Universidad de Castilla-La Mancha

Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de

Computación

Autor

David Miguel Miranda Rodríguez

Julio de 2024

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de

Computación

Estudio del algoritmo FF-QRAM y aplicaciones con algoritmos de Quantum Machine Learning

Autor: David Miguel Miranda Rodríguez

Tutor: Fernando López Pelayo

Cotutor: José Javier Paulet González

Julio 2024

Declaración de Autoría

Yo, David Miguel Miranda Rodríguez con DNI 48151110N, declaro que soy el único autor del trabajo fin de grado titulado "Estudio del algoritmo FF-QRAM y aplicaciones con algoritmos de Quantum Machine Learning"" y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a.....

Fdo: .....

Resumen

El objetivo principal de este Trabajo Fin de Grado es implementar el algoritmo de codificación cuántica Flip-Flop QRAM y disponerlo para su uso para otros algoritmos de Quantum Machine Learning. Para ello, hace falta entender el algoritmo, que será explicado teóricamente, y a continuación se expondrá una introducción a la propia computación cuántica y al contexto de la tecnología tanto en la historia como en la actualidad. Finalmente, se dará lugar a unas conclusiones y a algunas consideraciones en cuanto al futuro de este algoritmo.

Agradecimientos

En primer lugar, me gustaría empezar agradeciendo a mis tutores Fernando López Pelayo y José Javier Paulet González, por ofrecerme la oportunidad de realizar este Trabajo Fin de Grado relacionado con un tema tan fascinante. También quiero agradecerles por haberme acompañado y ayudado durante la realización de este trabajo.

Quiero también agradecer a todos los profesores que han contribuido en mi formación académica, tanto en la etapa universitaria como en las demás.

También quiero dar las gracias a mi familia, en especial a mis padres y hermano por haber estado siempre para mí, depositándome su confianza y apoyo. Por último, quiero hacer un agradecimiento especial a mi pareja, quién me ha escuchado siempre que he necesitado alguien que me escuchara y me ha apoyado siempre que he necesitado alguien que me apoyara.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1 Motivación

Es un hecho que la computación cuántica está cada vez más presente en el paradigma de la informática y con el aumento exponencial de empresas dispuestas a invertir en esta tecnología, estamos siendo testigos de una carrera tecnológica en este campo.
Debido al auge general que este campo está teniendo tanto en el interés que causa a las empresas debido a sus prometedores incrementos de productividad como en el interés científico y curiosidad que causa a los profesionales del sector, he decidido optar por realizar mi Trabajo Fin de Grado utilizando este campo como base, personalmente, por el interés que me causa y, por la proyección que va a tener a futuro.

1.2 Objetivos

Como se comenta brevemente en el "Resumen" de este trabajo, el objetivo principal que persigue este proyecto es la implementación del algoritmo FF-QRAM. Para llegar a ese logro, se proponen los siguientes objetivos de menor entidad:

1. Estudio de los principios básicos de la computación cuántica.
2. Contextualización del funcionamiento de RAM clásica.
3. Estudio teórico y práctico de QRAM.
4. Estudio del algoritmo cuántico FF-QRAM.

5. Manejo del *framework* de computación cuántica Qiskit.
6. Implementación del algoritmo FF-QRAM para su uso en *Quantum Machine Learning*.

1.3 Estructura del proyecto

La memoria está estructurada de la siguiente forma:

1. Capítulo 1: Introducción. En esta sección se trata la motivación del autor al realizar este trabajo, los objetivos de este y, la propia forma en que se ha estructurado el proyecto.
2. Capítulo 2: Antecedentes y Estado del Arte del campo en que se desarrollará este TFG. En este apartado se verá la evolución de la computación cuántica desde sus inicios, así como su evolución hasta su estado actual. Se presentará con detalle el estado actual del paradigma y se hará una introducción de los conceptos básicos de la computación cuántica. También se verá una breve introducción al funcionamiento de una memoria RAM.
3. Capítulo 3: Quantum RAM y algoritmo FF-QRAM. En esta sección se entrará en materia explicando el concepto de QRAM y su utilidad, lo que dará a pie a estudiar el algoritmo FF-QRAM. Finalmente se presentará el primer contacto con el *framework* de computación cuántica Qiskit.
4. Capítulo 4: En este capítulo se describirá en profundidad el cuaderno de Jupyter donde se ha implementado el algoritmo FF-QRAM.
5. Capítulo 5: El capítulo de Conclusiones y Trabajo Futuro resume el nivel de consecución de los objetivos propuestos y plantea hipótesis sobre las posibilidades de aplicación en el futuro.

