturas de cómputo actual presentan, la computación cuántica aparece como una opción para la resolución de problemas complejos que no son abordables en el paradigma actual. Entre estos problemas aparece la descomposición de números en factores primos, que es la base para los sistemas de cifrado actuales, ya que un ordenador convencional no puede resolver este cálculo en un tiempo razonable para el ser humano. No obstante, la tecnología cuántica sí. Esto traza una frontera denominada supremacía cuántica. Que acota la potencia de los ordenadores actuales, quedando inferiorizados frente a cómputo de esta nueva tecnología.

La motivación de este trabajo es la de obtener una noción más acertada acerca de la posibilidad de que se produzca un cambio realmente importante en toda la informática que nos rodea, con el desarrollo de la computación cuántica -y se consiga esta supremacía-, analizar cómo afecta este cambio a la seguridad implantada en la actualidad y profundizar en el conocimiento de esta nueva tecnología que tantas cosas tiene por ofrecer.

1.2 Objetivos

El objetivo de este proyecto es un estudio teórico, práctico y pragmático sobre la computación cuántica y cómo de lejos -o cerca- nos encontramos de alcanzar la

supremacía cuántica, basándonos en la factorización de números enteros, y conocer las consecuencias de esta supremacía. Concretamente, los objetivos específicos son:

- Hacer un recorrido por el modelo teórico y el desarrollo que se ha llevado a cabo sobre la computación cuántica, así como entender el nuevo modelo de programación.

- Comprender la amenaza que supone la potencia de la computación cuántica para la criptografía clásica actual con el algoritmo de factorización de Shor, y comentar la relación que conlleva esto con la supremacía cuántica.

- Comparar los resultados obtenidos de varias ejecuciones del algoritmo mencionado con la realidad de la criptografía clásica actual y discernir acerca de la posibilidad de alcanzar la supremacía cuántica.

1.3 Estructura de la memoria

A continuación se presenta la estructura de la memoria:

El capítulo 1 introduce la memoria a través de la motivación del trabajo y los objetivos a cumplir.

En el capítulo 2 se hace una introducción a los conceptos básicos de este campo de la computación, su funcionamiento y las limitaciones que presenta.

El capítulo 3 presentará brevemente el estado del arte asociado a la tecnología cuántica, explicando cuáles son las empresas punteras en esta área de investigación, las herramientas que ofrecen y los *softwares* más potentes actuales.

En el capítulo 4 se detalla el proceso de desarrollo del experimento, los detalles de implementación del código utilizado para la ejecución de los algoritmos de Deustch-Jozsa, Grover y Shor, y las limitaciones que se han encontrado durante el proceso.

El último capítulo presenta las conclusiones del trabajo respecto a los objetivos establecidos inicialmente, y se determina cuál es el límite que presenta esta computación, así como el potencial que tiene.

CAPÍTULO 2

Computación cuántica

2.1 ¿Qué es la computación cuántica?

La computación cuántica [1] es el paradigma de computación que se cimienta en las leyes de la mecánica cuántica para procesar la información.

En los procesadores convencionales, la unidad mínima de información que se procesa es el bit, y su estado puede tomar los valores 0 o 1, de forma totalmente excluyente. Sin embargo, en la computación cuántica necesitamos una nueva unidad de información que se adecue a la mecánica cuántica: el bit cuántico o cúbit.

El **cúbit** es el concepto lógico de unidad de información que se implementa sobre sistemas cuánticos. Es un bit que puede tomar valor 0 o 1, pero que es capaz de trabajar con un porcentaje de 0 y un porcentaje de 1 simultáneamente hasta que se quiere leer su valor. El hecho de que se asocien probabilidades a cada posible valor se debe a que mientras que el cúbit no se accede en lectura, no posee un estado definido. Es decir, es una combinación de ambos estados, que se conoce como **superposición de estados o estado cuántico**. En el momento que se quiere leer el estado del cúbit, tomará un valor u otro, se dice que los estados se colapsan en uno solo. Y será un valor u otro dependiendo de las probabilidades de cada estado.

Por tanto, la motivación de este paradigma computacional es poder manipular los cúbits en estados superpuestos para obtener un estado resultante que se pueda leer tantas veces como queramos porque sus probabilidades siempre harán que colapse al mismo valor. Sin embargo, si las probabilidades no son claramente favorables para uno de los estados, y hay un 40% de probabilidad para el estado 0 y un 60% para el estado 1, si medimos el estado 100 veces, 40 de ellas veremos que el cúbit tiene valor 0 y 60 veces observaremos que tiene valor 1.

Cuando el sistema conste de n cúbits, se manipularán 2^n posibles estados. Si tras las ejecución queremos leer el estado de los n cúbits, entonces los 2^n posibles estados para esos n cúbits colapsarán en uno solo.

Análogamente a las puertas lógicas bien conocidas que realizan operaciones lógicas simples sobre una serie de bits de entrada, produciendo una función combinacional a la salida, en este paradigma se presentan las puertas lógicas cuánticas de uno o varios cúbits, que operan produciendo una transformación sobre los cúbits de entrada.

Se llaman registros cuánticos a las entradas que reciben las puertas lógicas cuánticas, que almacenan el estado de un cúbit, ya sea en superposición o determinado.

La encadenación de una serie finita de puertas cuánticas se denomina circuito cuántico y, tras aplicar las transformaciones de cada puerta sobre los registros de entrada, el resultado es un nuevo estado cuántico que colapsará cuando lo observemos. Si el circuito se ha construido adecuadamente, el colapso de la salida

se producirá con una probabilidad muy alta para la salida correcta y esta será el valor que tomará el cúbit.

Definimos como **ordenador cuántico** al conjunto de unos registros de entrada, un circuito cuántico -que aplica transformaciones sobre los estados cuánticos de los registros- y la lectura del estado de salida.

2.2 Fundamentos teóricos de la computación cuántica

En 1900, Max Planck [10] introdujo la idea de que la materia absorbía y emitía luz en forma de paquetes de energía y no en forma de onda, como se había considerado hasta entonces. Planck denominó a estos paquetes *cuantos de energía*, y tras una serie de experimentos observó que su emisión siempre tomaba valores discretos múltiplos de un valor concreto (6.63×10^{-34}) en lugar de la tendencia a infinito que se esperaba, y definió este valor como *cuanto de acción* o *h*. El descubrimiento de esta constante fue el comienzo de la mecánica cuántica y todo un giro en los conocimientos físicos a escala atómica y sub-atómica.

Cinco años más tarde, Albert Einstein [14] explicó el efecto fotoeléctrico respaldándose en la publicación de Planck, y propuso que la luz estaba formada por partículas de luz a las que llamó fotones. Y fue en 1909 cuando introdujo la idea de dualidad onda-partícula y apostó por afirmar que la luz era, a su vez, una onda y una partícula. Pero explicar esto conceptualmente resultó ser complicado.

Erwin Schrödinger [15] descubrió en 1925 que a cada punto del espacio se le podía asociar un número complejo, que podríamos entender como un vector espacial que tiene un tamaño y una dirección determinada en un momento temporal. Tras numerosos experimentos comprobando de qué manera influía este vector espacial sobre una serie de electrones que lanzaba contra una rendija y que la atravesaban situándose en diferentes puntos del espacio, se dio cuenta de que este valor correspondía a la probabilidad de encontrar un electrón en cada punto. Y definió a todo este conjunto de números como la función de onda de una región espacial, tomando como referencia el carácter oscilador de las ondas, puesto que estos vectores oscilaban constantemente entre varios valores. Esta función de onda fue un elemento clave para que Heisenberg entendiera el indeterminismo que presentan las partículas cuánticas en algunas de sus propiedades, las cuales no se pueden predecir con exactitud hasta el momento de su medición. Como es el caso de la posición y el momento[1] de un átomo en un instante de tiempo.

2.2.1 Propiedades de la mecánica cuántica involucradas

Este nuevo paradigma de computación no se rige por las propiedades eléctricas de la física determinista, sino que se fundamenta en el nuevo comportamiento de los átomos que se comportan de forma impredecible para la concepción determinista de la física. Estas nuevas máquinas aprovechan las características que hicieron que la mecánica cuántica diese un giro a toda la física de partículas.

Las propiedades que se han utilizado para la creación de la computación cuántica son:

- La **superposición** de estados es la característica fundamental de las partículas cuánticas que permite que tengan varios valores para una propiedad simultáneamente. Esto se debe a la función de onda del espacio que actúa de oscilador para los valores del átomo. Al medir -saber su valor de- una propiedad en un instante de tiempo no vamos a poder observar todos los posibles valores sobre los que oscila, sino que se produce un colapso de la función de onda y se reduce a un solo valor de los posibles. Es lo que sucede con los cúbits en superposición, que poseen ambos estados 0 y 1 a la vez, pero

[1] El momento de una partícula se conoce como el par de propiedades velocidad y dirección.

que cuando queramos saber su valor tras la ejecución solo vamos a ser capaces de percibir o bien 0, o bien 1. La propiedad que decide qué valor toma un cúbit se llama espín[2]. Para entender su valor debemos concebir los electrones como esferas, y esta propiedad se define sobre el eje vertical, siendo (0) el valor positivo del espín y (1) el valor negativo. Para tener una visión más clara de la representación que se hace de un cúbit se utilizan las llamadas esferas de Bloch, sobre las que se expresa el vector que representa la superposición de los estados binarios que puede tomar. Los estados bien definidos se encuentran en el norte y en el sur de la esfera.

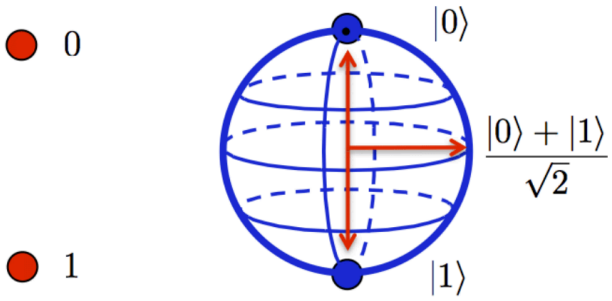


Figura 2.1: Esfera de Bloch que representa los posibles estados de un cúbit [2].

- El **entrelazamiento** de partículas, y en este caso de cúbits, es un fenómeno de la mecánica cuántica que relaciona dos partículas de forma que son totalmente dependientes entre sí. Si una se modifica, la otra también lo hace instantáneamente, aunque se encuentren físicamente separadas sin ningún tipo de canal de comunicación que les permita conocer el estado de la otra [5] .Para entender mejor este concepto, vamos a hacer uso de un ejemplo simple y poco realista, pero que ejemplifica bien lo que sucede entre las partículas. Supongamos que Alice y Bob van a recibir un regalo cada uno, ambos empaquetados en cajas idénticas. Saben que una caja contiene un regalo de Tipo1 y la otra contiene un regalo de Tipo2 necesariamente. Escogen una caja y deciden abrirla cada uno en su casa. Hasta este momento, ninguno de los dos sabe qué contiene cada caja, y según la mecánica cuántica, el contenido de ambas cajas está indeterminado[3]. Ninguna de las dos cajas contiene ni el Tipo1 ni el Tipo2, sino que ambos regalos son equiprobables de encontrarse en ambas cajas. Cuando Alice llega a casa y decide abrir su regalo, encuentra que su caja contiene el regalo de Tipo2. Y automáticamente sabe que la caja de Bob contendrá el Tipo1, sin necesidad de observar la otra caja, ni que le llame Bob para comunicárselo. A partir de este momento, el contenido de ambas cajas ha pasado a estar determinado, pero además lo está de una forma especial, porque necesariamente el estado de uno quedará definido por el estado del otro. Este comportamiento es el que exhiben dos cúbits -o más- que se encuentran entrelazados. El estado de todos estos cúbits

[2] El espín es una propiedad física que hace referencia al giro sobre sí mismo que posee un electrón. Sus posibles valores son +-1/2.

[3] Otra concepción de este caso sería que el contenido de las cajas sí se encuentra determinado, pero ni Alice ni Bob están teniendo en cuenta -o no perciben- la variable que decide qué contiene cada caja. No saben qué determina qué contiene cada caja. Se conoce como teoría de las variables ocultas.

debe entenderse como un estado global, del que no podemos determinar el valor de ninguno de ellos por separado hasta que quede determinado algún cúbit. En el momento que se lea el valor de un cúbit y se determine su estado, automáticamente el resto de cúbits entrelazados también perderán su estado cuántico y quedarán determinados a uno de los posibles estados. Gracias a este fenómeno, dos cúbits que se encuentran físicamente separados son capaces de intercambiar información. Porque en el momento que se determine el estado de uno de ellos, el otro tomará el estado complementario instantáneamente [14]. En computación cuántica, cuando se aplica este concepto a un par de cúbits del circuito que estemos ejecutando, en el momento que se determine que uno de los dos cúbits tiene cierto estado, necesariamente el otro cúbit poseerá el estado contrario.

Paralelismo cuántico

Gracias a la superposición que presentan los cúbits, en el cálculo computacional se produce este fenómeno, que permite evaluar todas las combinaciones de estados de n cúbits de entrada simultáneamente, y manipular las 2^n combinaciones de estados mediante las puertas cuánticas a la vez. Así que la cantidad de información que podemos manipular con un número dado de cúbits es mucho mayor que con la misma cantidad de bits.

Además de estas propiedades cuánticas, es necesario definir qué es la **coherencia cuántica**. Los objetos o materiales que se encuentran en estado de coherencia son aquellos que se mantienen en una superposición de estados o entrelazamiento. Es decir, decimos que un cúbit mantiene la coherencia cuántica cuando no tiene un valor determinado.

A partir del concepto anterior cabe definir, en contraposición, la **decoherencia cuántica.** Este término hace referencia al hecho de que un objeto que está en un estado coherente deje de estarlo y pase a un estado físico clásico. Cuando esto sucede, el objeto dejar de comportarse bajo los efectos cuánticos y pasa a tener un carácter determinista. Las causas de este fenómeno son que, o bien el entorno influye sobre el estado del sistema y no permite conservarlo durante mucho tiempo, o bien que la propia naturaleza del sistema cuántico pierde su sentido al aumentar de tamaño, ya que para sistemas macroscópicos el tiempo de coherencia cuántica es prácticamente nulo. Es por este motivo que en nuestra escala macroscópica de percepción no podamos observar los efectos cuánticos que nos rodean. Para el caso de los cúbits, cuando uno de ellos sufre este hecho, automáticamente pierde su indeterminismo pasando a ser un bit clásico con un valor bien definido, normalmente cero. Y hasta que no reiniciemos el cúbit para una nueva ejecución este ya no volverá a poseer su carácter cuántico.

2.2.2 Modelo matemático: Cúbits

Para definir con mayor precisión los conceptos cuánticos que vamos a tratar en este trabajo, es necesario formalizarlos con la representación matemática correspondiente.

La computación cuántica se expresa en términos matemáticos de espacios vectoriales del álgebra lineal. Los estados cuánticos y las operaciones correspondientes a las puertas cuánticas se describen en notación *bra-ket*[4] [11, 23].

En primer lugar, los valores binarios -$|0\rangle$ y $|1\rangle$ que puede tomar un cúbit bien definido -en estado clásico- se representan por los *kets* $|0\rangle$:=[0 1] y $|1\rangle$:=[1 0], respectivamente. Ambos vectores forman una base canónica[5] en C^2 y se les denomina estados base. Nótese que cada vector tiene un 1 en la posición del vector correspondiente al estado que representa. El *ket* $|0\rangle$ tiene un 1 en la posición 0, y el *ket* $|1\rangle$, en la posición 1.

Cuando el cúbit se encuentra en estado de superposición, su vector de estado se puede representar como una combinación lineal de los estados base.

$$a_0 * |0\rangle + a_1 * |1\rangle$$

A los coeficientes de cada vector se les llama amplitudes, y corresponden a la probabilidad asociada a cada estado para que, cuando leamos el cúbit, sea el estado que observemos.

Para un sistema formado por n cúbits, el estado de superposición del sistema estará formado por las posibles combinaciones que se pueden dar con dichos cúbits, es decir, 2^n estados base. La expresión que representará el sistema en estado de superposición será:

$$a_0 * (0\ldots0) + a_1 * (0\ldots1) + \ldots. + a_{2^n-1} * (1\ldots1)$$

Por ejemplo, si consideramos un sistema con 2 cúbits, los estados base que podrá tener el sistema serán: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$. Por tanto, el estado de superposición para este sistema se representará con la siguiente expresión:

$$a_0 * |00\rangle + a_1 * |01\rangle + a_2 * |10\rangle + a_3 * |11\rangle$$

Un estado en superposición debe cumplir con la siguiente restricción:

$$a_0, a_1, \ldots, a_{2^n-1} \textit{ deben cumplir que } |a_0|^2 + |a1|^2 + \ldots + |a_{2^n-1}|^2 = 1$$

Si esto no se cumple, entonces no es un estado en superposición válido, ya que todos los estados superpuestos son una distribución de probabilidad y deben normalizar su módulo.

Se distinguen tres tipos de estados para los cúbits:

- El **estado básico** o clásico, que es el que tiene bien definido un solo valor para su estado, no hay combinación de valores. Por tanto, en su expresión como combinación lineal de los vectores de la base, solo encontraremos una amplitud con valor diferente a cero.

---

[4] La notación *bra-ket* es un formalismo matemático creado por Paul Dirac a mediados del siglo XX que recibe su nombre del producto escalar o interno de dos vectores que es denotado por el "paréntesis angular", llamado "*angle braket*" en inglés.

[5] Una base canónica está formada por vectores espaciales que son: i) ortogonales -perpendiculares- entre sí, ii) orto-normales, es decir, sus módulos se han reducido a la unidad. Se dice que es canónica por ser la más simple para representar las rectas, los planos y los espacios. Una base canónica de C^2 sirve de ejes de referencia para un espacio bidimiensional, que contiene rectas y planos.

$$0^* |00\rangle + 1^* |01\rangle + 0^* |10\rangle + 0^* |11\rangle$$

La anterior combinación lineal representa el estado básico $|01\rangle$ con probabilidad 1, es decir, este estado siempre va a leer en este mismo valor.

- El **estado producto**, que es aquel estado cuántico en superposición de un conjunto de cúbits que se puede expresar como el producto tensorial de los estados de sus componentes. Por lo que representará la combinación de varios posibles estados cada uno con una probabilidad asociada de que sea el estado que se lea.

$$0'5^* |00\rangle + 0'5^* |01\rangle + 0'5^* |10\rangle + 0'5^* |11\rangle$$

El estado anterior representa una superposición equiprobable de los cuatro posibles estados que se puede obtener con dos cúbits. Además, cumple con la restricción de las amplitudes para que sea un estado válido.

- El **estado en entrelazamiento**, en contraposición al anterior, cuando el estado cuántico de un conjunto de cúbits no puede expresarse como el producto tensorial de sus estados componentes.

2.2.3 Puertas cuánticas

Por otro lado, para realizar operaciones sobre un conjunto de cúbits y construir circuitos tenemos las puertas cuánticas, que forman un circuito encadenándose una detrás de otra, igual que los circuitos clásicos.

Para representar los circuitos formados, se dibuja una especie de pentagrama, donde cada línea que se añade es un cúbit que se va a utilizar en el circuito. Las líneas representan la evolución temporal de izquierda a derecha de los cúbits, y sobre esas líneas se colocan las puertas cuánticas que realizan las operaciones.

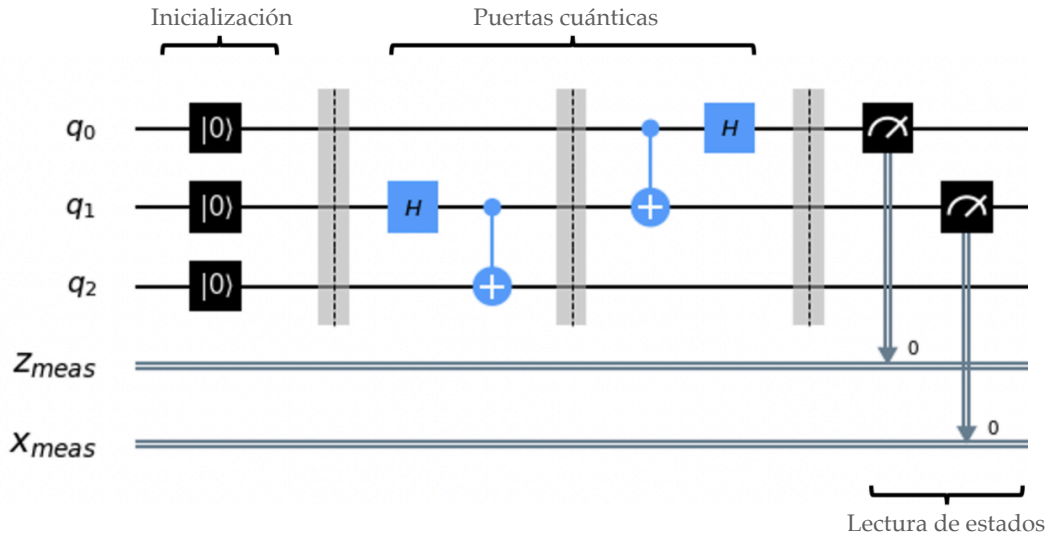


Figura 2.2: Representación de un circuito cuántico.

En la imagen se puede ver que aparecen dos tipos diferentes de líneas: la línea simple y la doble línea. La primera es la que representa la evolución temporal de los cúbits, mientras que la segunda sirve para representar partes de la ejecución que no tienen carácter cuántico, como es el caso de *Zmeas* y *Xmeas* que son bits clásicos que se usan para leer los valores de salida.

Podemos distinguir tres partes del circuito: El inicio, donde se inicializan todos los cúbits a un mismo estado base, la parte intermedia donde se colocan las puertas cuánticas que van a realizar los cálculos del algoritmo, y el final, que suele darse por la lectura de los estados de los cúbits. Como se puede observar, se realiza un volcado de los estados de q_0 y q_1 sobre los bits clásicos *Zmeas* y *Xmeas*.

Matemáticamente, las puertas cuánticas se representan con matrices de dimensión $2^n \times 2^n$ que actúan modificando los valores de los vectores de estados de los cúbits de entrada. En concreto, estas matrices deben realizar transformaciones unitarias y lineales, para que cuando actúe sobre un estado en superposición el resultado sea otro estado en superposición que cumple con la restricción de que su producto interno debe ser unitario, es decir, la suma de los coeficientes al cuadrado debe ser uno [7]. Por tanto, como las puertas cuánticas son matrices unitarias, todas son reversibles. Gracias a esta **reversibilidad**, podemos obtener los valores de entrada a partir del estado resultante aplicando la matriz inversa correspondiente. Esto proporciona una habilidad a los circuitos cuánticos que no siempre podemos obtener de los clásicos. Como es el caso de la puerta clásica constante, que para cualquier entrada la salida es cero -o uno para su complementaria-, no podemos saber cuál era el valor de entrada de la puerta.

El hecho de que estas nuevas puertas tengan carácter reversible las condiciona a que el número de cúbits entrada siempre haya de ser equivalente al número de cúbits salida. En caso contrario se estaría perdiendo o replicando información. Debido a esto, en los circuitos cuánticos no se añaden cúbits en tiempo de ejecución [4].

Actualmente existe una gran variedad de puertas cuánticas de uno, dos o más cúbits. Cada una de ellas manipula ciertas propiedades de los cúbits para modificar sus amplitudes del estado. A continuación se presentan las puertas más conocidas e importantes [8].

Puerta Identidad 

La más simple de todas que actúa sobre un solo cúbit. Su comportamiento es el de devolver como salida el estado de entrada.

Para obtener la matriz de una puerta, se debe hacer la "tabla de verdad" de forma parecida a como sucede con la puertas lógicas clásicas. Colocando en la parte superior de la tabla las combinaciones de entrada, y en la columna de la izquierda las combinaciones de salida, pondremos un 1 en la fila de salida correspondiente para cada entrada y un 0 en el resto. La tabla resultante será la matriz que hace la transformación deseada sobre los *kets* de los cúbits. Para la puerta identidad, la matriz se construirá como se muestra en la Fig. 2.3:

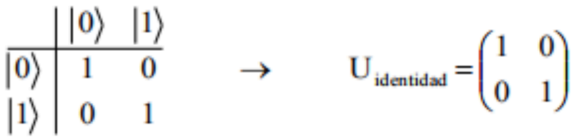


Figura 2.3: Construcción de la matriz de identidad con la "tabla de verdad".

Puerta X

Esta puerta actúa sobre un solo cúbit y es la equivalente a una puerta *NOT* clásica. La matriz correspondiente a esta puerta será:

$$\oplus = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Existen otras dos puertas similares a esta -las puertas *Y* y *Z*- que junto con la *X* forman el conjunto de **puertas de Pauli** (*X*, *Y*, *Z*). Estas tres puertas hacen una rotación de "pi radianes" sobre cada uno de los ejes de la esfera de Bloch.