Capítulo 2

Estado del Arte y antecedentes del campo de trabajo

2.1 Introducción

Antes de hablar de algoritmos QRAM, QML y demás conceptos que pueden quedar lejanos aun teniendo conocimientos de computación, es necesario conocer otros conceptos y terminologías previamente. De manera análoga se hará una revisión al contexto histórico, para entender la evolución de este paradigma desde que se conceptualizó.

Se puede definir la computación cuántica como la tarea de procesamiento de información que se apoya en un motor de cómputo basado en las leyes de la mecánica cuántica. Este campo además de los conceptos fundamentales de la mecánica cuántica, involucra a la teoría de la información, a la ciencia de la computación en general y a la criptografía.

2.2 Evolución de la computación cuántica.

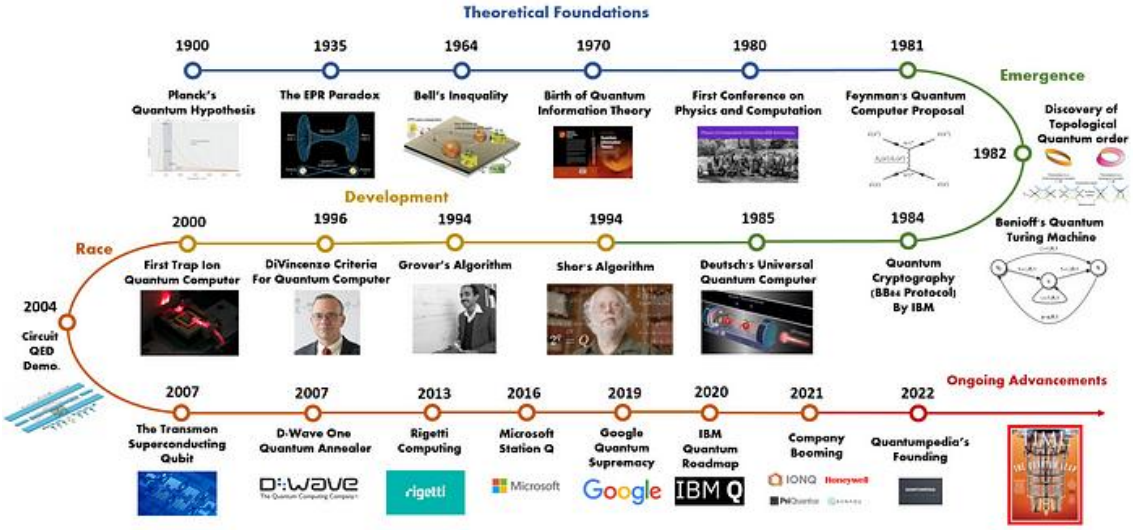

Figura 1 Momentos relevantes en la historia de la computación cuántica

En 1900, Max Planck, se adentra en el estudio del fenómeno de la radiación emitida por un cuerpo negro ideal. Propuso que la energía de la radiación no es emitida de manera continua, sino en unidades discretas llamadas "cuantos". Esta idea fue revolucionaria y marcaría el inicio de una era de desarrollo de la teoría cuántica. Cinco años después, Albert Einstein publicaría un artículo relacionado con el efecto fotoeléctrico, explicando que la luz consiste en partículas llamadas **fotones**, proponiendo que cada una tiene una energía específica y esta es proporcional a su frecuencia, teniendo presente la siguiente ecuación: $\boldsymbol{E = h \cdot f}$, donde E es la energía, h es la constante de Planck y f es la frecuencia. Este artículo se apoyó en las ideas de los cuantos de Max Planck, quien cambió el entendimiento de la radiación de tal manera que, en vez de ser conceptualizada como un chorro de agua, se pensara como un chorro de pelotas de ping-pong, así como también se apoyó en los descubrimientos e ideas de otros científicos. Dicho artículo contribuyó enormemente a la comprensión del comportamiento de la luz en forma de partículas.

En 1913, Niels Bohr desarrolló el **modelo atómico de Bohr**, donde se postula que los electrones orbitan el núcleo en niveles cuantizados de energía. Aparte de proveer un mejor entendimiento de la estructura atómica y los niveles cuantizados de energía, también se introduzco el concepto de **"saltos cuánticos"**, que teorizaba que los electrones se movían entre niveles de energía absorbiendo o emitiendo fotones.