La **puerta *Y*** introduce números complejos en las amplitudes.

$$\boxed{Y} = \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Por otro lado, la **puerta *Z*** hace un cambio de signo. Si el cúbit de entrada es un 0 no tiene efecto, y si la entrada es un 1 le cambia el signo y la salida es -1. Si el cúbit de entrada se encuentra en estado de superposición -$a_0^*|0\rangle + a_1^*|1\rangle$- el resultado de aplicar la puerta será $a_0^*|0\rangle - a_1^*|1\rangle$.

$$\boxed{Z} = \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Puerta de Hadamard

Esta puerta es fundamental. Al aplicarla se obtiene un estado en superposición equiprobable de los estados base, es decir, utilizando esta puerta se obtienen estados cuánticos que pueden aprovechar el paralelismo cuántico para hacer cálculos. Se suele utilizar al principio de los circuitos para dar comienzo a la ejecución cuántica. Esta puerta actúa sobre un cúbit, y hace un giro de *"pi radianes"* sobre el eje *X* y *"pi/2 radianes"* sobre el eje *Y*.

Para un cúbit, se representa con la matriz:

$$\boxed{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\hat{H}|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$\hat{H}|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Aplicando esta matriz sobre los estados base |0⟩ y |1⟩ se obtienen dos estados en superposición equiprobable.

A los estados resultantes de aplicar Hadamard se les conoce como |+⟩ y |-⟩, y si se realiza la operación inversa, es decir, se aplica Hadamard sobre estos estados en superposición, se obtienen los estados base:

$$\hat{H}\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |0\rangle \quad , \quad \hat{H}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = 1\rangle$$

Además de las puertas unitarias, otras puertas que resultan muy interesantes en el cómputo son las puertas de control -*controlled gates*-. Son puertas que requieren mínimo dos cúbits de entrada y actúan como lo hace una estructura de decisión *if… else.*

Puerta *CNOT*

La *"controlled" NOT* es la más importante de este tipo de puertas *"controlled"*. Con dos cúbits de entrada, la puerta actúa como una puerta *NOT* sobre el segundo cúbit si y solo si el estado del primer cúbit es |1⟩.

Su matriz unitaria tiene la siguiente forma:

$$= \text{CNOT} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

La representación gráfica de esta puerta se hace uniendo las líneas de ambos cúbits de entrada con una línea perpendicular como en la Fig. 2.4:

Figura 2.4: Funcionamiento y representación de una puerta *CNOT*.

La puerta *CNOT* tiene un especial interés, puesto que tiene propiedad de universalidad. Es decir, cualquier circuito cuántico se puede construir únicamente usando puertas *CNOT* y puertas de rotación -de Pauli-. Además de esto, si aplicamos una puerta *CNOT* sobre dos cúbits que se encuentran en superposición -se les ha aplicado una puerta Hadamard previamente- se obtiene un estado de entrelazamiento entre el cúbit de control y el cúbit de aplicación [32].

La estructura de puerta "*controlled*" se puede generalizar para cualquier puerta unitaria denominándose "***Controlled U***", donde la *U* equivale a la puerta unitaria que queramos controlar. La matriz se construirá de forma similar a la *CNOT*, escribiendo los valores de verdad en los cuadrantes donde el primer cúbit de entrada es 1.

$$CU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}$$

Además de estas, existe una gran variedad de puertas cuánticas que manipulan las propiedades algebraicas de la esfera de Bloch para controlar el movimiento, dirección y sentido del vector de estado de los electrones que construyen los cúbits. También existen puertas de tres cúbits que hacen una puerta con dos cúbits de control -como por ejemplo la *CCNOT* o puerta de Toffoli- , y otras que combinan algunas de las puertas básicas para hacer transformaciones más avanzadas [8] . Cabe recordar que la mayoría de las transformaciones que se realizan hacen uso de números complejos para operar sobre dichas propiedades algebraicas y no van a ser objeto de estudio en este trabajo.

Puerta QFT (Transformada cuántica de Fourier)

Es la análoga cuántica de la transformada discreta de Fourier. Matemáticamente, esta transformación genera un vector resultante a partir de aplicar la siguiente fórmula sobre un vector de entrada:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=x}^{N-1} x_j e^{2\pi i \frac{jk}{N}}$$

Al aplicar esta transformación sobre un vector de estado formado por *N* cúbits, obtenemos un cambio de estado tal que:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j w_N^{jk} = |x\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} w_N^{jk} |y\rangle$$

siendo $w_N{}^{jk} = e^{2\pi i(jk/N)}$.

Pero para este trabajo no es tan importante el entendimiento matemático como una visión general del cambio que realiza sobre los cúbits, ya que esta puerta tiene una importante aplicación en algunos algoritmos cuánticos que veremos más adelante.

Si visualizamos los estados base de un cúbit en la esfera de Bloch, los estados $|0\rangle$ y $|1\rangle$ se representan sobre el eje X. Al aplicar una QFT el estado del cúbit sufre una rotación hacia el eje Z, sobre el que se representan los estados $|+\rangle$ y $|-\rangle$, que son las bases de la superposición cuántica. El ejemplo más simple de este tipo de transformaciones es la puerta Hadamard que hemos explicado anteriormente.

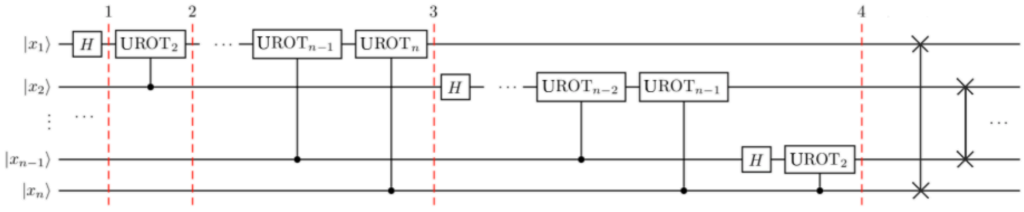


Figura 2.5: Circuito genérico de la transformación QFT para n cúbits.

A nivel general, en la Fig. 2.5 vemos la transformación QFT que se aplica para n cúbits. Primero de todo, siempre se aplican puertas Hadamard sobre todos los cúbits. De este modo ya podemos aplicar transformaciones sobre el eje Z. La puerta UROT$_i$ es la que aplica dichas transformaciones sucesivamente. UROT$_i$ equivale a una puerta RZ, que aplica una rotación sobre el eje Z de "pi/2 radianes" o "pi/4 radianes".

Vamos a ver un ejemplo concreto para entender mejor su funcionamiento. La representación en la esfera de Bloch del estado $|100\rangle$ -cúbit a cúbit- se muestra en la Fig. 2.6:

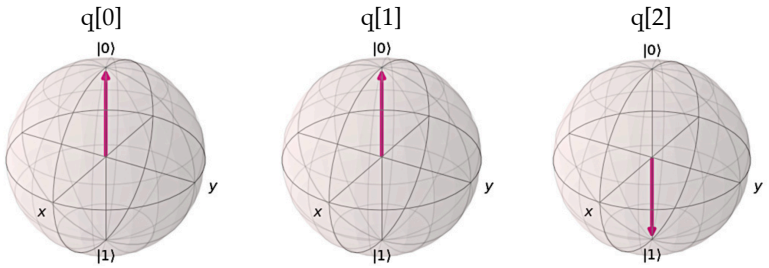


Figura 2.6: Representación cúbit a cúbit del número 4 en la esfera de Bloch.

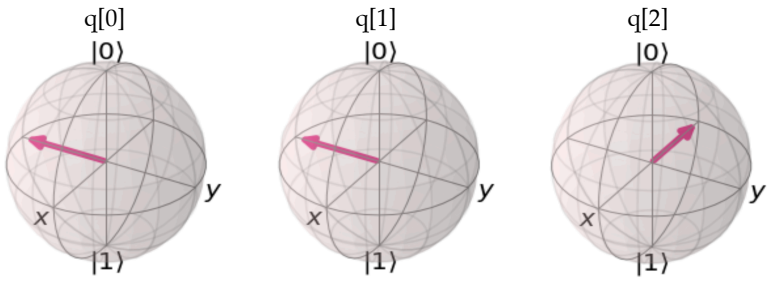


Figura 2.7: Representación cúbit a cúbit del número 4 en la esfera de Bloch sobre las bases de Fourier.

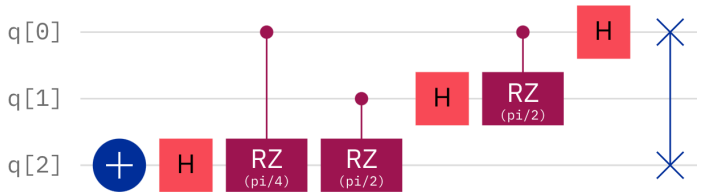


Figura 2.8: Circuito de la QFT para 3 cúbits.

Para conseguir poner los estados de los cúbits en las posiciones mostradas en la Fig. 2.7, la QFT que debemos aplicar se construye con el circuito de la Fig. 2.8, siguiendo la implementación generalizada.

Sin embargo, la utilidad que veremos que tiene este tipo de puerta en algoritmos cuánticos es la aplicación inversa de este proceso. Es lo que se conoce como IQFT -*InverseQFT*-. Que simplemente consiste en construir el circuito inverso. Principalmente se usa para obtener mediciones de los resultados de un algoritmo, es decir, para devolver los cúbits a un estado colapsado -un estado representado sobre el eje *X*-.

2.3 Complejidad computacional en la computación cuántica

La complejidad computacional es la cantidad necesaria de recursos que se utilizan para resolver un problema. Estos recursos generalmente son el tiempo para resolver un problema -pasos que realiza en el cálculo- y el espacio necesario, o cantidad de memoria utilizada.

La **Teoría de la Complejidad** [3] estudia este concepto sobre el cómputo de algoritmos para determinar su eficiencia. Se dice que un algoritmo es más eficiente que otro cuando requiere menos recursos computacionales.
Llevar a cabo este estudio sobre los algoritmos es de gran utilidad para decidir cuáles harán un cálculo deseado de forma más eficiente y aprovechando mejor los recursos. Si un algoritmo necesita mucho tiempo para realizar un cálculo no será de interés. Se dice que tiene una complejidad temporal elevada.

Además del tiempo y el espacio también se tienen en cuenta otros aspectos para decidir la complejidad, como el número de procesadores que utiliza un algoritmo paralelizado. Que es aquel que ejecuta varias partes del cálculo simultáneamente. Puede ser el caso de un bucle que ejecuta una operación sobre un conjunto de datos y el resultado de una iteración no influye en iteraciones posteriores, o bien pueden ser varias partes independientes entre sí del algoritmo. Realizar una correcta paralelización reducirá el tiempo de ejecución y la memoria necesaria.

Para decidir la complejidad de un algoritmo se toma como referencia el tamaño del problema, la cantidad de elementos a procesar, el tamaño del conjunto de elementos de entrada sobre el que se realiza el cálculo. A mayor tamaño del conjunto de entrada, mayor será la complejidad. Este conjunto de datos se utiliza como referencia para determinar la complejidad y se expresa utilizando la notación asintótica. Esta notación comúnmente utilizada considera tres posibles casos para expresar la complejidad: el ***mejor caso*** o el que menos pasos necesita, el *peor caso* o

el que más pasos realiza, y el ***caso promedio*** para la mayoría de casos. En este trabajo vamos a hacer uso de la notación asintótica del ***peor caso***, puesto que no encontraremos una complejidad mayor a la que nos indique este caso. Para expresar las funciones se utiliza la expresión $O(T)$, donde T hace referencia a la magnitud temporal en la que se incluye el problema a resolver, y se lee como "problema del orden de T".

2.3.1 Problemas P y NP

A raíz de todo el estudio realizado sobre los procesos computacionales se ha podido hacer una clasificación de los problemas en aquellos que tienen solución -*decidibles*- y aquellos que no -*indecidibles*-, siguiendo el criterio temporal [9]. Los que tienen solución constan de al menos un algoritmo que lo solucione en un tiempo finito.

Dentro del conjunto de problemas que tienen solución podemos distinguir dos sub-clases de problemas: problemas P y problemas NP [3]. Esta clasificación se rige por la condición de si el problema se soluciona en un tiempo determinístico polinómico, es decir, que a medida que aumenta el tamaño de los datos de entrada el tiempo aumenta siguiendo una proporción polinómica. Estos son los problemas P, y se consideran *tratables*. Es el caso de las búsquedas binarias, búsquedas secuenciales o las factoriales.

Los problemas NP son los que no cumplen esta condición, se consideran *intratables*, pero siguen siendo *decidibles*. La caracterización de los problemas *intratables* es que su complejidad temporal crece hasta ser de orden exponencial, y su resolución se hace compleja. Sin embargo, en esta sub-clase, verificar una solución dada resulta eficiente. Los problemas *intratables* se encuentran en un limbo en el que no se puede saber con certeza si en algún momento se desarrollará algún algoritmo que dé solución a cada uno de estos problemas -y si se trata de una solución eficiente-, o si de verdad no existe solución. A los problemas que se encuentran en este limbo se les conoce como problemas NP-completos. Un problema es NP-completo si los problemas NP se pueden reducir[6] a él. Si se resuelve algún problema NP-completo de forma eficiente, significaría que todos los problemas NP también cuentan con solución eficiente, por tanto, se podría afirmar que P=NP [31].

Por definición, se cumple que P pertenece a NP, y ambas clases están incluidas en el conjunto de problemas *decidibles*. Sin embargo, la comprobación de que P=NP está considerada una de las conjeturas sin resolver más importantes de la complejidad computacional[7].

El paradigma de la computación cuántica acelera los procesos de cálculo. Si se aplica sobre los problemas *intratables* existentes, su complejidad temporal que actualmente es exponencial sobre un ordenador clásico pasaría a ser de orden polinómico.

---

[6] Un problema se considera reducible a otro cuando para resolver el primero eficientemente se necesita encontrar primero un algoritmo eficiente para resolver el segundo.

[7] https://www.claymath.org/millennium-problems

El desarrollo del paralelismo cuántico ha conllevado un avance en la complejidad temporal de la computación. Ha permitido que todos los posibles cálculos de una función puedan ser llevados a cabo en un solo instante de la ejecución, obteniendo como salida todos los posibles resultados con una probabilidad asociada en un mismo estado de salida. Gracias a esta potencia de cálculo, es posible resolver problemas que necesitaban más de un siglo para resolverse en el cómputo clásico en un tiempo de a penas unos minutos.

Para tener una idea de la potencia de cálculo que se presenta, un procesador cuántico con 30 cúbits equivale a un procesador convencional de 10 *teraflops* (millones de millones de operaciones en coma flotante por segundo), mientras que actualmente los ordenadores convencionales trabajan en el orden de *gigaflops* (miles de millones de operaciones en coma flotante por segundo) [12].

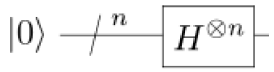
2.4 Algoritmos cuánticos

En 1985, David Deutsch [11] describió el primer modelo de una máquina cuántica abstracta universal -*Quantum Turing Machine* o *QTM*-, basada en el modelo de Turing que obedeciese las leyes mecano-cuánticas, con el objetivo de explotar las ventajas de este paradigma, como el paralelismo cuántico o el entrelazamiento.

Fue Paul Benioff (1981) quien pensó que los ordenadores cuánticos se podrían construir y empezó a experimentar con algunas leyes de la mecánica cuántica sobre máquinas de Turing clásicas. Posteriormente, Richard Feynman aseguró que estos ordenadores podrían llegar a resolver cálculos muy complejos mucho más eficientemente que un ordenador convencional.

Tras crear la *QTM*, Deustch ideó el primer algoritmo cuántico capaz de decidir si una función de caja negra[8] de un solo bit $f : \{0, 1\} \rightarrow \{0, 1\}$ es balanceada o constante[9] en un solo paso, y afianzó la idea de Feynman. Se conoce como **algoritmo de Deustch** [10]. A partir de este momento se empezaron a idear algoritmos y aplicaciones para estas nuevas máquinas cuánticas, y se abrió un abanico de posibilidades de cálculo que hoy en día ya supone un gran avance tecnológico.

Con la aparición de las puertas cuánticas, se pudo empezar a desarrollar aplicaciones prácticas de las propiedades cuánticas que teóricamente ya se planteó que sería beneficioso. La puerta de Hadamard es la más especial de todas ellas porque es la que permite obtener estado en superposición para el cúbit de entrada, permitiendo a partir de ese momento trabajar con estados cuánticos y manipular los dos estados base a la vez. Además, esta puerta se puede generalizar para n cúbits -aunque su transformación sea sobre un solo cúbit de entrada- representando en el circuito la aplicación de la puerta de la siguiente forma:



Cuando se tienen n cúbits de entrada con el mismo valor, se representan con la barra partida, y a continuación se coloca la puerta Hadamard con el símbolo del producto tensorial[10], lo que aplica Hadamard para cada cúbit englobado en esa línea. Esta operación se conoce como transformación de Hadamard ($\hat{H}$) para n cúbits y al aplicarse sobre los cúbits de todo el sistema se obtiene un estado de superposición equiprobable (estado de Bell) para todos los posibles estados que se pueden obtener como solución [10]. Además, como las puertas cuánticas tienen la propiedad de reversibilidad, también es posible aplicar esta puerta sobre cúbits en estado de superposición para obtener estados básicos.

---

[8] Una función de caja negra es aquella de la que no se conoce su código y solo podemos saber su comportamiento comprobando las salidas para entradas diferentes.

[9] Una función se dice que es constante cuando la salida es totalmente independiente de la entrada y siempre es la misma. Mientras que en una función balanceada la salida será la mitad de las veces 0 y la otra mitad, 1.

[10] El producto tensorial de dos vectores se corresponde con el producto combinatorio de cada elemento de ambos vectores.

En este apartado se dan a conocer los algoritmos más importantes que han surgido en este paradigma con el fin de entender las potenciales aplicaciones que presenta este modelo de programación.

2.4.1 Algoritmo de Deustch-Jozsa

Este algoritmo lo propusieron D. Deustch y R. Jozsa [12] en 1992 como mejora del algoritmo de Deustch, con el fin de extenderlo a tantos n bits de entrada como se quisiera. Aunque es de poca utilidad, este algoritmo sirvió para demostrar la mejora de complejidad temporal que eran capaces de conseguir con los ordenadores cuánticos.

Análogamente a su antecesor, este algoritmo determina si una función

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

es constante o balanceada. Un cálculo que para un ordenador clásico conllevaría al menos 2 comprobaciones de 2 salidas diferentes para la función, con un ordenador cuántico podemos decidirlo realizando un solo paso de comprobación.

2.4.2 Algoritmo de búsqueda Grover

Fue propuesto en 1998 por el informático Lov Grover [34], con el fin de mejorar las búsquedas de elementos en bases de datos no indexadas -desordenadas-. Es lo que se conoce como el problema de la *"unstructured search"*.

Supongamos que tenemos una base de datos -o un conjunto de elementos- con N entradas que son, por ejemplo, nombres de usuario y su correo electrónico sin ordenar. Y queremos hacer una búsqueda para encontrar el correo electrónico de un determinado usuario a partir de su nombre. Como la base de datos no está ordenada, solo podemos buscar el elemento por la fuerza bruta, es decir, comprobando todas las entradas una por una. En el mejor caso, el usuario estará en la primera posición y solo se realizará una comprobación, pero si el usuario se encuentra en la última posición, se realizarán N comprobaciones. De media, habrá que hacer $N/2$ comprobaciones para encontrar un elemento en la base de datos, lo que no resulta nada eficiente si tenemos miles de millones de entradas. No obstante, la propuesta cuántica de Grover para solucionar este problema reduce el número de comprobaciones al orden de $O(\sqrt{N})$, que supone una mejora significativa de la complejidad temporal a medida que aumenta el tamaño de la base de datos.

Aprovechando el paralelismo cuántico, la estrategia que sigue este algoritmo es construir un "**oráculo**" -sub-rutina de comprobación- con $log_2(N)$ cúbits de entrada. Para empezar, se deben poner todas las entradas de la base de datos en un estado global de superposición aplicando una transformación de Hadamard. Lo que pone las amplitudes de todos los posibles estados al mismo valor.

$$|\psi_1\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

A continuación, el "oráculo" preguntará simultáneamente a cada una de las posibles entradas del estado en superposición "¿Es esta la entrada correcta?". Esta comprobación es la que hay que definir en el oráculo para que la sub-rutina

responda adecuadamente. El hecho de realizar todas las comprobaciones a la vez es lo que da a este algoritmo tal potencial de eficiencia. Más concretamente, el funcionamiento que sigue este "oráculo" se puede describir de la siguiente forma, que se detalla con un ejemplo.

Consideramos una base de datos con N=16 entradas. Para la que necesitaremos construir un circuito con $log_2(16) = 4$ cúbits para el "oráculo". Los estados base serán $|0000\rangle$, $|0001\rangle$, … , $|1110\rangle$, $|1111\rangle$. Vamos a considerar que la entrada que buscamos es $|1111\rangle$, por tanto:

- Si se trata de cualquiera de las N-1 entradas incorrectas, el "oráculo" devolverá el mismo estado. Para $|0000\rangle$, devolverá $|0000\rangle$.

- Si se trata de la entrada correcta, el "oráculo" invertirá de signo el estado. Así que para $|1111\rangle$, devolverá $-|1111\rangle$.

En el ejemplo, la amplitud equiprobable con la que se empezará el algoritmo será de $1/\sqrt{16} = 1/4$. Por tanto, el estado en superposición inicial del sistema tendrá esta forma:

$$1/4*|0000\rangle + 1/4*|0001\rangle + ... + 1/4*|1110\rangle + 1/4*|1111\rangle$$

Tras aplicar el "oráculo" sobre este estado, el estado del sistema pasará a ser:

$$1/4*|0000\rangle + 1/4*|0001\rangle + ... + 1/4*|1110\rangle - 1/4*|1111\rangle$$

Después de hacer todas las preguntas en un instante de tiempo, la salida del "oráculo" serán todas las respuestas que ha obtenido en estado de superposición. Por tanto, el "oráculo" no devuelve cuál es la entrada correcta, devuelve todas las respuestas que obtiene -N respuestas- entre las que se encuentra la correcta invertida de signo. Si hacemos una lectura del estado de salida del oráculo, este colapsará en alguna de las salidas, pero no tenemos certeza de que sea la correcta.

El último paso del algoritmo consiste en destacar de entre todas las respuestas cuál de ellas es la correcta. Es lo que se conoce como **amplificación de amplitud**. Para este paso, Grover ideó un operador de difusión [8], que hace una inversión sobre la media de todas las amplitudes, para aumentar la probabilidad de la respuesta correcta y disminuir la del resto. Esto sucede porque al tener la probabilidad correcta multiplicada por -1, la media de todas las amplitudes será un poco menor al valor de amplitud equiprobable obtenido al aplicar Hadamard (1/4). Así que al hacer la inversión sobre la media, como la amplitud correcta se encuentra más separada de la media que el resto de valores, se destacará.

El problema es que con una sola comprobación no se llega a obtener una clara diferencia entre las amplitudes. Por tanto, habrá que repetir el proceso de "oráculo" y amplificación hasta obtener una probabilidad muy alta para la respuesta correcta. Además, el número de repeticiones que realizar debe quedar determinado, puesto que si nos excedemos con las iteraciones, se acaban corrompiendo las probabilidades y la correcta disminuye su valor. El número de iteraciones viene dado por el tamaño del problema, así que para N elementos sobre los que buscar, serán necesarias $\sqrt{N}$ iteraciones, de aquí que la complejidad temporal de este algoritmo sea de ese orden [35].

Cabe remarcar que este algoritmo es heurístico, por tanto la respuesta que obtengamos siempre será con cierta probabilidad y habrá que tener en cuenta el ratio de error. Con una cantidad de iteraciones del orden de $O(\sqrt{N})$ se obtendrá una respuesta incorrecta con una probabilidad de $O(1/N)$, lo que es más que aceptable ya que a medida que crezca el tamaño de la base de datos, la probabilidad de error tenderá a cero.

No se desarrolló una versión que funcionase hasta 2017, este algoritmo tiene el impacto menos significativo a niveles de complejidad de todos los algoritmos cuánticos que se han desarrollado, ya que el aumento de eficiencia que supone es de orden cuadrático, frente al resto de algoritmos que son de orden exponencial. No obstante, como el análogo clásico de este método sería la fuerza bruta, sus aplicaciones son numerosas. Por ejemplo, en el campo de *machine learning* se usa este algoritmo para explorar el espacio de soluciones de problemas NP-completos [13]. Además, tras varios estudios se ha confirmado que este algoritmo es una versión óptima para resolver las *"unestructured search"*, y cualquier otra versión similar del algoritmo cuántico debería de llamar al "oráculo" al menos tantas veces como este.

2.4.3 Algoritmo de factorización de Shor

El algoritmo de Shor debe su nombre a su inventor, Peter Shor [], quien en 1995 ideó esta estrategia para resolver un tipo de problema de los más costosos computacionalmente: la factorización de número enteros. Este problema forma parte del conjunto NP, y en particular es del tipo llamado *one-way*. Esta clase de problemas se centran en hacer un cálculo que resulta muy sencillo de resolver en un sentido, pero si se quiere deshacer el cálculo es convencionalmente imposible. Es el caso de la factorización de números enteros considerablemente muy grandes. La idea consiste en factorizar un número dado N, habiendo construido ese número con la operación $p * q = N$, siendo p y q números primos. El proceso resulta trivial para números pequeños -como pueden ser 15, 21 o 33-, pero cuando se trata de números con decenas o centenas de cifras un ordenador clásico no puede abordar el problema, ya que tardaría más de cien años en resolver el cálculo. En la siguiente tabla se puede ver como la complejidad temporal de este problema crece exponencialmente al aumentar la cantidad de cifras del número a factorizar:

Dígitos	Resolución
60	3 días
100	74 años
500	10^{25} años

Tabla 1: Crecimiento del coste temporal de la factorización en función de la longitud del entero.

La solución cuántica que propuso Shor consigue reducir esta complejidad de orden exponencial al orden logarítmico de $O((logN)^3)$, siendo N el número que se quiere descomponer. Lo que resultó muy interesante para deshacer el problema de la factorización que se consideraba intratable.

Criptografía de clave asimétrica: encriptación RSA

La criptografía de clave asimétrica o de clave pública [17] utiliza un par de claves -una pública y una privada- para encriptar y desencriptar los mensajes que se quieren enviar. La clave pública es la que se comparte entre el emisor y el receptor, y la clave privada la conserva el receptor en secreto. Este tipo de criptografía proporciona una mayor seguridad en las comunicaciones porque aunque se comparta una de las claves, sin el conocimiento de la clave privada no es posible descifrar el mensaje.

Por contra, en la criptografía de clave simétrica se utiliza el mismo secreto para el cifrado y el descifrado, y requiere que se comparta dicha clave para que ambas partes puedan entenderse. Si el canal de comunicación por el que comparten esta clave no es totalmente seguro, cabe la posibilidad de que interfiera una tercera entidad y robe la clave para poder leer los mensajes enviados por ambas partes.

El sistema de encriptación RSA fue desarrollado en 1978 por R. Rivest, A. Shamir y L. Adleman [], a quienes debe su nombre. Este algoritmo se basa en el problema de la factorización de números enteros para generar las claves publica y privada, ya que parecía ser un problema irresoluble por los ordenadores clásicos actuales por muy avanzados que fuesen. Su mayor inconveniente es que necesita generar claves de al menos 1024 bits para tener una seguridad fuerte, a diferencia de los algoritmos de clave simétrica que cuenta con una buena seguridad con claves de tan solo 128 bits. Sin embargo, se han desarrollado algunas técnicas clásicas que pretenden resolver este cálculo, como GNFS y los algoritmos de curvas elípticas, pero solo se ha conseguido factorizar números de hasta 232 dígitos decimales (768 bits) [].

Debido a que las claves que se deben generar son considerablemente largas, hacer un uso habitual de RSA resultaría muy costoso. Este algoritmo se suele combinar con criptografías de clave simétrica como AES o DES, haciéndose uso de RSA para compartir las claves simétricas entre emisor y receptor de forma segura.

El algoritmo funciona de la siguiente forma:

El objetivo es generar dos pares de números que serán la clave publica -*PUB(N, E)*- y la clave privada -*PRIV(N, D)*-. La clave *PUB* será la que se comparta entre emisor y receptor y no importa si es conocida. Como se ve, el número *N* se encuentra en ambas claves, y si la clave pública es conocida entonces conocemos también la mitad de la clave privada. Pero ahora veremos como no tiene relevancia conocer *N*, sino que la clave de la encriptación se encuentra en el número *D*.

Para empezar se escogen dos números primos *p* y *q*, a partir de los que se obtiene el número *N* multiplicándolos. Para mostrar un ejemplo del funcionamiento del algoritmo vamos a escoger dos números simples para construir un caso de uso trivial. Escogemos $p = 3$ y $q = 11$ y se calcula $N = p*q$ y $Z = (p - 1)*(q - 1)$:

$$N = p*q = 7*11 = 33$$
$$Z = (p - 1)*(q - 1) = 2*10 = 20$$

A continuación se busca un número *E* que sea coprimo[11] de *Z*. Por ejemplo, *E*=7 para nuestro caso *Z*=20. Y debemos encontrar también un número *D* tal que:

$$D*E = 1 \ mod \ Z$$

[11] Un número es coprimo de otro cuando no tienen factores primos en común, por lo que el único factor común que comparten es 1 y -1.

Para nuestro ejemplo se cumple esta igualdad con $D = 97$ tal que:

$$7*3 = 1 \ mod \ 20$$

Una vez hemos encontrado los números que cumplen esta condición ya se pueden construir las claves de forma que:

$$PUB(33, 3) \text{ y } PRIV(33, 7)$$

Estas serán las claves que se utilizarán para cifrar los mensajes que se quieran transmitir.

Ahora vamos a proceder a cifrar un mensaje que el emisor quiera enviar de la siguiente forma:

Supongamos que el emisor quiere enviar la palabra "ALFA" al receptor. El procedimiento del algoritmo deberá aplicarse a cada letra de la cadena de texto que se quiera enviar. Empezamos por la letra "A". Primero de todo, se codifica dicha letra para obtener un número que la represente en algún sistema de codificación -por ejemplo ASCII-. La representación para el texto plano "A" es $Tp = 65$.

A continuación aplicamos la siguiente fórmula para obtener la letra cifrada:

$$Tp^E \ mod \ N = 65^3 \ mod \ 33 = 274625 \ mod \ 33 = 32 = Tc$$

Este será el valor cifrado que se enviará al receptor, y para obtener el texto real deberá realizar la operación complementaria:

$$Tc^D \ mod \ N = 32^7 \ mod \ 33 = 34359738368 \ mod \ 33 = 65 = Tp$$

Este algoritmo tuvo gran éxito porque su complejidad es muy elevada para números muy grandes. Y a pesar de que no resulta eficiente para cifrar grandes cantidades de datos, juega un papel fundamental para comunicar las claves que cifran tales volúmenes de datos.

Funcionamiento del algoritmo de Shor

La estrategia que sigue este algoritmo está basada en la Teoría de Números[12]. Aprovechando que el método RSA se desarrolla a partir de cálculos modulares, Shor ideó que se podría llegar a obtener el mismo resultado parametrizando este cálculo modular y utilizando la potencia cuántica para resolverlo.

Como este algoritmo va a ser objeto de estudio en el proyecto, a continuación se va a detallar su funcionamiento paso a paso para entenderlo claramente. Se va a escoger $N = 15$ como ejemplo trivial del algoritmo. Para encontrar un número primo factor de un número dado N impar, el algoritmo sigue los siguientes pasos:

1) Primeramente, se debe escoger un número primo básico a, tal que $a<N$, y que a sea coprimo de N. Vamos a empezar escogiendo $a = 2$.

2) A continuación, se calculan todos los valores para la función $\boldsymbol{f_{a,N}(x) = a^{2^x} \ mod \ N}$. En este punto es donde entra en juego la potencia del cálculo cuántico. Podemos obtener todos los valores necesarios para esta función haciendo uso

[12] La Teoría de Números es la rama matemática que estudia las propiedades de los números enteros.

de la superposición y el paralelismo cuántico de forma que en un solo paso obtengamos todos los resultados de $f_{a,N}(x)$ que queremos. Este paso es el que se conoce como "*order-finding*" [14].

En nuestro ejemplo, la función que deberemos calcular es $f_{2,15}(x)=2^x \bmod 15$. Obteniendo los respectivos valores para $f_{2,15}(0)$, $f_{2,15}(1)$, $f_{2,15}(2)$, ….

Una vez obtenidos los resultados de $f_{a,N}(x)$, se puede observar que los valores en algún momento se repiten y siguen un patrón. Esto se debe a que el cálculo se realiza sobre una función modular -o lo que es lo mismo, función periódica-, por lo que en algún momento se producirá una repetición de números y entonces podremos determinar el periodo r de la función contando la cantidad de números que se engloba en una repetición. En caso de que el periodo encontrado sea impar, se debe desechar el número a seleccionado y escoger el siguiente que cumpla con los requisitos iniciales.

En el caso del ejemplo, hemos obtenido que r=4, como es par es un periodo válido.

La Teoría de Números explica que para cualquier co-primo $a <= N$, la función $f_{a,N}(x)$ volverá a valer 1 para algún $r<N$, por tanto se cumple que:

$$f_{a,N}(r+1) = f_{a,N}(1)$$

Si el r obtenido es par, entonces [15]:

$$a^r \equiv 1 \bmod N$$

En caso de que r sea impar, deberemos volver al paso *1)* y escoger otro número para a.

3) A partir de este momento se hacen una serie de cálculos que ya no requieren del uso de las propiedades cuánticas porque matemáticamente no son complejos.

Sabiendo ya el periodo r, para obtener un divisor de N primero de todo restaremos 1 a cada lado de la equivalencia anterior:

$$a^r - 1 \equiv 0 \bmod N \;\; \text{—>} \;\; a^r - 1 \equiv N$$

A partir de la equivalencia anterior se puede deducir que $a^r - 1$ será un divisor de N, ya que si es equivalente a $0 \bmod(N)$ significa que el resto de hacer la división es nulo y es un divisor válido.

A continuación elevamos al cuadrado ambas partes de la equivalencia[13]:

$$(\sqrt{a^r} - 1)(\sqrt{a^r} + 1) \equiv N \;\; \text{—>} \;\; (a^{r/2} - 1)(a^{r/2} + 1) \equiv N$$

Esta transformación matemática se hace para obtener dos resultados al resolver la equivalencia, es decir, para obtener los dos factores de N.

[13] Se aplica la propiedad matemática $x^2 - y^2 = (x + y)(x - y)$.

4) Los números que se obtienen al hacer las sustituciones correspondientes en la equivalencia tendrán los mismos factores que *N*, así que para encontrar dichos factores bastará con aplicar el algoritmo de Euclides del máximo común divisor -*MCD*- entre $(a^{r/2} -1)$ y *N*, y $(a^{r/2} +1)$ y *N*. Este cálculo es sencillo para un ordenador clásico y con un coste temporal bajo. El resultado de $MCD((a^{r/2} -1), N)$ y $MCD((a^{r/2} +1), N)$ serán ambos factores primos que estamos buscando.

Por tanto, en el ejemplo deberemos calcular

$$(a^{r/2} -1) = (2^{4/2} -1) = (2^2 -1) = 3$$
$$(a^{r/2} +1) = (2^{4/2} +1) = (2^2 +1) = 5$$

y obtener el máximo común divisor entre estos factores y 15, que como ya se observa a simple vista, los propios números obtenidos son los factores que buscábamos:

$$MCD(3, 15) = 3$$
$$MCD(5, 15) = 5$$

Más adelante, en el Capítulo 4, expondremos implementaciones en circuitos cuánticos para cada uno de los algoritmos explicados y veremos cuáles son las expectativas y los resultados que se obtienen al ejecutarlos.

2.5 Arquitecturas para un procesador cuántico

La arquitectura básica desarrollada para los procesadores cuánticos sigue la misma filosofía que los convencionales. Oskin y Chuang *et. al.* [21] propusieron un modelo basado en cuatro componentes principales:

- El componente fundamental es la **ALU cuántica**, que se encarga de ejecutar las operaciones matriciales que corresponden a las puertas del circuito, de forma similar a como funciona la ALU clásica con las operaciones aritméticas.
- Para agilizar la ejecución, el procesador cuenta también con varios bloques de **memoria** cuántica que desempeña la función de memoria RAM dinámica y almacena los cúbits y sus estados. Cada bloque de memoria cuenta con una unidad de actualización que se encarga de mantener los estados de los cúbits refrescados en cada momento y que no hayan inconsistencias de información.
- La información de los cúbits que se encuentran en la memoria pasan a la ALU a través del **tele-transportador de código**, que hace de la función de bus entre ambos componentes utilizando la propiedad cuántica de la teleportación que se produce entre cúbits entrelazados. Capaces de transmitir información sin ningún tipo de comunicación con contacto.
- Todo el proceso de ejecución lo controla un microprocesador clásico que funciona como **planificador dinámico** que, empleando algoritmos de planificación, mezcla las operaciones cuánticas con estructuras de control de flujo para conseguir un código más completo.

Además, en todos los componentes se añade una pequeña unidad de corrección de errores que se centra en detectar y mitigar las perturbaciones que aparecen en los cúbits.

Para implementar esta arquitectura, se utilizan varias tecnologías físicas que construyen los cúbits lógicos a partir de estructuras de partículas:

- **Superconducción**: Esta técnica es la más utilizada hasta el momento por grandes empresas como IBM, Google y D-Wave.

  El cúbit se construye a partir de un par de metales superconductores separados por un nanómetro, entre los cuales se atrapa el tránsito de electrones y se manipulan utilizando pulsos de microondas para generar los estados de superposición y el entrelazamiento.

  Estos ordenadores necesitan estar completamente refrigerados a una temperatura muy cercana al cero absoluto, lo que resulta una dificultad para la corrección de errores.

  Estas máquinas presentan una velocidad de cómputo muy elevada, pero mantienen una coherencia cuántica corta, lo que hace que la información procesada no perdure a lo largo de toda la ejecución.

- **Trampa de iones**: Se basa en el uso de chips que crean campos electromagnéticos para atrapar iones e incidir sobre ellos mediante rayos láser para conducirlos hacia un estado u otro. Con estos láseres se consigue manipular los cúbits a lo largo de la ejecución a realizar, y se consigue un tiempo de coherencia bastante aceptable para realizar cálculos complejos. Para medir el estado de los cúbits, se

iluminan todos los iones por igual, y dependiendo de si emiten luz o no se determina el estado que posee.

Estas máquinas son refrigeradas mediante láseres que mantienen la temperatura óptima cercana al cero absoluto, y se encuentran en neveras herméticas casi al vacío. Gracias a este aislamiento del ordenador, el ruido que interfiere en los cúbits es muy pequeño porque los iones no chocan con otros átomos que puedan haber en la nevera.

Aunque no tan demandada desde los inicios, esta técnica la están utilizando empresas como Honeywell y IonQ, y pretenden darle un buen ritmo de evolución. IonQ publicó a finales del año pasado que ya poseía un ordenador cuántico de este tipo que contaba con 32 cúbits [19].

- **Fotones**. La característica de los cúbits fotónicos es que no interaccionan entre sí, por tanto no se propagan los errores y las medidas de tolerancia a fallos resultan efectivas. Los cúbits se forman haciendo viajar un tránsito de fotones a través de unas guías de ondas fotónicas creadas con silicio, y consiguen entrelazarse usando componentes ópticos. Una ventaja de esta arquitectura es que con una serie de cúbits entrelazados y detectores de fotones se puede implementar cualquier puerta lógica.

  Esta forma de interacción de los cúbits permite crear las series de cúbits, llamadas "estados cluster". Así, en lugar de trabajar sobre cúbits individuales, se interacciona con cadenas de cúbits capaces de mantener un estado entrelazado y de procesar una gran cantidad de información. Esto permitió empezar a trabajar sobre bytes cuánticos. Además, estos ordenadores se pueden mantener a temperatura normal, sin necesidad de extremarla al cero absoluto, lo que supone una gran ventaja frente a otras arquitecturas que requieren de un entorno específico para reducir los errorres. Es posible probar esta implementación sobre fotones en las máquinas de Xanadu, disponibles en su cloud Xanadu Cloud.

Las dos primeras implementaciones presentan un alto nivel de error si se intentan escalar a máquinas de un millón de qubits, por ejemplo. Ante esta barrera, ha surgido una nueva implementación basada en **fotones**. El único ruido que sufre el circuito se halla en las imperfecciones de los componentes [18].

2.6 Problemática y límites de la computación cuántica

Frente a todo el potencial que ofrece el desarrollo de esta nueva tecnología, el hardware sobre el que se respalda la computación cuántica presenta ciertas características que condicionan la explotación de esta computación para la resolución de problemas realmente relevantes. Los materiales sobre los que se construyen los cúbits necesitan de unas condiciones para poder manipular estados en superposición y mantener la coherencia cuántica el mayor tiempo posible. Si esto no se mejora, el tamaño de los problemas que pueden abarcar los ordenadores cuánticos se limita a unos pocos casos simples que de poco sirven para la realidad.

El tiempo durante el que se mantiene la coherencia cuántica, es decir, el estado en superposición de los cúbits de entrada del circuito que implemente cierto algoritmo, se denomina tiempo de coherencia. Este tiempo es la característica clave que hace que un procesador pueda solucionar problemas a gran escala o no. Si el tiempo de coherencia es bajo, los cúbits perderán su estado en superposición antes de finalizar el algoritmo y, o bien vuelven al estado base inicial -normalmente cero-, o bien toman valores aleatorios para las amplitudes. Por tanto, los resultados que se obtienen no son válidos. Al principio del desarrollo de los ordenadores cuánticos, el tiempo de coherencia que se obtenía era de unos pocos nanosegundos, un tiempo realmente muy limitado para resolver cualquier problema.

2.6.1 Características limitantes

El modelo teórico que fundamenta el paradigma cuántico es de carácter sólido y firme a nivel matemático, lo que ha llevado a impulsar el desarrollo de toda la tecnología cuántica. Sin embargo, al construir *chips** que implementen el concepto lógico de los cúbits han surgido diversos inconvenientes que suponen un reto para el avance en esta área.

En primer lugar, debido a las arquitecturas empleadas para la creación de los ordenadores cuánticos -superconductores y trampas de iones- el entorno interfiere en la coherencia cuántica de los cúbits produciendo una perturbación en las amplitudes que el estado cuántico del sistema, o lo que es lo mismo, produciendo la decoherencia cuántica. Es lo que se conoce como **ruido cuántico** []. Este ruido puede hacer que las amplitudes tomen valores erróneos y el estado cuántico de los cúbits deje de ser válido. Generalmente el ruido proviene de las puertas lógicas cuánticas, debido a su construcción no perfecta. No obstante, para mitigar la decoherencia, se han desarrollado diversas técnicas de corrección de errores como por ejemplo la implantación de *firmware* que estabiliza los cúbits rotando su esfera hacia una misma dirección constante y evitando que tomen valores aleatorios. Con esta técnica se ha llegado a estabilizar los cúbits consiguiendo un tiempo de coherencia de 10 minutos []. Pero en el momento que interactúan la coherencia empieza a perderse. La decoherencia es el hecho más limitante de la práctica de este paradigma que realmente dificulta el desarrollo de cálculos que excedan un determinado tiempo y es el foco de atención de las empresas que construyen las máquinas cuánticas reales.

Además de esto, para la construcción de un circuito cuántico es necesario poder inicializar todos los cúbits a un mismo estado base -o estado clásico-, y para ello la temperatura a la que se debe encontrar el material tiene que estar muy cerca del

cero absoluto (cero grados Kelvin). Los ordenadores cuánticos implementados con superconductores o trampas de iones necesitan estar en espacios refrigerantes con una temperatura lo más próxima al cero absoluto. Esto se ha llegado a conseguir en todas las empresas que comercializan estas arquitecturas, y constan de neveras que mantienen los ordenadores a una temperatura de 15mK. En caso de no mantener la temperatura de los cúbits en este bajo nivel los desestabiliza y también produce amplitudes erróneas. No obstante, el hecho de tener que mantener estas máquinas a unas temperaturas tan extremas dificulta el acceso a su construcción.

2.7 ¿Qué se considera supremacía cuántica?

Desde que se empezó a desarrollar este nuevo paradigma cuántico a finales del siglo pasado, los físicos y matemáticos han querido darle sentido a toda esta investigación, y ha sido necesario responder a la pregunta de para qué nos sirve esta nueva computación.

Los problemas NP que comentábamos antes han formado parte del desarrollo de los algoritmos que han aprovechado este potencial de cálculo. Se pensó que con la superposición de estados de los cúbits los problemas que parecían irresolubles con la tecnología clásica pasarían a ser problemas abordables en un tiempo considerable casi polinomial.

El concepto de **supremacía cuántica (computación post-clásica)** hace referencia a la capacidad de los ordenadores cuánticos de resolver problemas que los ordenadores clásicos no pueden resolver o tardarían demasiado tiempo en ello. Para conseguir esta supremacía, los procesadores cuánticos han de ser capaces de escalar hasta formar una arquitectura lo suficientemente potente como para ser capaces de abarcar la resolución de un problema real.

Expertos de las empresas punteras en esta área afirman que para conseguir la supremacía cuántica será necesario construir procesadores de al menos cúbits. Actualmente, ya se están desarrollando ordenadores con una capacidad superior a esta, pero por el momento es poca la información pública que podemos leer, aunque empresas como Google aseguraron en 2020 que ya han superado el umbral de la supremacía cuántica con su procesador Bristlecone, que cuenta con un total de 72 cúbits [1].

El hecho de alcanzar un ordenador cuántico con estas prestaciones tan potentes supone tener que hacer un cambio en el *modus operandi* de la informática clásica. El hecho de que hayan ordenadores capaces de abordar los problemas más difíciles planteados hasta el momento abre un mundo de mejoras y optimizaciones en el tratamiento de datos. Pero a su vez pone en riesgo la seguridad de muchos sistemas y brinda un poder a quien posea dichas máquinas. Por ejemplo, si el algoritmo de Shor es capaz de factorizar los números que forman las claves RSA en un tiempo polinomial, el mundo de las comunicaciones se enfrenta a que los mensajes cifrados que se envían alrededor de todo el mundo puedan ser leídos por una tercera entidad sin autorización previa, es decir, que toda la información cifrada con clave asimétrica podría estar a disposición de quien cuente con esta supremacía cuántica. Y aunque son muchas las aplicaciones que está teniendo este paradigma computacional más allá de un abuso de poder, cabe tener en cuenta esta posibilidad de prácticas poco éticas que ofrece el uso de este potencial.

Pero la supremacía cuántica no solo habla de superar los límites de problemas computacionales clásicos, si no que aspira a conseguir la resolución de problemas que hasta ahora no hemos sido capaces de encontrar una forma de abarcarlos porque la tecnología no permitía procesarlos. Como por ejemplo las ecuaciones Hamiltonianas de las moléculas, de las cuales no somos capaces de extraer toda la información de la molécula debido a las propiedades cuánticas que afectan a las particulas que las forman. Con ordenadores convencionales se consigue una simulación aproximada, pero no completa de estas [41].

CAPÍTULO 3
Estado del arte

A lo largo de los más de veinte años de existencia de la computación cuántica se ha desarrollado gran cantidad de tecnología que ha dado uso al modelo teórico que se presentó. Desde el desarrollo de simuladores de procesadores cuánticos para ordenadores convencionales, hasta la creación de *hardware* cuántico real. Grandes e influyentes empresas tienen su propia linea de investigación en esta área para conseguir un desarrollo completo de la computación cuántica que permita dar el salto a este nuevo modelo de procesamiento y alcanzar la supremacía cuántica que tanto da que hablar.

En este capítulo vamos a hacer una presentación de las tecnologías más interesantes que se han desarrollado y se están usando para este paradigma.

3.1 Lenguajes cuánticos de alto nivel más importantes

Un lenguaje de programación de alto nivel proporciona una capa de abstracción entre el lenguaje que entiende el ordenador -binario o ensamblador- y la forma de comunicarnos que tenemos los humanos. Estos lenguajes utilizan expresiones muy similares al lenguaje natural y facilita su comprensión [29].

Los lenguajes cuánticos de alto nivel se respaldan sobre lenguajes de bajo nivel que operan con el procesador con instrucciones ensamblador, de igual forma que lo hacen los lenguajes no cuánticos. El *Quantum Assembly Language* es la base de desarrollo de otros lenguajes similares a C o FORTRAN que se utilizan para construir los circuitos cuánticos que representan los algoritmos ya mencionados en el apartado 2.5 y muchos otros.

3.1.1 QCL: *Quantum C Language*

El lenguaje QCL o *Quantum C Language* [22] fue uno de los primeros lenguajes cuánticos que se desarrolló, pretendiendo ser un análogo del lenguaje C. Es un lenguaje estructurado procedural interpretado, es decir, que ejecuta sentencias en un entorno virtual de un intérprete sin necesidad de compilar todo el código de una vez, lo que permite hacer modificaciones en el código en tiempo de ejecución.

Además de toda la gramática procedente de su referente clásico el lenguaje C, QCL añade nuevos tipos de datos cuánticos -por ejemplo `qureg` o `quconst`, que declaran registros cuánticos-, instrucciones para las transformaciones de las puertas cuánticas e instrucciones de lectura para los registros cuánticos mencionados [10].