En 1927, Werner Heisenberg, formuló uno de los conceptos más fundamentales de la mecánica cuántica, **el principio de incertidumbre**, que establece que es imposible medir con precisión exacta la posición y momento de una partícula al mismo tiempo. Mediante este principio, se conceptualizan las **limitaciones de las mediciones de sistemas cuánticos.**

En 1935, Albert Einstein, Boris Podolsky and Nathan Rosen publicaron un artículo presentando la **paradoja EPR** donde introducían el **principio de entrelazamiento**, que anuncia que dos o más partículas pueden estar relacionadas tal que el estado de una partícula afecta al momento al estado de la otra, independientemente de la distancia entre ellas. En 1964, John Bell formula una serie de desigualdades matemáticas que se utilizarán para poner a prueba la validez de la paradoja EPR de forma experimental (entre otras teorías físicas). Si los resultados de un experimento violan estas desigualdades, implica que las relaciones entre las partículas no pueden explicarse de manera consistente mediante teorías locales realistas, lo que sugiere, por tanto, la influencia de propiedades cuánticas.

En la década de 1930, John Von Neumann, desarrolló un **marco matemático para la mecánica cuántica**, formalizando estados cuánticos y operadores. Demostró de manera rigurosa el entendimiento del comportamiento de los sistemas cuánticos y sus contribuciones fueron esenciales para encaminar el desarrollo de la computación cuántica como campo.

En 1980 se celebra la primera conferencia de **“PhysComp”**, que congregó a matemáticos, físicos e ingenieros y científicos informáticos para discutir el potencial que podía tener la mecánica cuántica en el campo de la computación. Esta congregación facilitó el intercambio de ideas entre los diferentes campos, desembocando en el nacimiento de la computación cuántica como el campo que conocemos hoy en día. Además, esta conferencia demostró el alto interés que hay en este campo, así como su potencial.

Figura 2 Primera conferencia del PhysComp

El campo de la computación cuántica buscará aprovechar las propiedades únicas (entrelazamiento, superposición, dualidad onda-partícula…) de los sistemas cuánticos para dar comienzo a una nueva era y cambiar la forma en la que entendemos la computación. [1]

2.2.1 El nacimiento de la computación cuántica

En la década de los 80, múltiples investigadores exploraron cómo sacar provecho a estas propiedades cuánticas para las capacidades computaciones, desempeñando un papel clave figuras como **Richard Feynman, Paul Benioff, David Deutsch**:

- Richard Feynman, en la primera conferencia de "PhysComp", propuso que un computador que se rigiera mediante principios cuánticos podría simular entornos cuánticos.
- Paul Benioff, en 1982, publicó un artículo describiendo un modelo cuántico de una máquina de Turing, allanaría el terreno en el desarrollo de modelos de computadores cuánticos, demostrando en su modelo conocido hoy en día como **máquina de Turing cuántica (QMT)**, que era posible enmarcar los principios fundamentales cuánticos junto con la teoría computacional estándar.

- David Deutcsh, en 1985, apoyándose en las contribuciones anteriores de Benioff y Feynman, creó el **concepto de un computador cuántico universal**, donde se podía realizar cualquier operación que un computador clásico realizara, pero con las ventajas de la mecánica cuántica.

Gracias a estas contribuciones y avances, más tarde se empezarían a desarrollar los primeros algoritmos cuánticos.

También, a finales de los años 80 y principios de los 90 emergía el concepto de **"puerta cuántica"**, análogo a las puertas lógicas como bloque de construcción de los circuitos cuánticos en vez de clásicos. Algunas de las primeras puertas que se propusieron son fundamentales hoy en día, como la CNOT. EN el documento que nos ocupa se hará una revisión somera de algunas de las puertas cuánticas más importantes.