Un registro cuántico puede ser simple -cuando solo utiliza un cúbit- o compuesto -cuando abarca más de un cúbit-. El número máximo de cúbits utilizables se puede designar como parámetro del comando del intérprete `qcl`.

A continuación vamos a ver una ejecución de un ejemplo simple para mostrar el funcionamiento de este lenguaje.

El circuito que vamos a construir va a declarar el uso de un registro de dos cúbits, al que se le va a aplicar una puerta Hadamard para obtener un estado de superposición y a continuación leeremos el estado resultante a través de una variable clásica.

```
qcl> qureg q[2];        // Declara un registro de 2 cúbits
qcl> int l;             // Declara un registro de lectura
qcl> H(q);              // Aplica la puerta Hadamard (H)
 [2/32] 0.5 |0> + 0.5 |1> + 0.5 |2> + 0.5 |3>
qcl> measure q[0..1], l // Lee los 2 cúbits del registro
[2/32] 0.5 |0> + 0.5 |1> + 0.5 |2> + 0.5 |3>
qcl> print l;      // Muestra el resultado de la lectura
: 2
```

Se puede observar que se muestra el estado del registro, con las cuatro combinaciones posibles de estados para dos cúbits, en los resultados de cada instrucción. Al aplicar Hadamard, todos los estados obtienen una amplitud equiprobable, lo que hace que al leer el registro el estado resultante pueda ser cualquiera de los cuatro. Para este caso, se ha leído el estado 2, que corresponde a `|10>`. A partir del momento que se lee el `qureg` ya no podemos seguir operando *cuánticamente* con él. Sin embargo, podemos aplicar la instrucción `reset` para devolver los cúbits a su estado inicial y seguir trabajando con ellos.

```
qcl> reset;             // reinicializa todos los qureg
[2/32] 1 |0>
```

Tras esta instrucción, todos los cúbits que se hayan declarado para su uso en el programa recuperan su estado inicial `|0>`, de ahí que la respuesta del intérprete sea que con probabilidad total el estado que se lea sea este.

El lenguaje QCL permite crear funciones y sub-rutinas propias para importarlas en algoritmos más complejos como puede ser el algoritmo de Shor, que consta de una implementación en QCL propuesta por Bernhard Ömer [22]. No obstante, este lenguaje a día de hoy se ha quedado obsoleto y sirve como referente para transpilar lenguajes de más alto nivel a código ejecutable por ordenadores cuánticos, como por ejemplo QASM.

3.1.2 Qiskit

Qiskit[14] es un *SDK* de código abierto desarrollado por IBM. Este *framework* en forma de librería desarrollado sobre Python está orientado a la construcción de circuitos cuánticos orientados a objetos, a los que se les añade las puertas cuánticas deseadas.

Este *framework* cuenta con varias librerías de algoritmos, mitigación de errores, optimización de código, simuladores para ejecución en local, y librerías enfocadas a ámbitos de investigación como la química, *machine learning* o finanzas. Las cuatro librerías principales son:

[14] Página oficial de Qiskit: https://qiskit.org/

- *Aqua*. Librería de algoritmos cuánticos ya implementados.
- *Aer*. Simuladores para ejecutar de forma local.
- *Terra*. Herramientas para construir circuitos cuánticos en código máquina.
- *Ignis*. Herramientas para ajustar el ruido cuántico en los simuladores.

De igual forma que con el lenguaje anterior, vamos a implementar un pequeño ejemplo de funcionamiento de este *software* para mostrar las características de programación de un circuito y, posteriormente, su ejecución.

Es interesante destacar que para la visualización de resultados, Qiskit ofrece herramientas de impresión de un histograma que muestra un gráfico con las amplitudes resultantes para cada estado, para ver de forma más clara las probabilidades obtenidas. Además también tiene herramientas para imprimir el circuito desarrollado. Podemos desarrollar código tanto en un *IDE* en local como en la plataforma *IBM Quantum Experience*, que cuenta con un entorno de desarrollo integrado respaldado en Jupyter y que enlaza directamente con los *providers* de IBM.

A continuación se muestra el código de ejemplo implementado en Qiskit y el histograma obtenido. Además, desde *IBM Quantum Experience* podemos ver el código compilado a lenguaje ensamblador OPENQASM[15].

```python
# Instancia el simulador local
# simulator = QasmSimulator()

# Acceso a la cuenta de IBM Quantum
provider = IBMQ.enable_account('d4b1cf6aef2786bebb9ea75f65fb5a5f1
# Instancia de un ordenador cuantico de 15 cubits
backend = provider.get_backend('ibmq_16_melbourne')

# Crea el circuito cuantico de 2 cubits con 2 bits para lectura
circuit = QuantumCircuit(2, 2)

# Aplica Hadamard al cubit 0
circuit.h(0)

# Aplica CNOT con el cubit 0 de control sobre el cubit 1
circuit.cx(0,1)

# Lectura de los resultados en los bits clasicos
circuit.measure([0,1], [0,1])
```

(a)

```python
# Compila el circuito a instrucciones QASM
transpiled = transpile(circuit, backend=backend)
job = backend.run(transpiled)

# Ejecuta el circuito 100 veces
# job = simulator.run(compiled_circuit, shots=100)

result = job.result()

# Obtiene las probabilidades resultantes
probabilities = result.get_counts(circuit)
print("Probabilidades obtenidas:", probabilities)

# Imprime el circuito
circuit.draw()

# Imprime las probabilidades
plot_histogram(probabilities)
```

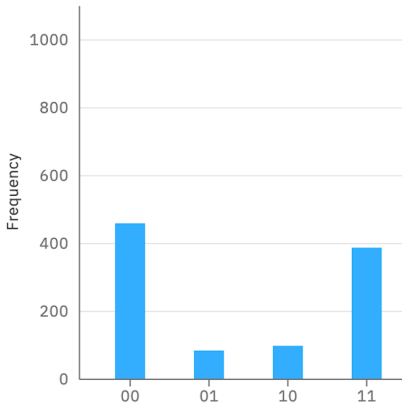
```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[15];

creg c[2];


rz(1.5707963267948966) q[0];
sx q[0];
rz(1.5707963267948957) q[0];
cx q[0], q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

(b)



(d)

(c) Probabilidades obtenidas: {'00': 458, '01': 83, '10': 97, '11': 386}

[15] https://github.com/QISKit/openqasm

Figura 3.1. Código de ejemplo en Qiskit(a), compilación de dicho código(b), el resultado tras la ejecución(c) e histograma resultante(d).

3.1.3 Cirq

De forma similar a Qiskit, Cirq[16] es otro *SDK* en forma de librería de Python para la implementación de circuitos cuánticos desarrollado por Google. Comparte muchas similitudes con el framework de IBM a nivel de escritura de código, ya que ambos tienen un enfoque de construir los circuitos como objetos a los que se les añaden las puertas como atributos. El concepto de cúbit también se trata como objeto, del que existen dos abstracciones -o tipos de datos-, que son el `GridQubit` y el `LineQubit`. El primero posiciona los cúbits en una red y los representa mediante coordenadas -tratándolo como un array-, mientras que el segundo posiciona los cúbits en una linea que los indexa con un solo entero. Además de los cúbits, Cirq incluye un concepto de abstracción dimensional: los *cúdits*. Un ***cúdit*** es un bit con d dimensiones, es decir, que los posibles estados que puede tener van de 0 a $d - 1$. Este nuevo tipo de datos solo puede ser utilizado en puerta cuánticas de la misma dimensión que el *cúdit*.

```
q0 = cirq.LineQid(0, dimension=3)
circuit = cirq.Circuit(
    QutritPlusGate().on(q0)
)
```

Figura 3.2. Construcción de un circuito con una puerta de dimensión 3 y un cútrit.

En la Fig. 3.2 se define un cútrit -cúdit de tres dimensiones-, se crea un circuito y utiliza una puerta de dimensión 3 que trabaja con dicho cútrit.

Cirq además cuenta con un tipo de colección de objetos que representa un momento temporal del circuito, es el objeto `Moment`. Un `Moment` engloba todas las operaciones que se realizan sobre todos los cúbits del circuito en un momento temporal especificado. Podemos construir un circuito como una colección iterable de momentos.

```
qubits = [cirq.GridQubit(x, y) for x in range(3) for y in range(3)]

# Creacion de las puertas cuanticas
cz01 = cirq.CZ(qubits[0], qubits[1])
x2 = cirq.X(qubits[2])
cz12 = cirq.CZ(qubits[1], qubits[2])

# Construccion del circuito
moment0 = cirq.Moment([cz01, x2])
moment1 = cirq.Moment([cz12])
circuit = cirq.Circuit((moment0, moment1))
```

Figura 3.3. Construcción de un circuito encadenando Momentos.

[16] https://quantumai.google/cirq

Esta no es la única forma de construir circuitos ni la más común, pero resulta útil para organizar el orden de las operaciones y visualizar las dependencias de cada puerta sobre los cúbits.

En la Fig. 3.4 se muestra un ejemplo de un programa que ejecuta el mismo código ya descrito en apartados anteriores sobre el simulador básico de Cirq y el resultado obtenido.

```
import cirq

# Crea el circuito y los 2 cubits a utilizar
circuito = cirq.Circuit()
(q0, q1) = cirq.LineQubit.range(2)

# Añade al circuito una Hadamard y una CNOT
circuito.append(cirq.H(q0))
circuito.append(cirq.CNOT(q0, q1))

# Añade al circuito la lectura de los cubits
circuito.append([cirq.measure(q0), cirq.measure(q1)])
print(circuito)

# Instancia el simulador y ejecuta 100 veces
simulador = cirq.Simulator()
resultados = simulador.run(circuito, repetitions=100)
print(resultados)
```

(a)

```
0: ———H———@———M———
          |
1: ———————X———M———
```

(b)

```
0=0111010000101011011001110111100001100110110001011011100110000110010000010100010111100000000001111010
1=0111010000101011011001110111100001100110110001011011100110000110010000010100010111100000000001111010
```

(c)

Figura 3.4. Código de ejemplo en Cirq (a), dibujo del circuito (b) y resultado de la ejecución(c).

La cadena de respuesta que se obtiene por consola (B) representa los valores que toman ambos cúbits en las 100 ejecuciones. Por ejemplo, la Fig. 3.4 muestra que en la primera ejecución el estado resultante es $|00\rangle$, mientras que en la siguiente ejecución es $|11\rangle$.

Este *framework* no cuenta con herramientas para imprimir diagramas o un resultado que sea más agradable visualmente.

3.2 Proveedores de servicios cuánticos en la nube

Un proveedor de servicios *cloud* o en la nube es una empresa que ofrece servicios a terceros que pueden ser: infraestructura(*IaaS*), plataforma(*PaaS*) o aplicación(*SaaS*). Estos servicios se consumen en base al modelo de pago por uso, aunque algunos cuentan con plan gratuito para probar su plataforma. El hecho de que los servicios de computación cuántica estén disponibles a través de internet conlleva beneficios para las empresas investigadoras que no se tienen que desplazar hasta otros laboratorios, o no tienen que fabricar su propio procesador cuántico.

3.2.1 IBM Quantum Experience

Con servicio disponible desde 2016, cuenta con casi veinte ordenadores cuánticos disponibles según el plan de servicio que tengas contratado [25]. De

forma gratuita tiene disponibles máquinas de 1, 5 y 15 cúbits. No obstante, con una subscripción de investigación o educativa ofrecen ordenadores de hasta 65 cúbits. Todas estas máquinas se han desarrollado con un procesador de material superconductor. Dentro de la plataforma se puede desarrollar código en el *Quantum Lab*, un IDE de Jupyter integrado en la plataforma desde el que puedes lanzar ejecuciones. Además cuenta con un módulo llamado *Quantum Composer*, un editor gráfico de circuitos cuánticos que permite construir programas añadiendo las puertas cuánticas directamente sobre el diagrama de los cúbits.

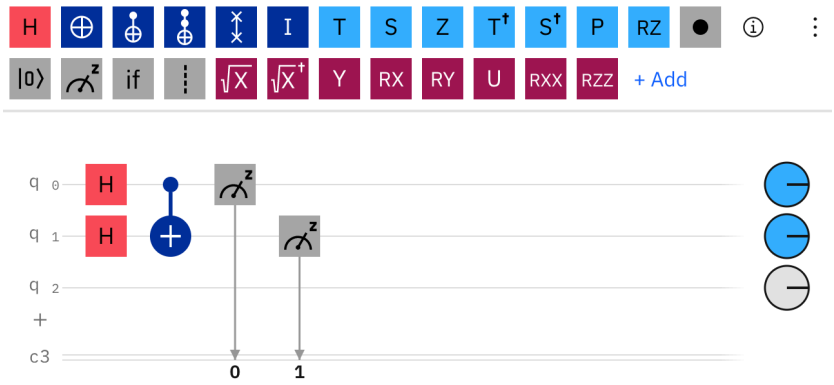


Figura 3.5: Composer de circuitos de IBM Quantum.

La plataforma cuenta también con 5 simuladores de varias características, incluyendo un simulador de 5000 cúbits, el más potente de todos. Estos simuladores son accesibles desde cualquier subscripción mediante la *API key* de usuario.

3.2.2 Microsoft Azure Quantum

Es un servicio de pago por uso disponible en la plataforma de Microsoft Azure, que no cuenta con plan gratuito. Desde este servicio se pueden ejecutar programas cuánticos contra máquinas propias de Microsoft o contra otros proveedores como IonQ o Honeywell, ambas empresas constructoras de ordenadores cuánticos. El lenguaje predominante en esta plataforma es Q#, desarrollado por Microsoft, que puede usarse con Python o con .NET [27].

3.2.3 Google Quantum AI

Es el área *cloud* de Google dedicada a esta computación. Su servicio, *Google Quantum Computing Service*, todavía no se encuentra disponible de forma pública en su plataforma -*Google Cloud*-, sino que solo obtienen acceso bajo demanda aquellas instituciones o empresas que se dediquen a la investigación [26]. *Google Quantum Computing* actualmente comercializa procesadores superconductores de hasta 54 cúbits -basados en su procesador Sycamore- a través de su API *Quantum Engine* para Cirq. No obstante, el servicio cuenta con otros proveedores de hardware como IonQ o Alpine Quantum Technologies. Y aparte, el procesador Britlescone, de 72 cúbits, es la mejor apuesta de esta empresa hasta el momento,

con el que aseguran que llegarán a alcanzar la supremacía cuántica. Pero aún no está disponible para su uso comercial.

Además de los procesadores cuánticos, ofrece varios simuladores de código abierto disponibles para su uso a través del gestor de paquetes de Python `pip`: qsim y qsimh [30], y otros simuladores externos disponibles en Github, como por ejemplo qFlex, un simulador desarrollado por la NASA[17]. Este último será el que utilizaremos más adelante para las pruebas del experimento.

3.2.4 Otras empresas del mercado

Existen más empresas que también han apostado por el desarrollo de procesadores cuánticos aprovechando otras arquitecturas que mejoran la coherencia cuántica y reducen el ratio de error para obtener cálculos más fiables. Como Xanadú Cloud[18], que ha construido procesadores fotónicos de hasta 24 cúbits [24]. O Honeywell[19] e IonQ[20], ambas empresas del sector tecnológico que proveen de su servicio cuántico a Microsoft o Google y que han construido procesadores basados en trampas de iones.

3.4 Crítica al estado del arte de la tecnología cuántica

El éxito que está teniendo esta área de la informática se debe a la gran ayuda que proporciona en los ámbitos de investigación más cotizados actualmente: simulaciones de sistemas químicos, M*achine Learning*, procesos financieros u optimización de problemas. La computación cuántica combinada con la informática convencional permite que los pasos hacia adelante que se den sean mucho más grandes y el ritmo de investigación se acelere.

La carrera hacia la construcción de procesadores cada vez más potentes está en el punto de mira de los gigantes tecnológicos. IBM fue la primera empresa en apostar por empezar a comercializar las máquinas cuánticas -como lo hizo antaño con los ordenadores convencionales-. El modelo de negocio QaaS -*Quantum as a Service*- es la comercialización actual que se le está dando a esta tecnología, y será el negocio que permanecerá por el momento, mientras que estos ordenadores estén en fase de desarrollo.

Generalmente, ninguna empresa se impone sobre otra. No obstante, IBM resulta ser la única que proporciona servicios gratuitos, con un paquete de software estable y una plataforma amistosa, lo que resulta atractivo para la gente que quiere iniciarse en los conocimientos de este paradigma y quiera probar a ejecutar su código en máquinas reales.

---

17 **https://github.com/ngnrsaa/qflex**

18 Página oficial de Xanadú Cloud: **https://www.xanadu.ai/cloud**

19 Página oficial de Honeywell: **https://www.honeywell.com/us/en**

20 Página oficial de IonQ: **https://ionq.com/**

Pero son muchas otras las empresas que apuestan por el desarrollo de ordenadores cuánticos, y no solo empresas dedicadas a la informática. Amazon también cuenta con una plataforma para desarrollar circuitos cuánticos, Intel se encuentra desarrollando procesadores cuánticos desde 2019. Volkswagen investiga aplicaciones cuánticas, Mitsubishi también [28].

CAPÍTULO 4

Ejecución de algoritmos

4.1 Descripción del experimento

Para comprender mejor cómo funciona la programación cuántica, vamos a mostrar la ejecución de los algoritmos presentados en el Capítulo 2 en tres entornos diferentes: resultado ideal, ejecución en un simulador de IBMQ y ejecución en un ordenador cuántico real de IBMQ. De esta forma podremos comprobar si los *hardwares* y simuladores son fiables y cuál es el porcentaje de error que presentan actualmente los procesadores, para hacernos una idea de en qué punto del desarrollo de esta tecnología nos encontramos.

Las implementaciones que vamos a usar están desarrolladas en Qiskit, ya que hemos escogido el framework y entorno de trabajo de IBMQ, porque la interfaz gráfica Quantum Composer facilita la comprensión a la hora de construir los circuitos de ejemplo. Para abreviar, nos referiremos a esta interfaz como Composer.

Si quisiéramos escribir el código de nuestros circuitos en algún IDE externo, tan solo tendríamos que añadir la *API key* generada por IBM Quantum y especificar el proveedor que queremos para la ejecución.

Haciendo pruebas en estas tres variantes veremos si los resultados obtenidos son correctos y coinciden, o qué diferencias presentan. Así, podremos corroborar si las implementaciones son correctas y si los *hardwares* son fiables.

4.2 Algoritmo de Deustch y Deustch-Jozsa

Este algoritmo, como hemos visto en el apartado 2.4.1, comprueba si una función es constante o balanceada dependiendo de si el valor de todas sus salidas es igual -sea 0, o 1- o si por el contrario aproximadamente la mitad de las salidas son 0 y la otra mitad 1. La aportación de Deustch-Jozsa generaliza este problema a n cúbits de entrada.

4.2.1 Implementación del algoritmo de Deustch

La construcción de este algoritmo se basa en el uso de una función personalizada a la que llamaremos *query function* $\boldsymbol{U_f}$ que decide si todos los posibles valores de salida son constantes o varían [33]. La *query function* aprovecha el paralelismo cuántico para realizar las comprobaciones en un solo paso. Para esto, debe recibir como entrada un estado de superposición de todas las entradas especificadas. Este estado lo conseguiremos aplicando puertas Hadarmard sobre cada cúbit de entrada.

La query function $\boldsymbol{U_f}$ puede implementarse de la siguiente forma:

$$\boldsymbol{U_f} \mid x\rangle = (-1)^{f(x)} \mid x\rangle .$$

Vamos a ver su aplicación sobre el estado $\mid -\rangle$ para comprender mejor su funcionamiento.

El estado $\mid -\rangle$ tiene la siguiente forma:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

Si le aplicamos $\boldsymbol{U_f}$, siguiendo la ecuación $\boldsymbol{U_f} \mid x\rangle$, obtenemos el estado:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle).$$

A partir de este momento el valor del estado será positivo o negativo en función de $\boldsymbol{U_f}$. A continuación vamos a suponer los diferentes valores que puede tomar.

Si $f(0) \neq f(1)$, entonces:

$$|\psi_3\rangle = \frac{\pm 1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Por el contrario, si suponemos $f(0) = f(1)$:

$$|\psi_4\rangle = \frac{\pm 1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

Una vez obtenidos los dos posibles estados, debemos revertir las puertas Hadamard que hemos utilizado para superponer los estados de los cúbits iniciales.

Tras aplicar Hadamard al cúbit que inicialmente su estado era $\mid 0\rangle$, procederemos a medir su valor.

Si medimos $\mid \psi_3\rangle$, obtendremos el estado $\pm \mid 1\rangle$. Asimismo, si medimos $\mid \psi_4\rangle$, el estado resultante será $\pm \mid 0\rangle$.

Luego, el algoritmo de Deustch nos dice que una función es constante si el resultado es cero, mientras que el resultado es 1 para una función balanceada. La mejora respecto a la computación clásica es evidente. En una sola evaluación comprobamos $f(0)$ y $f(1)$, cosa que, clásicamente, sería necesario realizar en dos evaluaciones.

Cabe remarcar que el algoritmo no nos proporciona información sobre cada valor concreto que puede tomar $f(x)$. Como hemos visto en el Capítulo 2, el paralelismo cuántico colapsa cuando medimos el estado en superposición.

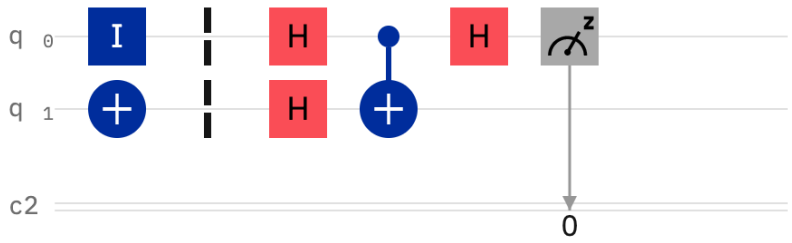


Figura 4.2: Circuito del algoritmo de Deustch.

En la Fig. 4.2 se muestra un esquema teórico del procedimiento que sigue este algoritmo, y en la Fig. 4.2 hemos transformado este esquema en un circuito implementado en el Composer de IBMQ utilizando las puertas cuánticas disponibles.

Vamos a ejecutar la implementación mostrada en la Fig. 4.2 y a observar los resultados obtenidos.

Ejecución en entorno ideal.

Nos interesa solamente el valor del primer cúbit, el que se encuentra más a la derecha.

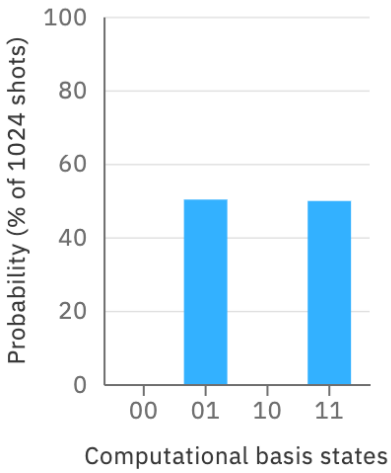


Figura 4.3: Histograma ideal del circuito de la Fig. 4.1.

Si nos fijamos en el primer cúbit de ambos resultados vemos que su valor es 1, por tanto se trata de una función balanceada.

Ejecución en un simulador.

En la plataforma de IBM podemos lanzar ejecuciones de nuestros circuitos a un simulador cuántico. Para las ejecuciones de este trabajo se ha escogido el simulador simulator_statevectorm que tiene una capacidad de 32 cúbits.

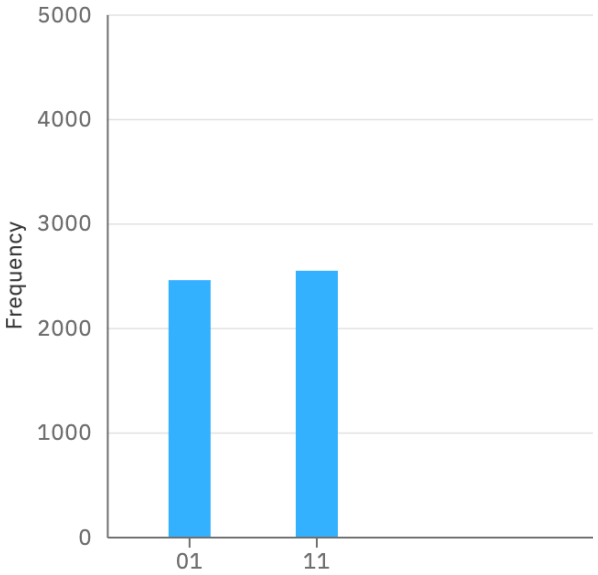


Figura 4.4: Histograma obtenido en la ejecución en el simulator_statevector de IBM del circuito de la Fig. 4.1.

Ejecución en un procesador cuántico real.

Para la ejecución en un entorno cuántico real vamos a escoger el procesador **ibmq_lima,** que cuenta con una capacidad de 5 cúbits. Vamos a ejecutar el circuito 5000 veces para obtener una mayor precisión en los resultados.

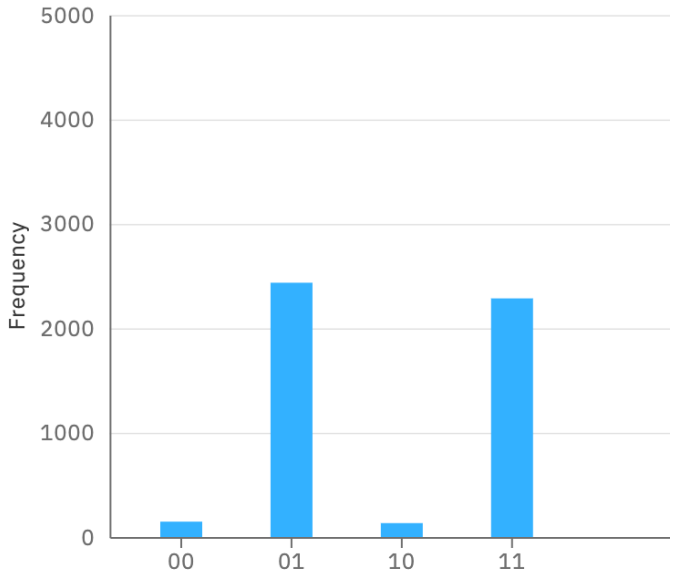


Figura 4.5: Histograma obtenido en la ejecución en procesador ibmq_limax5 del circuito de la Fig. 4.1.

En la Fig 4.5 podemos ver que al ejecutar el algoritmo en un entorno real aparecen pequeñas probabilidades para los valores erróneos. Esto se debe al ruido que todavía tienen los procesadores, que hace que los cúbits pierdan su coherencia y tomen valores no deseados. No obstante, el resultado es similar al deseado y al obtenido en un simulador.

4.2.2 Implementación de Deustch-Jozsa

La propuesta de mejora de Deustch-Jozsa [] se puede generalizar fácilmente siguiendo el esquema de la Fig 4.1, agrupando los n cúbits en el vector inicial de estado en superposición.

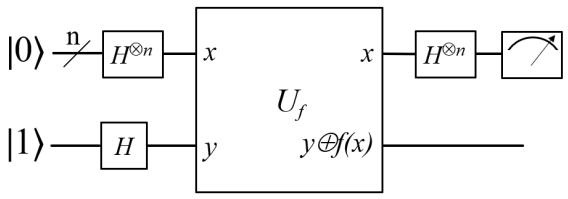


Figura 4.6: Esquema teórico del algoritmo de Deustch-Jozsa.

En este caso, la *query function* se beneficia mucho más del paralelismo cuántico, da igual cuántos cúbits tenga la entrada, que hará todas las evaluaciones en un solo paso igualmente.

Vamos a tomar como ejemplo una función $f\,(x_1, x_2, x_3)$, con talla $n = 3$, que consta de $2^3 = 8$ combinaciones de valores posibles, y para la que necesitaremos 3 cúbits de entrada para implementar su circuito. Además necesitaremos un registro auxiliar para almacenar $f\,(x_1, x_2, x_3)$ y un registro clásico de 3 bits para volcar los resultados de cada cúbit. El procedimiento a realizar es el mismo que para la versión primitiva:

1) Preparar el estado inicial en superposición aplicando puertas Hadamard a los n cúbits y obtenemos una probabilidad igual para todas las combinaciones.
2) Aplicar la *query function* $\boldsymbol{U_f}$ al estado obtenido.
3) Aplicar de nuevo puertas Hadamard a los n cúbits para revertir la superposición de estados con las probabilidades obtenidas tras $\boldsymbol{U_f}$.
4) Medir los resultados obtenidos tras colapsar los estados.

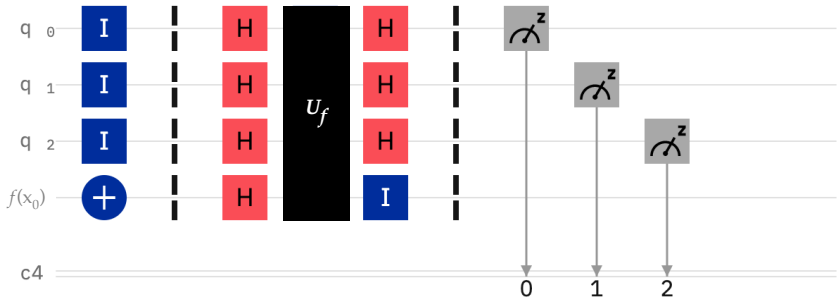


Figura 4.7: Circuito del algoritmo de Deustch-Jozsa para $n = 3$.

En el circuito de la Fig. 4.6 hemos añadido el paso de la *query function* como una "caja negra" porque su implementación va a variar dependiendo de si la función que estamos evaluando el constante o balanceada.

Vamos a considerar solamente que la función $f(x)$ que queremos analizar es balanceada. Para este caso, el mecanismo de funcionamiento de U_f será:

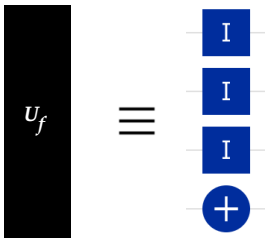


Figura 4.8: Mecanismo interno de U_f para una función balanceada.

Ejecución en entorno ideal.

Al tratarse del caso de una función balanceada, el resultado que debemos esperar es 1. Es decir, que el resultado -o resultados- que obtengamos con mayor probabilidad tengan al menos uno de sus cúbits con valor 1. Si fuera constante, todos los cúbits tendrían valor 0.

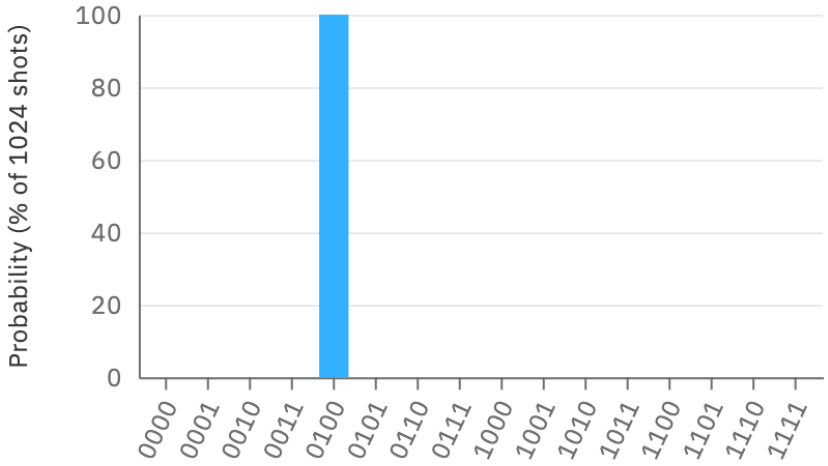


Figura 4.9: Histograma ideal del algoritmo Deustch-Jozsa sobre una función balanceada para $n = 3$.

La simulación ideal que ofrece el Composer nos muestra información adicional sobre los resultados.

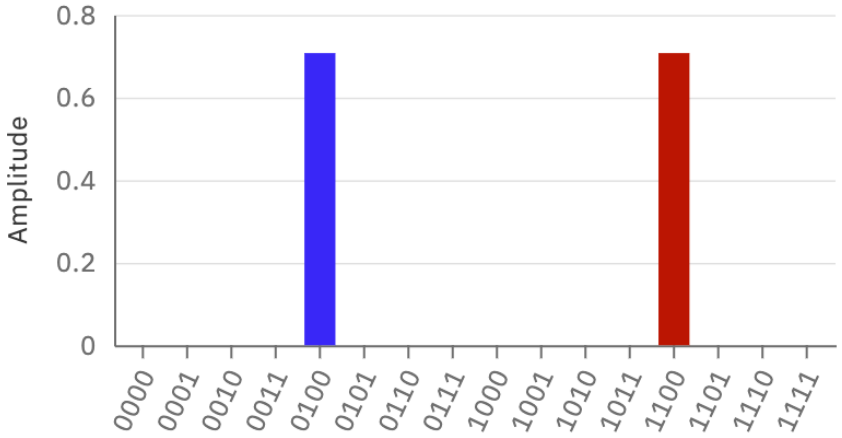


Figura 4.10: Histograma del vector de estado del algoritmo Deustch-Jozsa sobre una función balanceada para $n = 3$.

La Fig. 4.9 corresponde al vector de estado resultante, en el que se puede observar que la probabilidad del estado $|1100\rangle$ se muestra en rojo. Esto se debe a que se encuentra en fase π, o lo que es lo mismo, su amplitud tiene un valor negativo. Mientras que el estado $|0100\rangle$ se encuentra en fase 0.

Adicionalmente, podemos ver la representación de ambos estados sobre la esfera de Bloch.

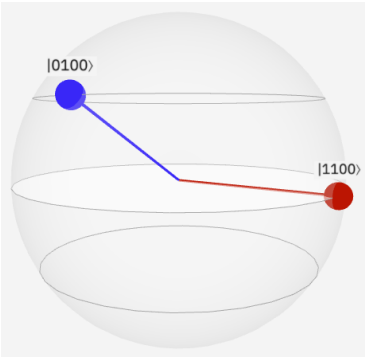


Figura 4.11: Amplitudes del vector de estado resultante representadas en la esfera de Bloch.

Ejecución en un simulador.

Los resultados de la ejecución en el simulador `simulator_statevector` son los esperados idealmente. No se observan estados con probabilidades pequeñas, como podríamos esperar en los simuladores.

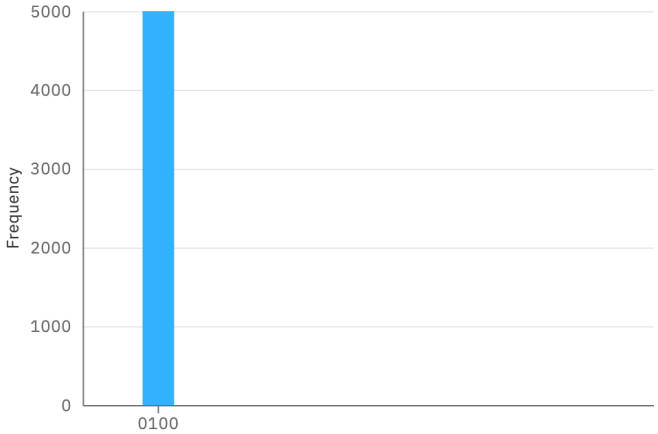


Figura 4.12: Histograma del algoritmo de Deustch-Jozsa balanceada para n =.3 obtenido en un simulador IBMQ.

Ejecución en un procesador cuántico real.

Para este algoritmo vamos a usar el procesador `ibmq_lima` de nuevo, y vamos a ejecutar el circuito 5000 veces también.

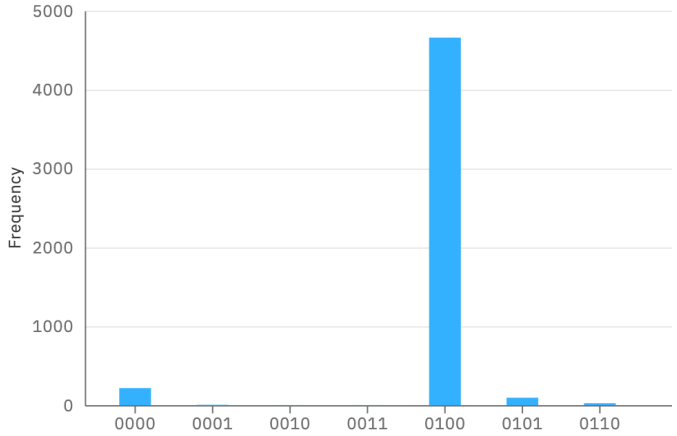


Figura 4.13: Histograma del algoritmo de Deustch-Jozsa balanceada para n =.3 obtenido en ibmq_limax5 IBM.

En los resultados obtenidos (Fig. 4.13) vemos como han aparecido probabilidades casi despreciables en otros estados. Los estados $|0101\rangle$ y $|0110\rangle$ son estados válidos para nuestra comprobación, porque tienen cúbits con valor 1. Sin embargo, también se muestra una probabilidad pequeña para el estado $|0\,0\,0\,0\rangle$, y este es el estado erróneo que no representa a una función balanceada. La causa de la aparición de estos resultados es el ruido cuántico intrínseco del procesador real, que todavía no logra mitigar todo el ruido y como resultado el algoritmo falla en algunas ocasiones.

En general, este algoritmo no es de mucha utilidad, ya que debemos presuponer que la función que se va a evaluar es, o constante, o balanceada, y no es de ningún otro tipo. Además, nos obliga a implementar una *query function* diferente para ambos casos, por tanto no es un algoritmo del todo generalizado. No obstante, la mejora de eficiencia es notable porque las evaluaciones de $\boldsymbol{U_f}$ para cada valor de $f(x)$ se realizan simultáneamente gracias al uso de puertas Hadamard, y mejora exponencialmente el coste temporal. En su momento, fue un descubrimiento esclarecedor acerca del potencial que tiene la computación cuántica.

4.3 Algoritmo de Grover

En el apartado 2.4.2 hemos mencionado que este algoritmo es útil para las llamadas *unestructured search.* Es el caso de la búsqueda de elementos en una base de datos desordenada. Recordemos que la mejora este algoritmo respecto a la búsqueda convencional es evidente -de orden cuadrática- porque tan solo necesita $O(\sqrt{N})$ operaciones para encontrar un resultado, mientras que en una búsqueda clásica se necesitan de promedio $O(N)$ operaciones al tener que comprobar todos los elementos uno por uno.

4.3.1 Implementación del algoritmo de Grover

Como hemos visto, este algoritmo se respalda en una función **oráculo** O, que recibe como entrada un estado en superposición de los estados iniciales e invierte la amplitud del estado buscado, y un operador de difusión o **amplificador** que destaca esta amplitud sobre el resto.

El procedimiento completo a seguir se define en cuatro pasos:

1) Inicializar el estado en superposición mediante puertas Hadamard para obtener la misma probabilidad en todos los estados.