2.2.2 Evolución de los algoritmos cuánticos

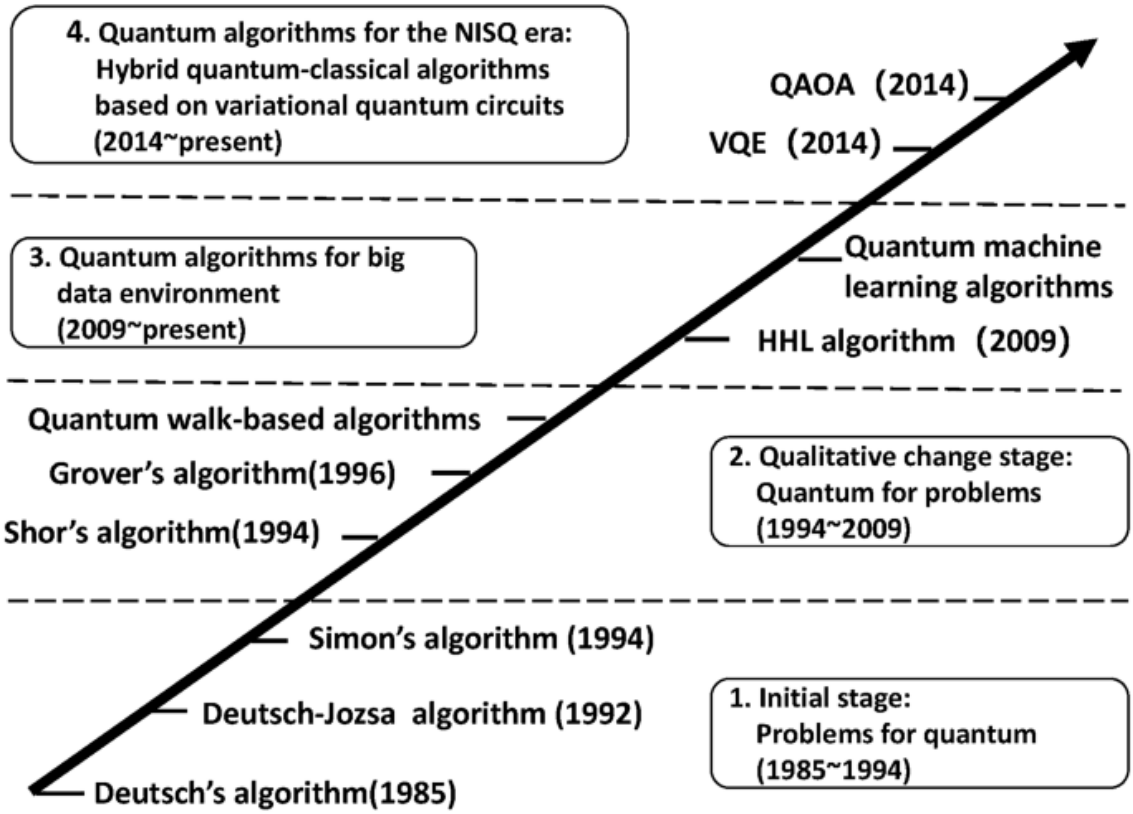

Figura 3 Esquema en el tiempo del desarrollo de algoritmos cuánticos

A mediados de la década de los 90, gracias a los avances previos en el campo y a la disposición de nuevos entornos y marcos de trabajo, se empiezan a hacer palpables algoritmos cuánticos, mostrando el potencial de la computación cuántica.

- En 1994, **Peter Shor**, desarrolló uno de los primeros **algoritmos** cuánticos y sin duda el más **famoso y trascendente**, conocido hoy en día como **algoritmo de Shor**, reconocido por su capacidad para factorizar números grandes en tiempo polinómico, algo que en un computador clásico tomaría mucho más tiempo. La capacidad de este algoritmo tiene importantes implicaciones en el campo de la criptografía, así como en la seguridad de algunos sistemas criptográficos.
- En 1996, **Lov Grover**, desarrolla un algoritmo cuántico para realizar búsquedas en bases de datos desordenadas de forma más eficiente que con los algoritmos clásicos. El **algoritmo de Grover** puede buscar en una secuencia no ordenada de *N* datos en un tiempo $\sqrt{N}$, mientras que los algoritmos clásicos tardarían *N*, ofreciendo una mejora de orden cuadrático en comparación.

Estos algoritmos proporcionaron una muestra más concreta y objetiva de formas en las que se podría aprovechar la computación cuántica, generando interés en las investigaciones y carreras de ambos científicos, así como de otros. El alto interés por la amplitud del dominio en que los algoritmos cuánticos pueden ofrecer una ventaja significativa incitaría a la inversión económica en este campo lo que a su vez provocaría que más investigadores se involucraran con la tecnología. El desarrollo de la computación cuántica se aceleraba significativamente y crecía el interés por el desarrollo de más algoritmos cuánticos.