2) Aplicar el **oráculo** *O* al estado en superposición para marcar con probabilidad negativa el estado correspondiente al elemento buscado, tal que:

$$O\,|x\rangle = -|x\rangle$$

3) Aplicar el operador de difusión o **amplificador** al nuevo estado en superposición, para hacer una inversión sobre la media y aumentar el valor de la amplitud del estado buscado.

Los pasos *2)* y *3)* es necesario repetirlos $\sqrt{N}$ veces, siendo N la longitud del conjunto de elemento sobre el que buscamos. Al hacer esta iteración, la amplitud del estado buscado aumenta su valor distinguiéndose más.

4) Medir el estado en superposición obtenido, que colapsará en el estado buscado ya que hemos amplificado su probabilidad hasta un valor lo suficientemente elevado.

El esquema teórico de este procedimiento se representa de la siguiente forma:

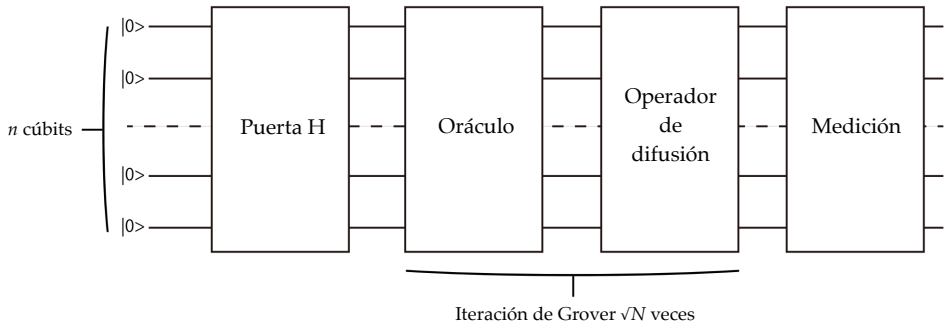


Figura 4.14: Esquema semántico de los pasos del algoritmo de Grover para *n* cúbits.

Vamos a considerar un conjunto de 16 elementos, para el que serán necesarios $\sqrt{16}=4$ cúbits para implementar el circuito. Y el elemento que queremos encontrar es el estado $|1111\rangle$.

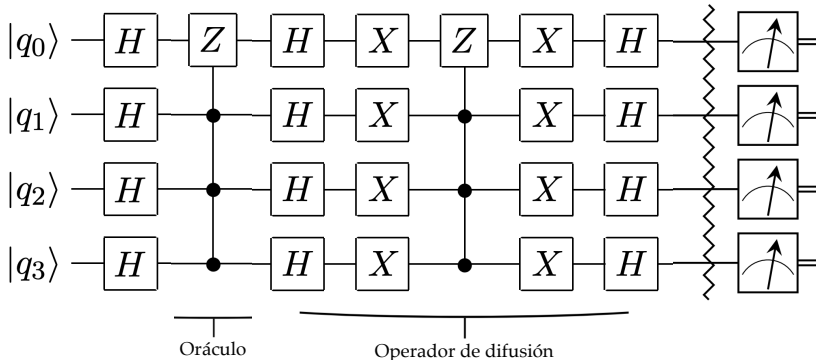


Figura 4.15: Implementación parcial del circuito del algoritmo de Grover para 4 cúbits.

La Fig. 4.15 corresponde a la implementación que vamos a construir en el Composer, construyendo en oráculo que encuentra el estado $|1111\rangle$ con una puerta *CCCZ*. En el apéndice F se muestran las implementaciones para el resto de estados del espacio n=4.

Vamos a tener que hacer una transformación previa sobre la puerta *CCCZ*, ya que IBMQ no cuenta por el momento con una implementación por defecto para la *triple-controlada Z*.

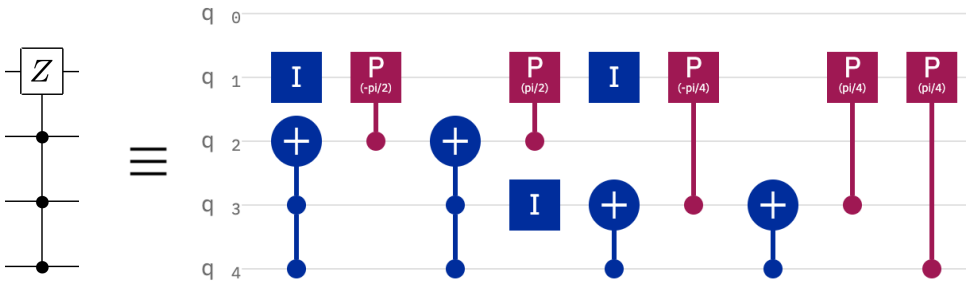


Figura 4.16: Adaptación de la puerta *CCCZ* a las puertas disponibles en IBMQ Composer [34].

La implementación para *CCCZ* la vamos a usar tanto en el oráculo como en el operador de difusión.

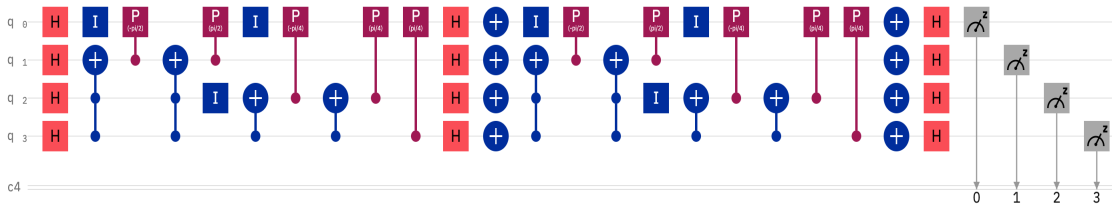


Figura 4.17: Implementación del circuito completo del algoritmo de Grover para 4 cúbits.

Cabe remarcar que la Fig 4.17 es una implementación simple del algoritmo, porque solo se hace una iteración de Grover. Así que los resultados de este circuito no mostraran una probabilidad alta para el estado que buscamos, sino que mostrará probabilidades similares para todos los estados. Para solucionarlo, vamos a añadir 3 iteraciones más, para obtener las 4 iteraciones necesarias para un conjunto de 16 elementos.

Ejecución en entorno ideal.

Considerando el circuito con 4 iteraciones, el resultado que esperamos obtener es una probabilidad casi del 100% para el estado $|1111\rangle$ y alguna probabilidad despreciable para algún otro estado.

Si nos fijamos, en la Fig 4.17 se muestran probabilidades muy próximas a 0 para los estados que no son el buscado, pero son despreciables en comparación a la alta probabilidad del que estamos buscando.

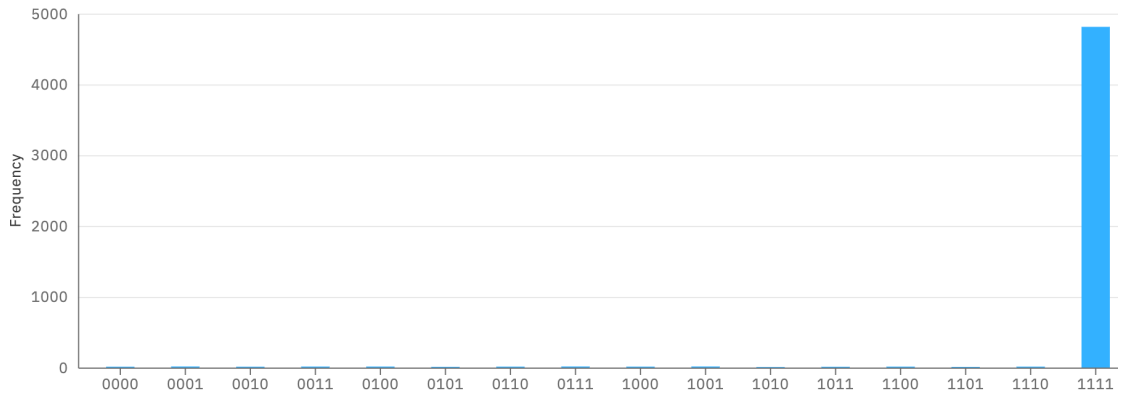


Figura 4.18: Histograma ideal del algoritmo de Grover para 4 cúbits.

Ejecución en un simulador.

Ejecutamos 5000 veces el circuito en el simulador simulator_statevector de IBMQ y obtenemos el siguiente resultado.

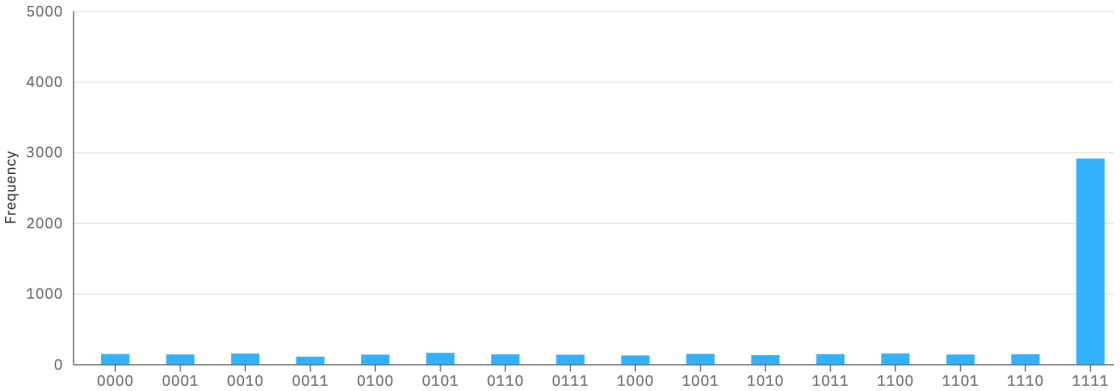


Figura 4.19: Histograma de la ejecución del algoritmo de Grover en un simulador de IBM para n=4.

Las probabilidades que idealmente serían casi nulas ahora tienen un mayor peso en el resultado, alcanzando en promedio el 3%. Aproximadamente, de 5000 veces que se ha ejecutado el circuito, entre 140 y 160 veces se ha obtenido como resultado cada valor erróneo. No obstante, se sigue manteniendo una clara predisposición hacia el valor correcto.

Paralelamente, hemos construido el circuito con 2 y 3 iteraciones con el objetivo de analizar si las probabilidades varían a mejor o a peor en función del número de iteraciones añadidas. En el caso de 3 iteraciones, la probabilidad obtenida para el estado que buscamos alcanza el 96,58%, un valor lo suficientemente elevado como para reducir el resto de estados a probabilidades nulas o por debajo de 0,5%.

Por tanto, podemos decir que es recomendable implementar 3 iteraciones para el caso en que $\sqrt{N} \simeq 4$. Esto nos muestra que no debemos seguir estrictamente los cálculos propuestos para el algoritmo y, si no obtenemos unos resultados del todo precisos, es recomendable re-calcular el algoritmo con una iteración de más o de menos [35].

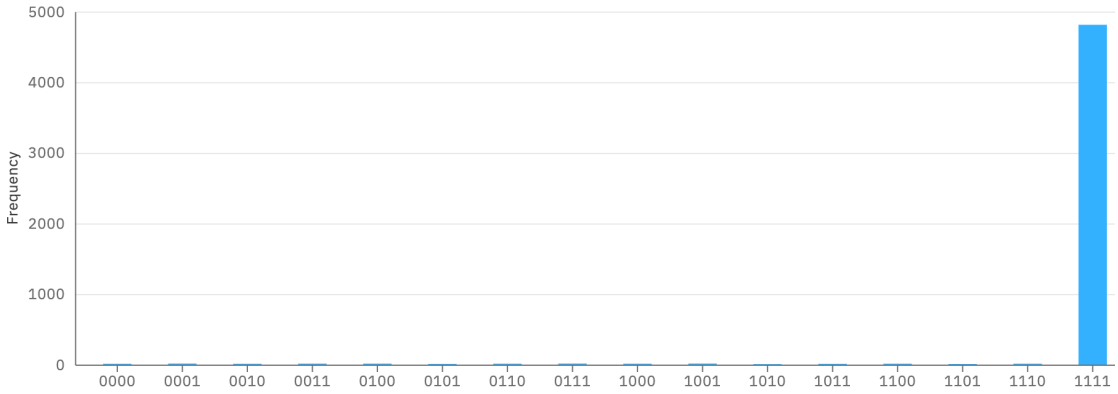


Figura 4.20: Histograma de la ejecución del algoritmo de Grover en un simulador de IBM para n=4, 3 iteraciones.

Ejecución en un procesador cuántico real.

Pasamos a observar los resultados obtenidos en el mismo procesador que para los algoritmos anteriores, y contrastar también la diferencia entre escoger 3 o 4 iteraciones de Grover.

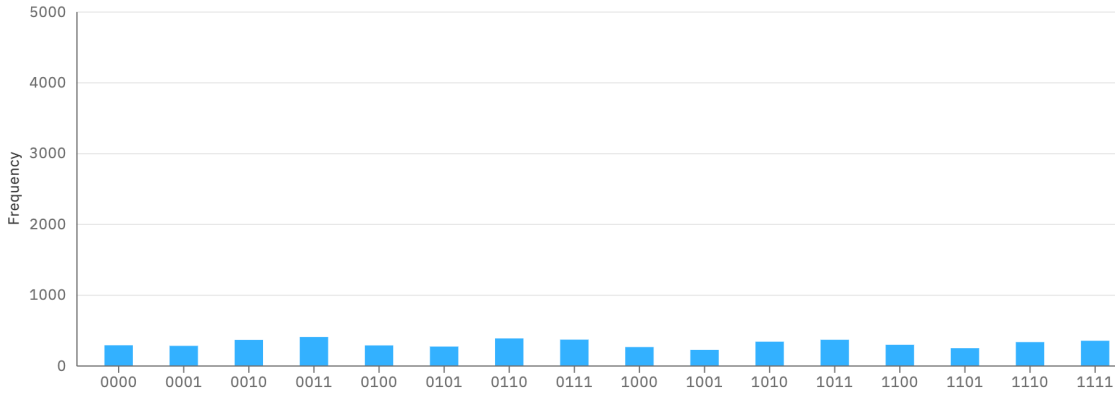


Figura 4.21: Histograma de la ejecución del algoritmo de Grover en `ibmq_bogota` para n=4, 3 iteraciones.

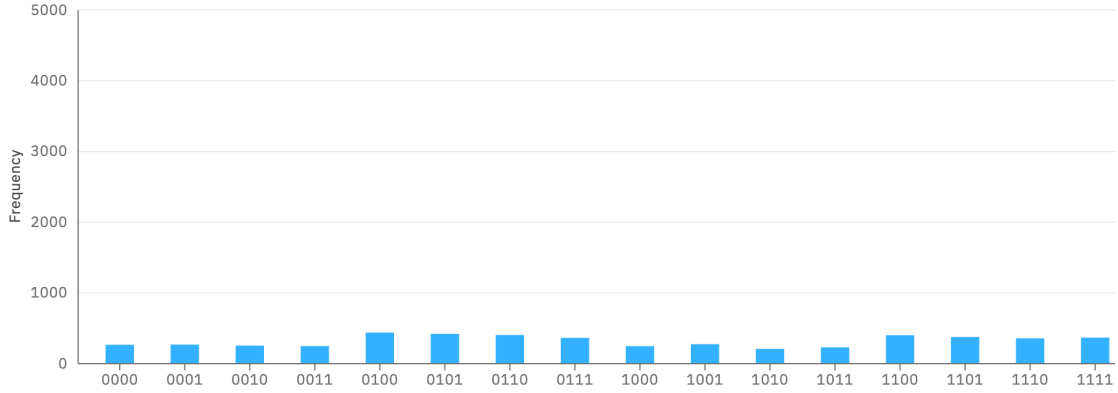


Figura 4.22: Histograma de la ejecución del algoritmo de Grover en `ibmq_bogota` para n=4, 4 iteraciones.

Contrariamente a lo que esperábamos, ninguno de los circuitos con 3 y 4 iteraciones nos proporciona un resultado similar al esperado. En las figuras Fig. 4.21 y Fig. 4.22 vemos que las probabilidades se distribuyen demasiado equitativamente entre todos los posibles estados, con lo que no podemos escoger ninguno como resultado principal. Para tener una visión más amplia, también se ha realizado la ejecución para 1, 2 y 5 iteraciones y los resultados son similares.

Esto no nos ha pasado en las ejecuciones realizadas en simulador, por tanto podemos pensar que la implementación del algoritmo es correcta. Sin embargo, el hecho de que en un entorno cuántico real no se pueda extraer ningún resultado principal -sea el correcto o no- nos lleva a pensar que el diseño de este algoritmo genera cierto ruido cuántico en los cúbits que hace que por tanto pierde la coherencia cuántica en el cálculo. Además, a esto se le debe sumar el error cuántico de cada puerta, que aumenta cada vez que añadimos una iteración.

Resumiendo, el algoritmo de Grover significa una mejora considerable en la búsqueda sobre datos desordenados al no tener la necesidad de ordenar previamente los datos. No obstante, hay que tener en cuenta que para poder aplicar este algoritmo se ha de cargar el conjunto de datos -base de datos- en los cúbits, aunque una vez cargada, se puede realizar tantas búsquedas como queramos.

Pero debemos tener en cuenta que los resultados obtenidos en un procesador real de los que tenemos a nuestro alcance por el momento no son fiables. Y, en general, el resultado obtenido siempre es probabilístico, por tanto siempre habrá un porcentaje de error, que deberemos reducir el máximo posible añadiendo o reduciendo el número de iteraciones.

Además, la plataforma de IBMQ está limitada en la creación de puertas de triple control, lo que nos obliga a transformar la puerta CCCZ en una equivalencia con puertas más simples. Este hecho nos limita también a la hora de implementar el algoritmo para conjuntos de datos más grandes, pues se requiere una puerta de control de mayor tamaño e implementarla con las puertas disponibles de IBMQ supone una pérdida de conectividad entre los cúbits.

Por tanto, este algoritmo es óptimo para su ejecución en simuladores, pero los procesadores de IBMQ que disponemos todavía no están preparados para este cálculo.

4.4 Algoritmo de Shor

El algoritmo de Shor, descrito en el apartado 2.4.3, es el último algoritmo que va a ser objeto de estudio. Debido a su importancia en el salto del desarrollo computacional, es el algoritmo principal que motiva este trabajo.

4.4.1 Implementación del algoritmo de Shor

Para el diseño de este algoritmo, hemos mencionado que necesitamos aplicar una función $f(x) = a^{2^x} \bmod N$, para calcular el periodo r de esta misma, siendo a un número $a < N$ que no tiene factores comunes con N. Recordemos que a lo escogemos aleatoriamente, empezando preferiblemente por los valores más simples a partir de

1. Este paso de calcular $f(x)$ es el que desarrollaremos de forma cuántica, así expandimos -teóricamente hablando- la función para N valores de x.

El paso inicial de calcular $MCD(a,N)$ y el paso final de resolver $(a^{r/2}-1)$ y $(a^{r/2}-1)$, como podemos deducir, no precisan de realizar ningún cálculo cuánticamente, y por tanto quedan fuera del diseño del circuito.

La parte en la que sí se se hace uso de la potencia cuántica se divide en cinco pasos, y hace uso de dos registros cuánticos: uno para implementar los cálculos de $f(x)$ -*R1*-, y otro para almacenar los resultados y obtenerlos -*R2*-.

Los cincos pasos a seguir los definimos de la siguiente forma:

1) Inicializar *R2* a $|0\rangle$, e invertir $R1_0$ al valor $|1\rangle$.

2) Usar puertas Hadamard sobre *R2* para ponerlo en estado de superposición.

3) Construir la función $f(x)$ sobre *R1* usando puertas controladas que implementen la multiplicación modular, y aplicar esta función sobre *R2*.

4) Para devolver *R2* a un estado de no superposición y poder obtener el resultado, aplicar una puerta QFT -Transformada de Fourier- sobre *R2*.

5) Medir el resultado obtenido en *R2*.

A continuación veremos tres variantes de la implementación de los pasos anteriores y las aportaciones de cada una a los resultados obtenidos. El caso de estudio que vamos a considerar es $N = 21$, para el que escogeremos valores diferentes para a en función del número de cúbits del que disponen las máquinas a las que tenemos acceso en IBMQ.

La primera versión, propuesta por Vandersypen [37], es la mas costosa en espacio de cúbits. Se requieren $log_2(N)$ cúbits para el registro *R1*, y $log_2(N)$ cúbits para el registro *R2*. Esto dificulta la escalabilidad del circuito para números muy grandes debido a la limitación que supone el hardware actualmente. Por ejemplo, para factorizar $N = 47*51 = 2397$ serían necesarios $n = 2log_2(N) = 2log_2(2397) =$ 23 cúbits. Si quisiéramos resolver una N de más de diez cifras decimales el circuito debería tener más de 90 cúbits. Y por el momento no se ha construido un procesador cuántico estable y tolerante a errores de esas dimensiones.

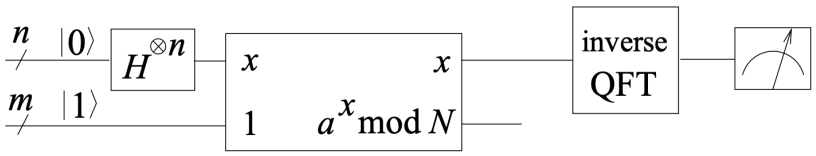


Figura 4.23: Esquema semántico de los pasos del algoritmo de Shor.

La Fig. 4.23 muestra el diseño del circuito descrito en los pasos *1), 2), 3) y 4)*. Cada una de las líneas representa un registro. El de arriba es el registro de resultados *R2*, implementado con n cúbits, y el de abajo es el registro de cálculo *R1*, implementado

con m cúbits. La caja grande de en medio corresponde al paso 3). La construcción de esta dependerá del valor de a que escojamos, pues se usan puertas multiplicadoras modulares diferentes.

Para el caso $N = 21$, el circuito debe estar formado por 10 cúbits, 5 para *R1* y 5 para *R2*. Vamos a escoger $a = 8$, que es par y no tiene factores comunes con 21, por tanto $f(x) = 2^{2^x} mod N$. Previamente a implementar $f(x)$ debemos calcular las multiplicaciones resultantes de la ecuación para saber qué puertas son necesarias.

X	$2^x \ mod \ 21$
1	2
2	4
4	16
8	4
16	16

Tabla 2: Cuadro de resultados de la función modular 2^x mod 21.

Como la representación binaria de 21 requiere 5 bits, calculamos solo los cinco primeros valores. Así que para construir *R1* es necesario multiplicar por 2, 4 o 16. Usaremos una puerta multiplicadora por dos, una puerta multiplicadora por cuatro y una puerta multiplicadora por dieciséis.

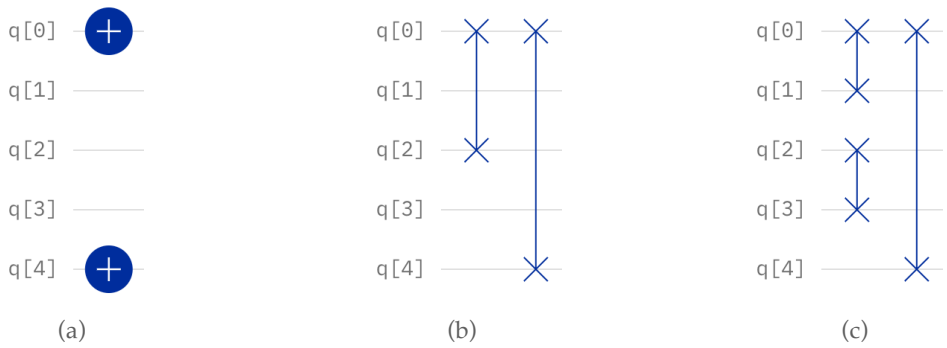


Figura 4.24: Circuitos implementados para las puertas multiplicadoras. (a) Multiplicador modular por 16. (b) Multiplicador modular por 4. (c) Multiplicador modular por 2.

Con estas tres puertas construimos sobre *R2* una secuencia de cálculo siguiendo el orden inverso de los resultados de $f(x)$. El orden a seguir será: 16, 4, 16, 4, 2.

Para finalizar el cálculo, es necesario añadir una QFT de 5 cúbits que actúe sobre el registro *R1*.

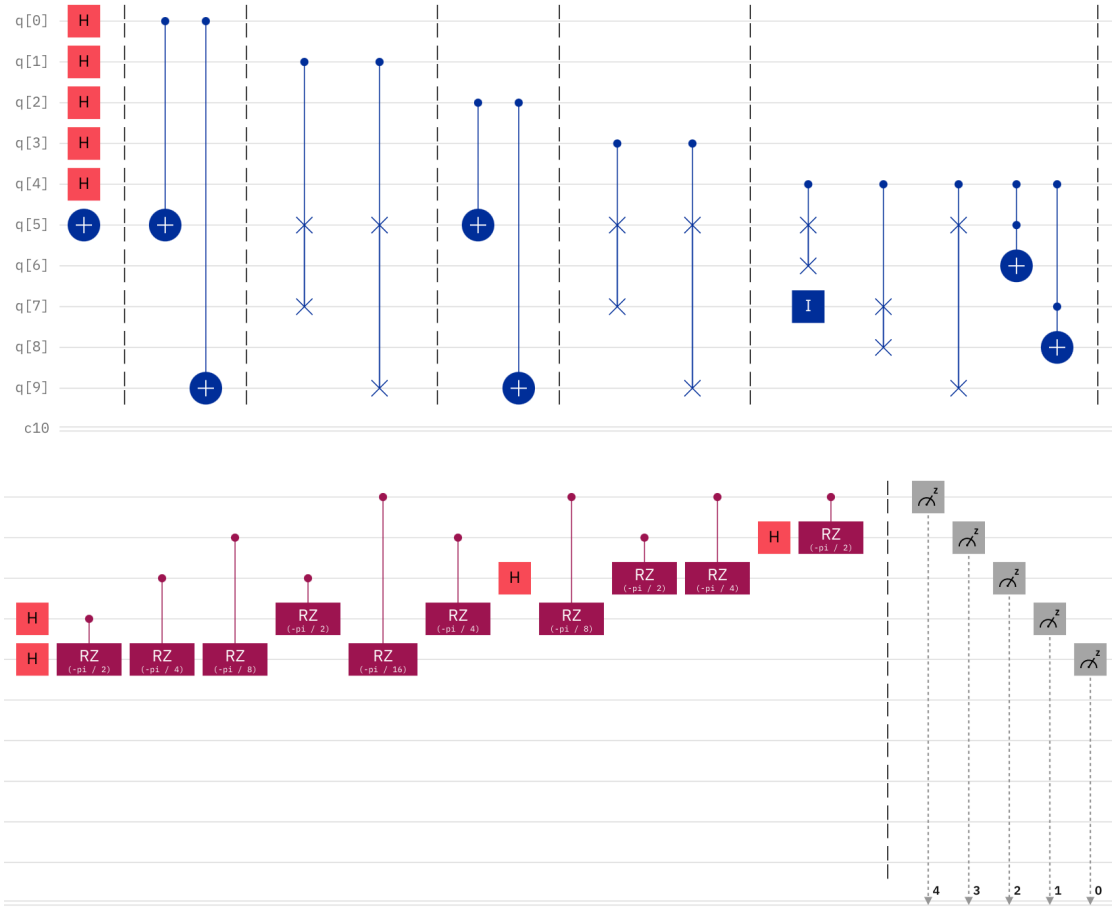


Figura 4.25: Implementación completa en Qiskit del algoritmo de Shor para N=21, a=2.

En el circuito de la Fig. 4.25 se muestra la secuencia completa de pasos a seguir, donde todos los pasos están separados por barreras. El registro *R2* abarca los cúbits $q_{0...4}$ inicializados al estado $|00000\rangle$, y el resto de cúbits $q_{5...9}$ pertenecen a *R1* inicializados al estado $|00001\rangle$.

Las puertas básicas que construyen las puertas multiplicadoras tienen como cúbit de control alguno de los cúbits de *R2*, empezando por el menos significativo para la primera puerta. Con esto se consigue que el registro *R1* influya sobre los valores de *R2*.

Tras aplicar la QFT sobre *R1*, se obtienen resultados estadísticos que se distribuyen de forma más o menos uniforme. Deberemos escoger los valores para r con mayor probabilidad para realizar los cálculos posteriores, y ver si r es un periodo válido. En caso de que ningún valor de r sea adecuado, deberemos volver al paso *1)* y volver a realizar los cálculos para otra a.

Ejecución en entorno ideal.

Una estadística resultante ideal marca una clara diferencia de uno o dos valores para r con el resto de posibles valores. Esto se observa cuando el valor correcto de r aparece en varios estados si las probabilidades están muy distribuidas, o en un solo estado con una probabilidad muy elevada.

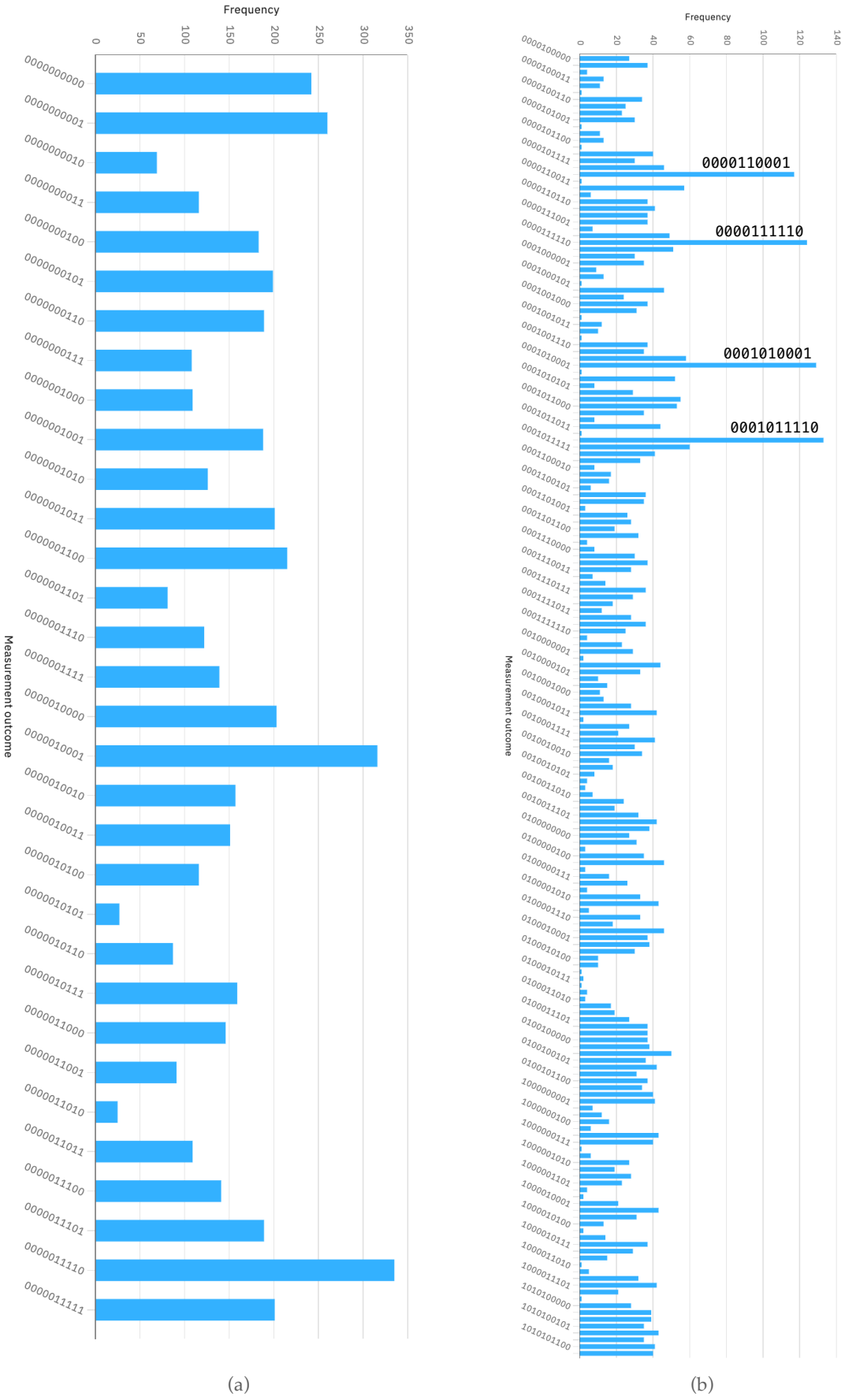


Figura 4.28: Histogramas de resultados de Shor N=21, a=2 en el simulador simulator_statevector. (a) Circuito sin medir el registro *R1*. (b) Circuito modificado que también mide *R1*.

Para el caso N=21, a=2 el periodo r correcto es r=6, de forma que si realizamos los cálculos clásicos posteriores al circuito:

$$(a^{r/2} + 1) = (a^3 + 1) = 9$$
$$(a^{r/2} - 1) = (a^3 - 1) = 7$$

$$MCD(9, 21) = 3$$
$$MCD(7, 21) = 7$$

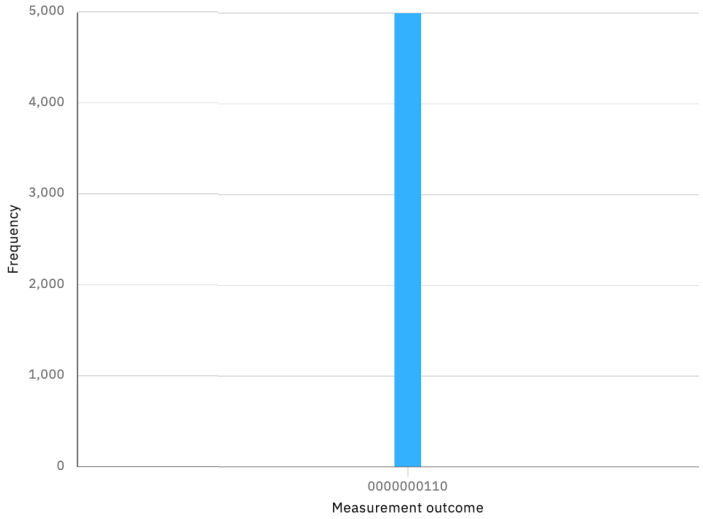


Figura 4.26: Resultados ideales para Shor N=21, a=2.

Ejecución en un simulador.

En el circuito construido en la Fig. 4.24 no hemos medido los resultados de *R1*, por tanto no estaríamos teniendo en cuenta todo el espacio de la distribución estadística de resultados. Aunque los cúbits que nos interesa observar sean los del registro *R2*, vamos a añadir la medición de los cinco cúbits de *R1* en un paso intermedio entre los pasos *3)* y *4)* -antes de aplicar la QFT-.

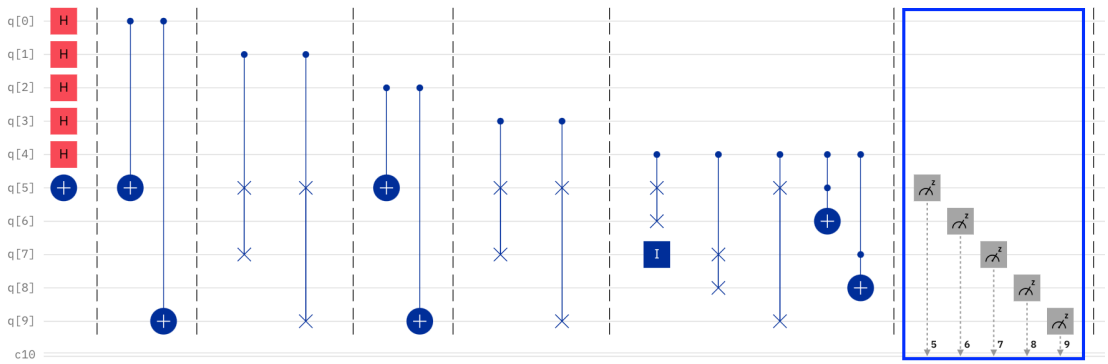


Figura 4.27: Agregación del paso intermedio de medición de R1.

De nuevo, se van a realizar 5000 ejecuciones para obtener resultados significativos.

En ambos histogramas de la Fig. 4.28 se ve que las probabilidades se distribuyen entre todos los estados combinatorios para el número de cúbits que observamos. Los resultados obtenidos en Fig. 4.28(a) están englobados en Fig. 4.28(b). En ambos casos hay ciertos estados que tienen una probabilidad más destacable que el resto.

En primer lugar, observamos que en Fig. 4.28(a) los estados con mayor probabilidad son $|0000010001\rangle$ y $|0000011110\rangle$. Es decir, que los posibles valores para el periodo *r* que estamos buscando son *r*=15 y *r*=30.

En Fig. 4.28(b), hay cuatro estados que resaltan sobre el resto: $|0000110001\rangle$, $|0000111110\rangle$, $|0001010001\rangle$ y $|0001011110\rangle$. Si nos fijamos solamente en los cinco bits menos significativos, los valores coinciden con los observados en Fig. 4.28(a). Sin embargo, midiendo los cúbits de R1 conseguimos una muestra más significativa. Podemos ver la suma de probabilidades de los estados que representan *r*=30 es mayor que la suma de las de *r*=15.

Una vez obtenidos los posibles valores del periodo *r*, comprobamos si son valores válidos para el algoritmo o no. El caso *r*=15, al tratarse de un número impar, no es un valor aceptable para seguir con el resto de cálculos. Si no hubieran más posibles valores a considerar, el algoritmo hubiese fallado para la *a* que estamos tratando y deberíamos repetir el proceso escogiendo otra *a*. Como en este caso sí que existe otro posible periodo y se trata de un número par, procedemos a realizar los cálculos posteriores y ver cuáles son los factores obtenidos.

$$(a^{r/2} + 1) = (a^{15} + 1) = 32769$$
$$(a^{r/2} - 1) = (a^{15} - 1) = 32767$$

$$MCD(32769, 21) = 3$$
$$MCD(32767, 21) = 7$$

Como los factores obtenidos son 3 y 7, el periodo *r*=30 es válido y el procedimiento ha sido correcto. Aunque *r*=30 era el periodo ideal esperado, como 30 es múltiplo de 6, es un resultado válido. Además, 30 / 6 = 5, que son los cúbits que hemos empleado para medir el registro *R2*, o lo que es lo mismo, la longitud del binario que representa *N*=21.

Cuando los cálculos finales fallan para un determinado periodo, se puede comprobar si alguno de sus múltiplos obtiene los factores buscados -en caso de ser un número no divisible por *n*, siendo n el tamaño de *R2*-.

Ejecución en un procesador cuántico real.

Debido a que solo disponemos de procesadores cuánticos de cinco cúbits en IBMQ, no es posible ejecutar la versión propuesta por Vandersypen. Sin embargo, existen otras implementaciones para este algoritmo, que optimizan el procedimiento que acabamos de explicar.

Según la implementación que propone Kitaev [36], el registro *R2* puede reducirse a un solo cúbit, lo que reduce el tamaño del circuito a *n*+1, donde *n* es la longitud de la representación binaria de *N*.

Este diseño requiere hacer la secuencia de cálculo de *R1* reiniciado *R2*, aplicando Hadamard y midiendo el resultado de forma iterativa, con la particularidad de que el

resultado que se almacena en la iteración anterior se usa como control para el nuevo valor de *R2*. Además elimina la QFT y utiliza una QFT semi-clásica controlada por los resultados anteriores, que actúa sobre un único cúbit. Esto que reduce la complejidad de implementación del paso *4)*. No obstante, cuando el número de iteraciones crece al tratar valores de *N* grandes, construir esta QFT semi-clásica se vuelve una tarea tediosa, debiendo usar cada vez más resultados para controlar las puertas QFT semi-clásicas.

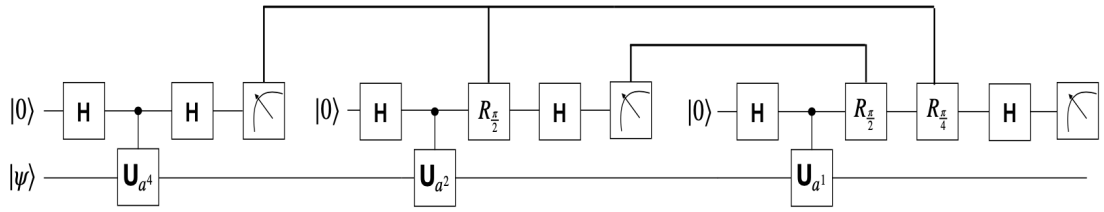


Figura 4.29: Circuito ideal del algoritmo de Shor, versión propuesta por Kitaev [36].

Desafortunadamente, este circuito tampoco podemos ejecutarlo dentro de nuestras posiblidades, debido a que IBMQ no permite realizar condicionales en tiempo real y no podemos reutilizar valores obtenidos en otro instante de tiempo que no sea el actual.

Como última opción, Skosana & Tame [39] proponen una optimización un paso más allá, haciendo necesarios solamente dos cúbits para *R1* y usando los tres cúbits restantes para el registro de lectura *R2*. Esta reducción de cúbits es posible porque f(x) se modifica de forma que el cálculo que deben implementar las puertas multiplicadoras es $f'(x) = \log_a(a^x \bmod N)$. Y el registro *R1* por tanto necesita solamente $\log_a(\log_2 N)$ cúbits.

Si seguimos con el caso $a = 2$ nos damos cuenta enseguida de que no podemos cumplir que *R1* los conformen dos cúbits, porque $\log_2(\log_2 21) = 2{,}32$. Lo que significa que para construir las puertas multiplicadoras serán necesarios tres cúbits.

Así que para conseguir realizar una ejecución en un entorno real escogemos $a = 4$, que es válido porque $\log_4(\log_4 21) = 1{,}13$, es decir, solo son necesarios dos cúbits en *R1*.

En este caso, la tabla de resultados de *f(x)* obtiene:

X	$4^x \bmod 21$
0	1
1	4
2	16
4	4
8	16
16	4

Tabla 3: Cuadro de resultados de la función modular $4^x \bmod 21$.

Los resultados que debemos considerar par construir las puertas multiplicadoras son 1, 4, y 16. Esta vez consideramos también el caso x = 0 para completar el volcado de los cálculos sobre tres cúbits. El siguiente paso es calcular el valor de *f'(x):*

$$
\begin{aligned}
|1\rangle &\mapsto |\log_4 1\rangle = |00\rangle, \\
|4\rangle &\mapsto |\log_4 4\rangle = |01\rangle, \\
|16\rangle &\mapsto |\log_4 16\rangle = |10\rangle.
\end{aligned}
$$

Así que las puertas que consideramos en realidad calculan $\log_4 1$, $\log_4 4$ y $\log_4 16$, que corresponden con U^1, U^2 y U^4, respectivamente.

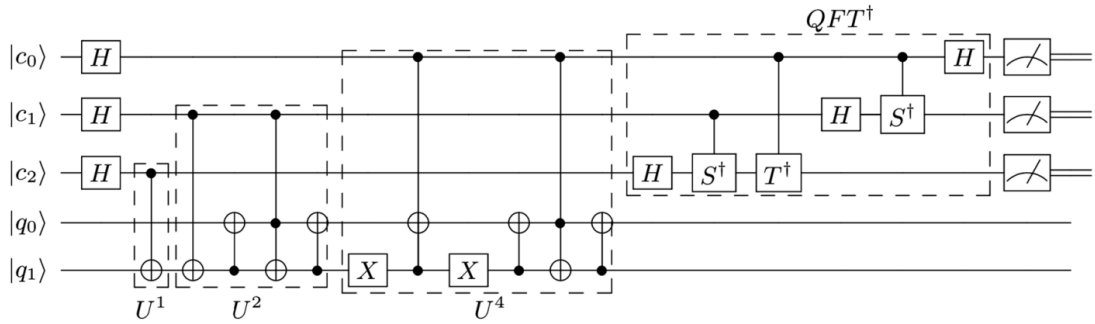


Figura 4.30: Circuito ideal del algoritmo de Shor, versión propuesta por Skosana & Tame [39].

El orden en el que se deben colocar las puertas en esta variante del algoritmo es el inverso al propuesto en el diseño de Vandersypen, es decir, orden creciente. Y la QFT final se construye sobre tres cúbits. La medición de resultados se vuelca en el orden de índices de *R2* -nombrado como c_i en la Fig. 4.29-.

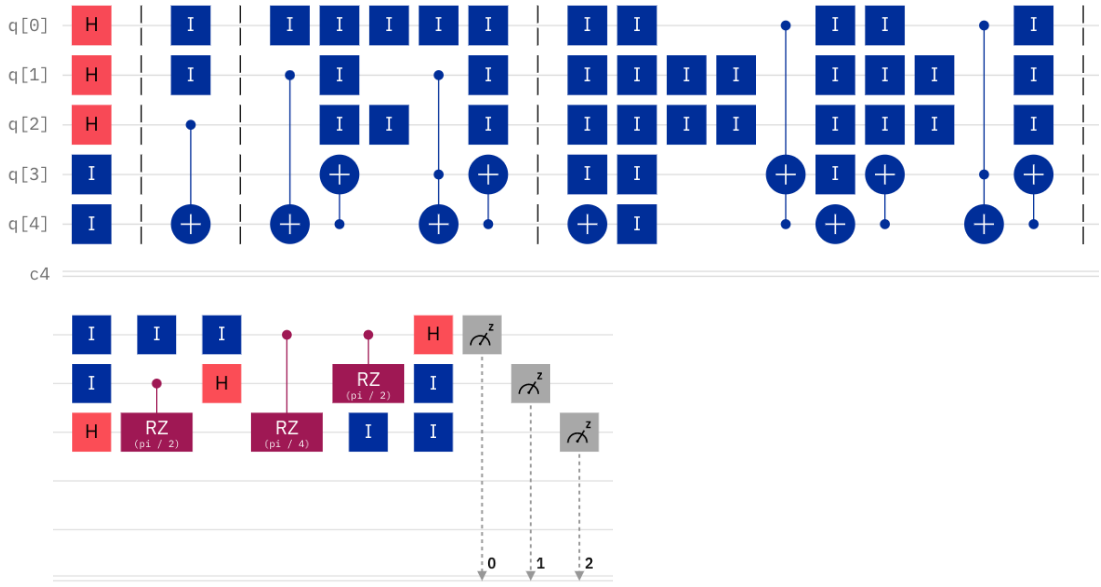


Figura 4.31: Implementación completa en Qiskit del algoritmo de Shor, versión propuesta por Skosana & Tame [39].

El hecho de que en el circuito se hayan agregado tantas puertas identidad se debe a que la interfaz del Composer lo requiere para poder colocar las puertas en el sitio correcto. Lo que es un hecho que nos beneficia porque estas puertas hacen que se mantenga el estado que posea el cúbit a lo largo de los instantes de tiempo en los que no se actúa sobre él. De esta forma, conseguimos mitigar -mínimamente- la decoherencia cuántica que se produce en los procesadores cuánticos.

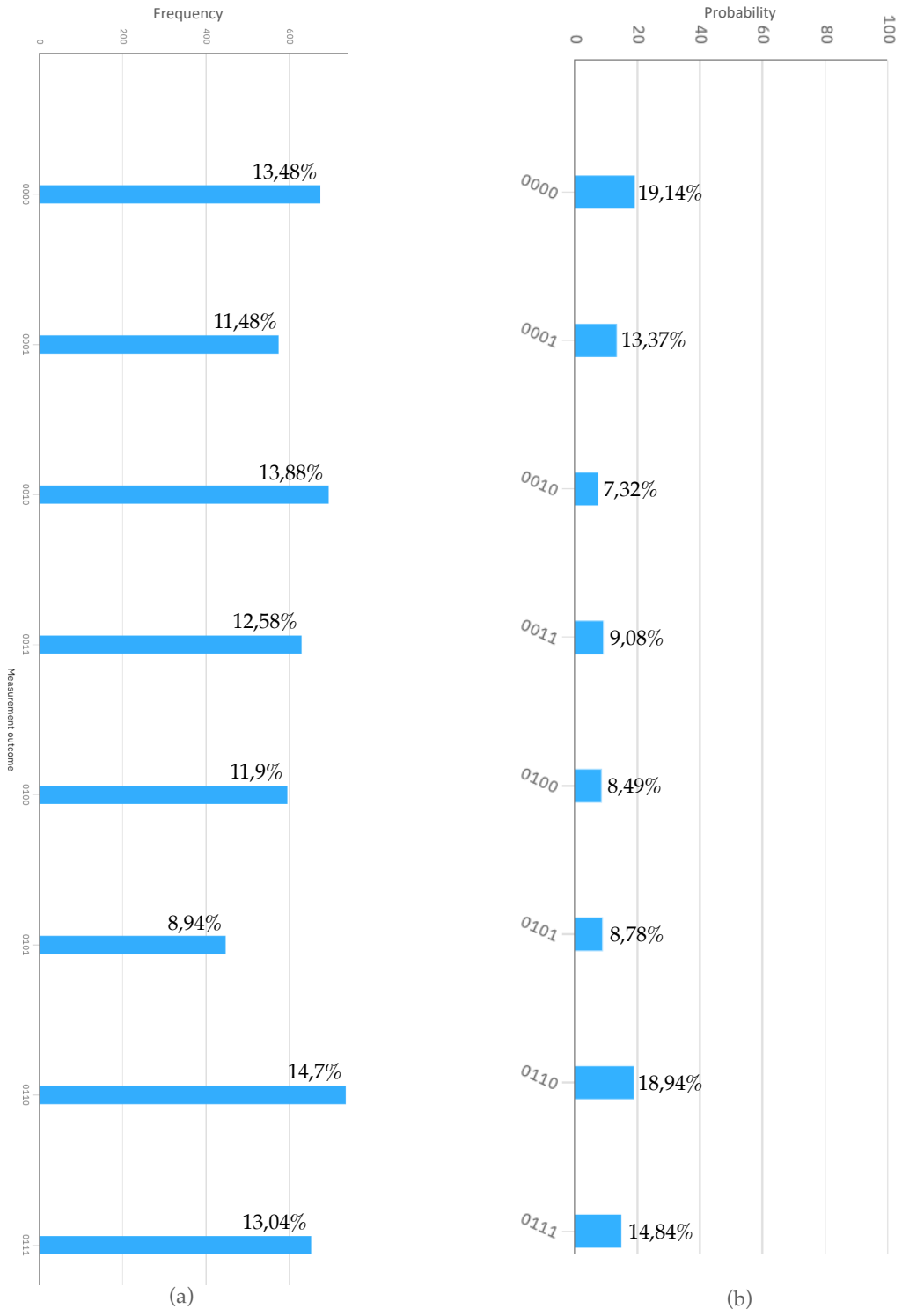


Figura 4.32: Resultados de la ejecución del algoritmo de Shor, versión propuesta por Skosana & Tame [39]. (a) Ejecución en el procesador ibmq_bogota. (b) Propabilidades resultantes en un entorno ideal.

En la Fig. 4.32 se muestra una comparativa entre el resultado debemos esperar del circuito -según el calculador ideal de IBMQ-, y los resultados obtenidos en el procesador ibmq_bogota tras realizar 5000 ejecuciones. El tiempo de ejecución ha sido de 6,7s.

Las dos probabilidades más elevadas que debemos esperar son las de los estados $|0000\rangle$ y $|0110\rangle$, que corresponden a los valores cero y seis en decimal. Anteriormente ya hemos comprobado que seis era un periodo correcto para resolver la factorización de N=21, así que damos por válida esta implementación.

El caso de estudio que hemos considerado N=21 no representa para nada la escala real del problema que este algoritmo pretende resolver. Pero nos ha servido para analizar diferentes formas de resolver esta implementación, cada una mejorando la anterior.

La versión de Vandersypen es la más primitiva -que no quita que sea correcta-, y el coste espacial que presenta necesitando $2\log_2 N$ cúbits para formar los registros es su mayor limitación para ser escalada. Si quisiéramos encontrar los factores de un número formado con 20 cifras decimales, serían necesarios aproximadamente 2*65=130 cúbits.

Por otro lado, la propuesta de Kitaev mejora significativamente el coste espacial a tan solo $(\log_2 N) + 1$ cúbits. Pero a cambio requiere hacer iteraciones condicionadas por los resultados anteriores, lo que reduce la potencia de cálculo cuántica. Esto nos obliga a crear un árbol de posibles circuitos con las puertas multiplicadoras requeridas según el estado de la iteración anterior, porque es posible que hayan diferencias en su construcción en función del estado que recibe. Para un número de 20 cifras decimales, se requieren aproximadamente 65 iteraciones, y en el peor de los casos, deberíamos construir 65*65 = 4225 puertas multiplicadoras.

Por último, la idea que proponen Skosana & Tame para optimizar este algoritmo es la más interesante. Parte de una base matemática que modifica f(x) con el objetivo de reducir el registro de cálculo y, a su vez, simplificar las puertas multiplicadoras. Con esto se consigue un coste espacial de $\log_a(\log_2 N) + \log_2 N$ cúbits, una mejora muy significativa. Escalando esta versión a números de 20 cifras, por ejemplo, considerando $a = 2$, se necesitarían aproximadamente $2\log_2 65 = 13$ cúbits. No obstante, el hecho de que haga uso de puertas Toffoli -CCNOT- para construir las puertas multiplicadoras implica que, en el circuito compilado, se haga uso de un gran número de puertas básicas -porque los procesadores actuales no tienen en su juego de puertas una puerta compleja como la CCNOT-. Al elevar el número de puertas requeridas, la coherencia cuántica será más frágil y el ruido distorsionará los resultados. A no ser que se apliquen técnicas de corrección de errores.

CAPÍTULO 5
Conclusiones

En relación al objetivo principal de este trabajo, gracias a los casos de estudio de algoritmos expuestos, hemos visto el proceso de creación de circuitos haciendo uso de las puertas cuánticas, que son la base para el paradigma de programación de esta nueva tecnología. Con esto, además, hemos visto en qué contextos tienen utilidad estos algoritmos, presentando la mejora computacional que suponen y cómo se deben interpretar los resultados.

Respecto a los objetivos de esta memoria, podemos extraer las siguientes conclusiones:

- Se han expuesto las bases matemáticas y los fundamentos físicos que respaldan esta tecnología y la diferencian de la computación clásica. Como son el paralelismo cuántico y el entrelazamiento de partículas. Además se han presentado los servicios y plataformas cuánticas que ofrecen varias empresas, y mediante el uso de ejemplos hemos visto cómo se transforma el modelo teórico en código ejecutable para cada *framework.*

- A través de la explicación e implementación de algoritmos más sencillos hemos aprendido cómo funcionan las diversas puertas cuánticas mencionadas y cuál es el esquema base para construir un programa que aplique un algoritmo cuántico. Primero la generación de un estado de superposición para el registro de entrada en cuestión. A continuación, la realización de los cálculos necesarios. Y por último, reversión de la superposición para obtener unos resultados medibles en un registro clásico.

- Explicando la complejidad computacional que presenta el algoritmo de Shor respecto a la encriptación RSA, hemos visto que este algoritmo es una amenaza potencial para la seguridad que brinda dicha criptografía. Y con esto hemos entendido que el uso del cálculo cuántico en este caso tiene una implicación que fuerza a la tecnología actual a adaptarse a estas nuevas máquinas para que se siga conservando el *statu quo* del mundo informático. Lo que nos lleva a comprender que la supremacía cuántica es un concepto que se está volviendo cada vez más tangible.

- Finalmente, con la ejecución de los tres algoritmos que se presentan hemos observado que su construcción es factible y mayoritariamente los resultados obtenidos son correctos. Con lo que *a priori* podemos deducir que implementándolos a una escala mayor los resultados superarán satisfactoriamente la barrera de la supremacía cuántica. Pero actualmente la tecnología a nivel de *hardware* no está preparada para soportar un tamaño de cálculo tan grande como puede ser encontrar los factores primos de un número de veinte cifras por el ruido cuántico que está presente en las arquitecturas actuales. La comunidad científica e informática no está capacitada para elaborar

un desarrollo de un circuito de estas dimensiones, porque todavía falta perfeccionar las técnicas de mitigación de errores que habría que aplicar.

Como reflexión personal, me gustaría comentar que la supremacía cuántica es una barrera que ya se ha superado recientemente, aunque solo se haya dado el primer paso en este nuevo espacio computacional. No obstante, entender y dominar el paradigma de programación no es una tarea fácil porque tiene sus dificultades. Y el acceso a ordenadores que te permitan aprovechar este cómputo no está al alcance de todos.

A lo largo del desarrollo del trabajo, me he encontrado con el problema de que, para cada caso concreto de datos en un algoritmo, hay que construir un circuito diferente. Lo que me ha resultado tedioso a la hora de querer generalizar el código de un algoritmo de la misma forma que sucede en la programación clásica. Por tanto veo complicado que se consiga un uso comercial y práctico de esta tecnología por el momento.

Por otro lado, otro inconveniente que me he encontrado es que los procesadores actuales tienen una arquitectura limitada en cuanto a la relación de los cúbits, que obliga a realizar transformaciones en la estructura del código para adaptarlo. Y esto añade una capa más de complejidad que se debe gestionar con técnicas que a su vez enrevesan más el funcionamiento del algoritmo.

Con todo esto, puedo responder a la pregunta que plantea este trabajo y considero que la supremacía cuántica va a suponer el fin de la seguridad clásica en un futuro no mucho más lejano de cinco o diez años, pero en la realidad inmediata todavía hay muchos factores a optimizar. Considero que impacta fuertemente en la ciberseguridad y por eso todas las empresas tienen el foco puesto en investigaciones de este campo. Y que llegado el momento en que una entidad tenga el control total para generar circuitos específicos para cada clave RSA, entonces habrá un problema real.

Personalmente, este trabajo combina el reflejo de los conocimientos que he adquirido en la universidad con mi ambición por aprender sobre nuevas tecnologías que surgen y la influencia que van a tener en el mundo tecnológico en un afán de ser una persona preparada para abordar problemas de este tipo de cómputo. En concreto, las asignaturas que más he podido aplicar en este trabajo son: Seguridad Web (SEW), Hacking Ético (HET) y Redes (RED). Con las que he adquirido el conocimiento de técnicas de encriptación y el uso que se le dan en las comunicaciones. Además de las asignaturas troncales Álgebra(ALG), Estadística (EST), Física(FIS) y Tecnologías de Computadores(TCO), que me han permitido comprender las bases de la computación cuántica.

Por último, en cuanto a futuras líneas de trabajo, se presentan dos posibles caminos. El primero, la construcción de un circuito a gran escala del algoritmo de Shor para conseguir una factorización de un número de. tamaño El segundo, aplicar técnicas de optimización de circuitos para reducir el número de cúbits necesarios.

Referencias

[1] Daniel Patrascu. *Google to Test New Britlescone Quantum Computer Chip with Daimler*. (2018) URL: https://www.autoevolution.com/news/google-to-test-new-bristlecone-quantum-computer-chip-with-daimler-124426.html Visitado el 13-05-21

[2] C.A Calderón, S.A Flórez, J.F Basto, J.N Amaris, J.A Adarme. *Supremacía Cuantica*. URL: **http://wiki.sc3.uis.edu.co/images/1/1f/Uno.pdf** Visitado el 13-05-21

[3] R. Rosenfeld y J. Irazábal. *Computabilidad, complejidad computacional y verificación de programas*. EDULP, 2013. ISBN: 950-34-0970-5.

[4] Pedro Gómez-Esteban. *Computación cuántica III. Las puertas lógicas de un qubit*. (2014) URL: **https://eltamiz.com/elcedazo/2014/04/05/computacion-cuantica-iii-las-puertas-logicas-de-un-qubit/** Visitado el 10-05-21

[5] Pedro Gómez-Esteban. *Cuántica sin fórmulas - El entrelazamiento cuántico*. (2009) URL **https://eltamiz.com/2009/06/24/cuantica-sin-formulas-el-entrelazamiento-cuantico/** Visitado el 15-05-21

[6] Página oficial de documentación de Qiskit. URL: **https://qiskit.org/documentation/qc_intro.html#quantum-circuits** Visitado el 23-04-21

[7] Eduardo Sáenz de Cabezón. *Introducción a la computación cuántica (sin la física)*. (2019) URL: **https://www.youtube.com/watch?v=KKwjeJzKezw** Visitado el 23-04-21

[8] Williams, Colin P. *Explorations in Quantum Computing*. Springer 2n ed. (2011) ISBN:978-1-84628-886-9 URL:**https://link.springer.com/chapter/10.1007%2F978-1-84628-887-6_2** (p. 55-68, 84-110)

[9] Jorge Contreras. *Complejidad Computacional*. URL: **https://jorgecontrerasp.wordpress.com/unidad-ii/complejidad-computacional/** Visitado el 15-04-21

[10] Bernhard Ömer. *Structured Quantum Programming*. Institute of Theoretical Physics Vienna (2009). URL: **http://www.itp.tuwien.ac.at/~oemer/doc/structquprog.pdf** (p. 4, 9-13)

[11] D. Deutsch. *Quantum theory, the Church-Turing principle and the universal quantum computer* (1985), URL: **http://user.it.uu.se/~hessmo/QI/notes/deutsch85.pdf**

[12] Vicente Moret Bonillo. *Principios fundamentales de la computación cuántica*. Universidad de A Coruña (2013). URL: **https://enginyeriainformatica.cat/wp-content/uploads/2016/05/PRINCIPIOS-FUNDAMENTALES-DE-COMPUTACI%C3%93N-CU%C3%81NTICA.pdf**

[13] Ket.G. *El algoritmo de Grover.* (6 may. 2020). URL: **https://www.youtbe.com/watch?v=xPkvB4ECVM4&t=208s** Visitado el 28-04-21

[14] David McMahon. *Quantum Computing Explained.* IEEE Computer Society Press (2007). ISBN : 9780470181362 (p. 173-188, 211, 241-242)

[15] P. Kaye, R. Laflamme, M. Mosca. *An introduction to Quantum Computing*. Oxford University Press (2007). (p. 21-35, 95-101)

[16] R. A. Grimes. *Cryptography apocalypse: preparing for the day when quantum computing breaks today's crypto*. Hoboken 1st edition (2020). ISBN: 1-119-61823-1 (p. 44, 59-74)

[17] A. S. Tanenbaum, D. J. Wetherall, A. Vidal Romero Elizondo. *Redes de computadoras.* Pearson Education (2011). ISBN: 6073208170 (p. 3-7)

[18] *A new kind of quantum* URL: **https://spie.org/news/photonics-focus/novdec-2020/a-new-kind-of-quantum?SSO=1** Visitado el 04-04-21

[19] *Ampliación rápida de las computadoras cuánticas comerciales con trampas de iones.* URL: **https://hwcol.com/2020/11/10/ampliacion-rapida-de-las-computadoras-cuanticas-comerciales-con-trampa-de-iones/** Visitado el 28-04-21

[20] *Una arquitectura nueva para ordenadores cuánticos.* URL: **https://cordis.europa.eu/article/id/33164-new-architecture-for-quantum-computers/es** Visitado el 28-04-21

[21] M. Oskin, F.T Chong, I.L Chuang. *A practical architecture for reliable quantum computers.* IEEE (2002) ISSN: 0018-9162. URL: **https://homes.cs.washington.edu/~oskin/Oskin-A-Practical-Architecture-for-Reliable-Quantum-Computers.pdf** (p. 81, 83-85)

[22] Bernhard Ömer. *Quantum Programming in QCL*. Institute of Theoretical Physics Vienna (2000). URL: **http://tph.tuwien.ac.at/~oemer/doc/quprog.pdf** (p. 4-6, 10, 16, 44-49, 85)

[23] Frank Rioux. *Elements of Dirac Notation*. URL: **https://faculty.csbsju.edu/frioux/dirac/dirac.pdf**

[24] Página oficial de Xanadu Quantum Cloud. URL: **https://www.xanadu.ai/cloud**

[25] Página oficial de IBM Quantum Experience. URL: **https://quantum-computing.ibm.com/services?systems=all**

[26] Página oficial de Google Quantum AI. URL: **https://quantumai.google/cirq?hl=es**

[27] Página oficial de Microsoft Azure Quantum. URL: **https://azure.microsoft.com/en-us/services/quantum/#features**

[28] *Computación cuántica: ¿Qué empresas lideran el sector?* URL: https://revistabyte.es/actualidad-it/computacion-cuantica-que-empresas-lideran-el-sector/ Visitado el 05-06-21

[29] Ecured. *Lenguaje de alto nivel.* (2021) URL: https://www.ecured.cu/Lenguaje_de_alto_nivel Visitado el 05-06-21

[30] Repositorio GitHub oficial del simulador qsim. URL: https://github.com/quantumlib/qsim

[31] L. Fortnow. *The status of the P versus NP problem*. (2009). ACM URL: https://dl.acm.org/doi/pdf/10.1145/1562164.1562186 (p. 78, 82)

[32] J.E González Cornejo. Conceptos matemáticos básicos de computación cuántica. URL: https://docirs.cl/Puertas_Circuitos_Cuanticos.asp Visitado el 02-06-21

[33] Antonio N. Oliveira, Estêvão V.B. de Oliveira, Alan C. Santos, Celso J. Villas-Bôas. *Algoritmos quânticos com IBMQ Experience: Algoritmo de Deutsch-Jozsa.* URL: https://arxiv.org/abs/2109.07910v1 (p. 6, 9, 11)

[34] Aamir Mandviwalla*, Keita Ohshiro*, Bo Ji. *Implementing Grover's Algorithm on the IBM Quantum Computers.* URL: https://cis.temple.edu/~boji/papers/REU2018.pdf (p. 1-4)

[35] J. Abhijith , Adetokumbo Adedoyin, John Ambrosiano, etc. *Quantum Algorithm Implementations for Beginners*. URL: https://arxiv.org/pdf/1804.03719.pdf (p. 7, 11, 17-19, 29-32)

[36] A. Yu. Kitaev. *Quantum measurements and the Abelian stabilizer problem.* (1995) URL: https://arxiv.org/pdf/quant-ph/9511026v1.pdf (p. 1, 2, 4, 5)

[37] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, I. L. Chuang. *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance.* (2001) URL: https://arxiv.org/pdf/quant-ph/0112176.pdf (p. 1, 2, 14, 15)

[38] M. Amico, Z. H. Saleem, M. Kumph. *An experimental study of Shor's Factoring Algorithm on IBM Q.* (2019) URL: https://arxiv.org/pdf/1903.00768.pdf (p. 4, 8, 9)

[39] U. Skosana, M. Tame. *Demonstration of Shor's factoring algorithm for N = 21 on IBM quantum processors.* (2021) URL: https://www.nature.com/articles/s41598-021-95973-w.pdf (p. 4-5)

[40] Guillermo Morales-Luna. Compatibilidad y computación cuántica: Revisión de modelos alternativos de computación. (2011). URL: https://www.academia.edu/2099910/COMPUTABILIDAD_Y_COMPUTACION_CUANTICA_REVISION_DE_MODELOS_ALTERNATIVOS_DE_COMPUTACION (p. 51-61)

[41] Sergio Boixo. *Conferencia de Sergio Boixo Introducción y estado actual de la computación cuántica. UCM.* (2021) URL: **https://www.youtube.com/watch?v=OaMLmGsNkDg&t=3927s**

Apéndices

A. Acrónimos

- **AES:** Estándar avanzado de encriptación
- **ALU:** Unidad aritmética lógica.
- **API:** Interfaz de programación de aplicaciones.
- **DES:** Encriptación de datos estándar.
- **GNFS:** Criba general del cuerpo de números.
- **IaaS:** Infraestructura como servicio.
- **IBM:** *International Business Machine*.
- **IBMQ:** Plataforma cuántica de IBM.
- **IDE:** Entorno de desarrollo integrado.
- **IQFT:** Transformada cuántica inversa de Fourier.
- **MCD:** Máximo común divisor.
- **NASA:** *National Aeronautics and Space Administration*.
- **PaaS:** Plataforma como servicio.
- **QaaS:** *Software* cuántico como servicio.
- **QASM:** Lenguaje ensamblador cuántico.
- **QCL:** Lenguaje C cuántico.
- **QFT:** Transformada cuántica de Fourier.
- **QTM:** Máquina de Turing cuántica.
- **RAM:** Memoria de acceso aleatorio.
- **RSA:** Algoritmo de cifrado Rivest, Shamir, Adleman.
- **SaaS:** *Software* como servicio.
- **SDK:** Kit de desarrollo de *software*.

B. Código del algoritmo de Deustch en Qiskit

```
from ibm_quantum_widgets import CircuitComposer
from qiskit import *
from numpy import pi

# Acceso a cuenta de IBMQ
provider = IBMQ.load_account()
# Inicialización registro cuántico
qreg_q = QuantumRegister(2, 'q')
# Inicialización registro clásico
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

# Inicialización de estados
circuit.id(qreg_q[0])
circuit.x(qreg_q[1])
# Barreras
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[1])
# Hadamard
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
# Query function
circuit.cx(qreg_q[0], qreg_q[1])
# Hadamard
circuit.h(qreg_q[0])
# Obtención de resultados
circuit.measure(qreg_q[0], creg_c[0])

# Ejecución en un procesador real
cpuq = provider.get_backend('ibmq_bogota')
transpiled = transpile(circuit, backend=backend)
job = execute(circuit, backend = cpuq)

job_monitor(job)
result = job.result()

# Imprime el circuito
circuit.draw()

# Imprime las probabilidades en un histograma
plot_histogram(result.get_counts(circuit))
```

C. Código del algoritmo de Deustch-Jozsa en Qiskit para *n*=3

```
from qiskit import *
from numpy import pi
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *

# Acceso a cuenta de IBMQ
provider = IBMQ.load_account()

# Inicialización registro cuántico
qreg_q = QuantumRegister(4, 'q')
# Inicialización registro clásico
creg_c = ClassicalRegister(4, 'c')

circuit = QuantumCircuit(qreg_q, creg_c)

circuit.id(qreg_q[0])
circuit.id(qreg_q[1])
circuit.id(qreg_q[2])
circuit.x(qreg_q[3])
# Barreras
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[1])
circuit.barrier(qreg_q[2])
circuit.barrier(qreg_q[3])
# Hadamard para superponer estados
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
# Query function
circuit.id(qreg_q[0])
circuit.id(qreg_q[1])
circuit.cx(qreg_q[2], qreg_q[3])
# Hadamard para deshacer la superposición
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.id(qreg_q[3])
# Barreras
circuit.barrier(qreg_q[3])
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[1])
circuit.barrier(qreg_q[2])
# Obtención de resultados
circuit.measure(qreg_q[0])
circuit.measure(qreg_q[1])
circuit.measure(qreg_q[2])

# Ejecución en un procesador real
cpuq = provider.get_backend('ibmq_lima')
transpiled = transpile(circuit, backend=backend)
job = execute(circuit, backend = cpuq)

job_monitor(job)
```

```
result = job.result()

# Imprime el circuito
circuit.draw()

# Imprime las probabilidades en un histograma
plot_histogram(result.get_counts(circuit))
```

D. Código del algoritmo de Grover en Qiskit para 5 cúbits

```
from qiskit import *
from numpy import pi
from qiskit.visualization import *
from ibm_quantum_widgets import *

# Acceso a cuenta de IBMQ
provider = IBMQ.load_account()

# Inicialización registro cuántico
qreg_q = QuantumRegister(4, 'q')
# Inicialización registro clásico
creg_c = ClassicalRegister(4, 'c')

circuit = QuantumCircuit(qreg_q, creg_c)

# Transformación Hadamard
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])

# Oráculo operador Grover
circuit.x(qreg_q[0])
circuit.id(qreg_q[1])
circuit.x(qreg_q[2])
circuit.x(qreg_q[3])

#####################
#    Puerta CCCZ.   #
#####################
circuit.cp(pi/4, qreg_q[0], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.cp(-pi/4, qreg_q[1], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.cp(pi/4, qreg_q[1], qreg_q[3])
circuit.cx(qreg_q[1], qreg_q[2])
circuit.cp(-pi/4, qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[2])
circuit.cp(pi/4, qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[1], qreg_q[2])
circuit.cp(-pi/4, qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[2])
circuit.id(qreg_q[0])
circuit.id(qreg_q[1])
circuit.cp(pi/4, qreg_q[2], qreg_q[3])
# Fin puerta CCCZ

circuit.x(qreg_q[0])
circuit.x(qreg_q[1])
circuit.x(qreg_q[2])
circuit.x(qreg_q[3])
# Fin oráculo operador Grover

# Amplificación
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
```

```
circuit.h(qreg_q[3])
circuit.x(qreg_q[0])
circuit.x(qreg_q[1])
circuit.x(qreg_q[2])
circuit.x(qreg_q[3])

####################
#    Puerta CCCZ.   #
####################
circuit.cp(pi/4, qreg_q[0], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.cp(-pi/4, qreg_q[1], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.cp(pi/4, qreg_q[1], qreg_q[3])
circuit.cx(qreg_q[1], qreg_q[2])
circuit.cp(-pi/4, qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[2])
circuit.cp(pi/4, qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[1], qreg_q[2])
circuit.cp(-pi/4, qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[2])
circuit.barrier(qreg_q[0])
circuit.id(qreg_q[1])
circuit.cp(pi/4, qreg_q[2], qreg_q[3])
# Fin puerta CCCZ

circuit.x(qreg_q[0])
circuit.x(qreg_q[1])
circuit.x(qreg_q[2])
circuit.x(qreg_q[3])
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
# Fin amplificación

# Barreras
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[1])
circuit.barrier(qreg_q[2])
circuit.barrier(qreg_q[3])

# Obtención de resultados
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])
circuit.measure(qreg_q[2], creg_c[2])
circuit.measure(qreg_q[3], creg_c[3])

# Ejecución en un procesador real
cpuq = provider.get_backend('ibmq_lima')
transpiled = transpile(circuit, backend=backend)
job = execute(circuit, backend = cpuq)

job_monitor(job)
result = job.result()

# Imprime el circuito
circuit.draw()

# Imprime las probabilidades en un histograma
```

```
plot_histogram(result.get_counts(circuit))
```

E. Código de Shor para N=21, a=2, versión Vandersypen

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

# Acceso a cuenta de IBMQ
provider = IBMQ.load_account()

# Inicialización del registro cuántico (R1 + R2)
qreg_q = QuantumRegister(10, 'q')
# Inicialización del registro clásico
creg_c = ClassicalRegister(10, 'c')

circuit = QuantumCircuit(qreg_q, creg_c)

# Transformación Hadamard sobre R2
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
circuit.h(qreg_q[4])

# Construcción del estado en R1
circuit.x(qreg_q[5])

# Barreras
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3], qreg_q[4],
qreg_q[5], qreg_q[6], qreg_q[7], qreg_q[8], qreg_q[9])

# Puerta multiplicadora modular por 16
circuit.cx(qreg_q[0], qreg_q[5])
circuit.cx(qreg_q[0], qreg_q[9])

# Barreras
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3], qreg_q[4],
qreg_q[5], qreg_q[6], qreg_q[7], qreg_q[8], qreg_q[9])

# Puerta multiplicadora modular por 4
circuit.cswap(qreg_q[1], qreg_q[5], qreg_q[7])
circuit.cswap(qreg_q[1], qreg_q[9], qreg_q[5])

# Barreras
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3], qreg_q[4],
qreg_q[5], qreg_q[6], qreg_q[7], qreg_q[8], qreg_q[9])

# Puerta multiplicadora modular por 16
circuit.cx(qreg_q[2], qreg_q[5])
circuit.cx(qreg_q[2], qreg_q[9])

# Barreras
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3], qreg_q[4],
qreg_q[5], qreg_q[6], qreg_q[7], qreg_q[8], qreg_q[9])

# Puerta multiplicadora modular por 4
circuit.cswap(qreg_q[3], qreg_q[5], qreg_q[7])
circuit.cswap(qreg_q[3], qreg_q[9], qreg_q[5])

# Barreras
```

```
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3], qreg_q[4],
qreg_q[5], qreg_q[6], qreg_q[7], qreg_q[8], qreg_q[9])

# # Puerta multiplicadora modular por 2
circuit.cswap(qreg_q[4], qreg_q[5], qreg_q[6])
circuit.cswap(qreg_q[4], qreg_q[7], qreg_q[8])
circuit.cswap(qreg_q[4], qreg_q[5], qreg_q[9])
circuit.ccx(qreg_q[4], qreg_q[5], qreg_q[6])
circuit.ccx(qreg_q[4], qreg_q[7], qreg_q[8])

# Barreras
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3], qreg_q[4],
qreg_q[5], qreg_q[6], qreg_q[7], qreg_q[8], qreg_q[9])

# Transformada de Fourier sobre 5 cúbits
circuit.h(qreg_q[4])
circuit.crz(-pi / 2, qreg_q[3], qreg_q[4])
circuit.crz(-pi / 4, qreg_q[2], qreg_q[4])
circuit.h(qreg_q[3])
circuit.crz(-pi / 8, qreg_q[1], qreg_q[4])
circuit.crz(-pi / 2, qreg_q[2], qreg_q[3])
circuit.crz(-pi / 16, qreg_q[0], qreg_q[4])
circuit.crz(-pi / 4, qreg_q[1], qreg_q[3])
circuit.h(qreg_q[2])
circuit.crz(-pi / 8, qreg_q[0], qreg_q[3])
circuit.crz(-pi / 2, qreg_q[1], qreg_q[2])
circuit.crz(-pi / 4, qreg_q[0], qreg_q[2])
circuit.h(qreg_q[1])
circuit.crz(-pi / 2, qreg_q[0], qreg_q[1])

# Barreras
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3], qreg_q[4],
qreg_q[5], qreg_q[6], qreg_q[7], qreg_q[8], qreg_q[9])

# Obtención de resultados
circuit.measure(qreg_q[0], creg_c[4])
circuit.measure(qreg_q[1], creg_c[3])
circuit.measure(qreg_q[2], creg_c[2])
circuit.measure(qreg_q[3], creg_c[1])
circuit.measure(qreg_q[4], creg_c[0])

# Ejecución en un procesador real
cpuq = provider.get_backend('ibmq_lima')
transpiled = transpile(circuit, backend=backend)
job = execute(circuit, backend = cpuq)

job_monitor(job)
result = job.result()

# Imprime el circuito
circuit.draw()

# Imprime las probabilidades en un histograma
plot_histogram(result.get_counts(circuit))
```

F. Código de Shor para N=21, a=2, versión Skosana & Tame

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

# Acceso a cuenta de IBMQ
provider = IBMQ.load_account()

# Inicialización del registro cuántico (R1 + R2)
qreg_q = QuantumRegister(5, 'q')
# Inicialización del registro clásico
creg_c = ClassicalRegister(4, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

# Transformación de Hadamard sobre R2
circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.id(qreg_q[4])
circuit.id(qreg_q[3])

# Barreras
circuit.barrier(qreg_q[3])
circuit.barrier(qreg_q[4])
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[1])
circuit.barrier(qreg_q[2])

# Puerta multiplicadora U^1
circuit.cx(qreg_q[2], qreg_q[4])
circuit.id(qreg_q[1])
circuit.id(qreg_q[0])

# Barreras
circuit.barrier(qreg_q[3])
circuit.barrier(qreg_q[4])
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[1])
circuit.barrier(qreg_q[2])

# Puerta multiplicadora U^2
circuit.cx(qreg_q[1], qreg_q[4])
circuit.id(qreg_q[0])
circuit.id(qreg_q[2])
circuit.cx(qreg_q[4], qreg_q[3])
circuit.id(qreg_q[0])
circuit.id(qreg_q[0])
circuit.id(qreg_q[2])
circuit.id(qreg_q[0])
circuit.ccx(qreg_q[1], qreg_q[3], qreg_q[4])
circuit.id(qreg_q[0])
circuit.cx(qreg_q[4], qreg_q[3])
circuit.id(qreg_q[1])
circuit.id(qreg_q[2])

# Barreras
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[4])
```

```
circuit.barrier(qreg_q[3])
circuit.barrier(qreg_q[1])
circuit.barrier(qreg_q[2])

# Puerta multiplicadora U^4
circuit.id(qreg_q[1])
circuit.id(qreg_q[2])
circuit.id(qreg_q[3])
circuit.x(qreg_q[4])
circuit.id(qreg_q[2])
circuit.id(qreg_q[1])
circuit.ccx(qreg_q[0], qreg_q[4], qreg_q[3])
circuit.id(qreg_q[1])
circuit.x(qreg_q[4])
circuit.id(qreg_q[3])
circuit.id(qreg_q[2])
circuit.cx(qreg_q[4], qreg_q[3])
circuit.id(qreg_q[1])
circuit.id(qreg_q[2])
circuit.id(qreg_q[1])
circuit.id(qreg_q[2])
circuit.ccx(qreg_q[0], qreg_q[4], qreg_q[3])
circuit.id(qreg_q[0])
circuit.id(qreg_q[1])
circuit.cx(qreg_q[4], qreg_q[3])
circuit.id(qreg_q[2])

# Barreras
circuit.barrier(qreg_q[4])
circuit.barrier(qreg_q[0])
circuit.barrier(qreg_q[1])
circuit.barrier(qreg_q[3])
circuit.barrier(qreg_q[2])

# Puerta QFT sobre tres cúbits
circuit.h(qreg_q[2])
circuit.crz(pi / 2, qreg_q[1], qreg_q[2])
circuit.crz(pi / 4, qreg_q[0], qreg_q[2])
circuit.h(qreg_q[1])
circuit.id(qreg_q[2])
circuit.id(qreg_q[2])
circuit.crz(pi / 2, qreg_q[0], qreg_q[1])
circuit.id(qreg_q[2])
circuit.h(qreg_q[0])
circuit.id(qreg_q[1])

# Obtención de resultados
circuit.measure(qreg_q[1], creg_c[1])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[2], creg_c[2])

# Ejecución en un procesador real
cpuq = provider.get_backend('ibmq_bogota')
transpiled = transpile(circuit, backend=backend)
job = execute(circuit, backend = cpuq)

job_monitor(job)
result = job.result()
```

```
# Imprime el circuito
circuit.draw()

# Imprime las probabilidades en un histograma
plot_histogram(result.get_counts(circuit))
```

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**			x	
ODS 2. **Hambre cero.**			x	
ODS 3. **Salud y bienestar.**		x		
ODS 4. **Educación de calidad.**				x
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**			x	
ODS 7. **Energía asequible y no contaminante.**		x		
ODS 8. **Trabajo decente y crecimiento económico.**			x	
ODS 9. **Industria, innovación e infraestructuras.**	x			
ODS 10. **Reducción de las desigualdades**.				x
ODS 11. **Ciudades y comunidades sostenibles**.	x			
ODS 12. **Producción y consumo responsables.**	x			
ODS 13. **Acción por el clima.**	x			
ODS 14. **Vida submarina.**		x		
ODS 15. **Vida de ecosistemas terrestres.**		x		
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En el año 2015, los países que conforman la Organización de Naciones Unidas redactaron una serie de objetivos globales para frenar los problemas que ya están afectando al bienestar del planeta y de la sociedad a nivel mundial. Y que en futuro pueden tener consecuencias devastadoras si no nos concienciamos como especie humana inteligente que somos y empezamos a actuar en conjunto.

Estos objetivos, que podrían considerarse OKRs -hablando en términos de ingeniería del software-, deben alcanzarse con acciones -que podemos considerar como KPIs de ingeniería de software- de toda la población, integrándolas en todos los ámbitos de nuestra vida. La previsión que se hizo fue que en un máximo de 15 años se tienen que haber logrado todos ellos.

Las acciones se pueden tomar relacionadas con este trabajo y la computación cuántica son:

- **Salud y bienestar.** El uso de procesadores cuánticos para simular el comportamiento de las moléculas supondrá un gran avance en la investigación de enfermedades y métodos de curación que hasta ahora no es posible simular ni en los mejores super-computadores debido a que no pueden replicar el comportamiento exacto de las partículas.

- **Energía asequible y no contaminante.** El paradigma cuántico abre nuevas puertas al desarrollo de algoritmos que optimicen el uso de recursos energéticos con un procesamiento de datos más rápido e inteligente y, por tanto, beneficiará también a hacer un consumo más responsable.

- **Industria innovación e infraestructura.** En sí la computación cuántica ya supone una revolución innovadora para toda la industria tecnológica, y el descubrimiento de algoritmos como el de Grover o el de Shor está motivando a la comunidad científica a desarrollar nuevas mejoras tecnológicas que se adapten a la existencia de ellos. Por ejemplo, el desarrollo de un sistema de base de datos que integre la búsqueda cuántica en las consultas. O también la creación de nuevas técnicas criptográficas post-cuánticas que se puedan aplicar a los sistemas de datos actuales.

- **Ciudades y comunidades responsables.** También cabe mencionar este objetivo porque con el procesamiento cuántico se van a poder construir núcleos de población más

inteligentes, que sepan recopilar y analizar un gran volumen de datos para después enfocar la eficiencia de la tecnología que nos rodea por las calles hacia donde sea más necesaria.

ETS Enginyeria Informàtica
Camí de Vera, s/n. 46022. València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es