2.2.3 La carrera hacia la construcción de ordenadores cuánticos

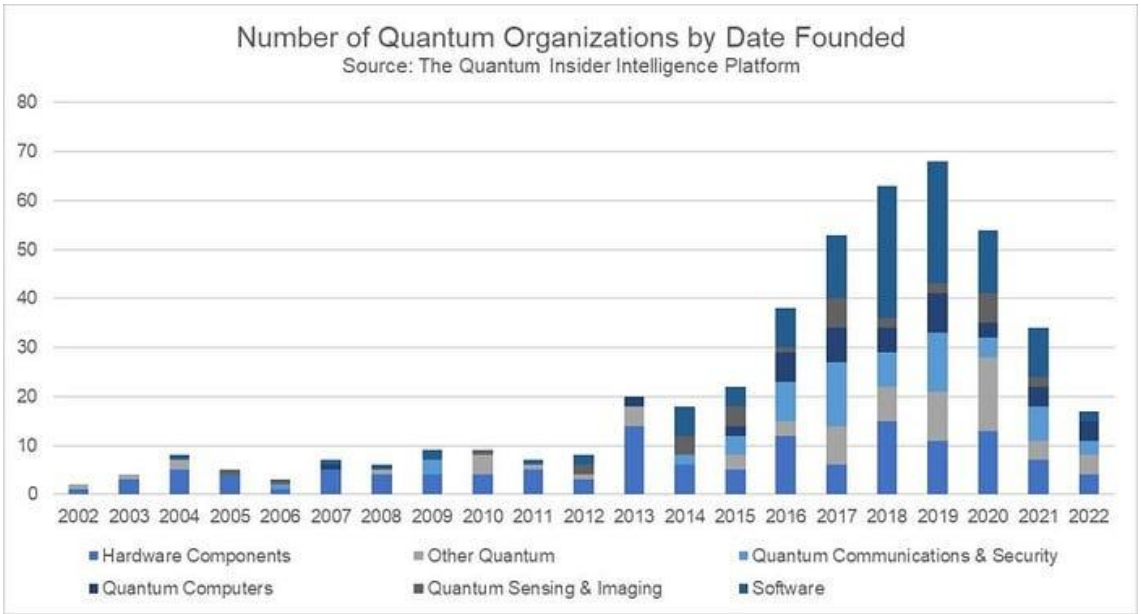

Figura 4 Número de organizaciones cuánticas ordenadas por su fecha de fundación

El siglo XXI se caracteriza, en el ámbito de la computación cuántica, por un gran interés por parte de los gigantes tecnológicos en esta tecnología traduciéndose en grandes inversiones de dinero propiciando la irrupción de una nueva era en computación. Veremos **grandes inversiones** tanto en tiempo como en dinero por parte de investigadores y gigantes tecnológicos en este campo, generándose una carrera hacia la construcción de computadores cuánticos.

En 2011, D-Wave Systems, una empresa privada canadiense de computación cuántica reivindica haber construido el **primer computador cuántico del mundo disponible comercialmente, el D-Wave One, basado en el algoritmo de temple cuántico *(quantum annealing)***, que es una forma de conceptualizar la computación cuántica centrándose en resolver problemas de optimización. Sin embargo, la naturaleza cuántica de este sistema fue sujeto de debate y críticas debido a que la mejora del rendimiento respecto de los computadores clásicos **resultó ser más limitada** de lo esperado teniendo en cuenta que aprovechaba las propiedades de la cuántica.

En 2019, **Google anuncia haber logrado la "supremacía cuántica"** (característica que posee un idílico dispositivo de computación cuántica para resolver problemas que un computador clásico no puede) por primera vez en la historia con su procesador Sycamore de 53 qubits, que era capaz de resolver un determinado problema matemático en 200 segundos mientras que a un supercomputador clásico estándar el

mismo problema le tomaría 10000 años (aunque, finalmente IBM demostró que no había tal supremacía, pues dicho cálculo podía ser procesado en dos días y medio y no en 10000 años, quedando al alcance de los ordenadores actuales). Gracias a esta carrera todavía vigente se logran avances muy significativos, aunque todavía quedan muchos retos por afrontar desde este paradigma.

2.2.4 Situación actual y proyección

Dado el desarrollo emergente de esta tecnología y la inversión que se está realizando en ella por parte de inversores e investigadores, **continuamente tenemos noticias** de avances e hitos en diferentes materias dentro del paradigma, por ello, aunque solo hayan pasado apenas 5 años desde la presentación del procesador Sycamore de 53 qubits de Google, sigue habiendo cosas que se deben comentar.

En 2023, una investigación liderada por el mismo equipo encargado del procesador Sycamore de 53 qubits, informó de que una segunda generación de su procesador cuántico Sycamore, **ahora de 70 qubits, había conseguido realizar una operación en 6,18 segundos**, mientras que, al supercomputador clásico más rápido del mundo, el supercomputador Frontier en el laboratorio nacional de Oak Ridge (Tennesse), le tomaba realizar esa misma operación 47 años. A pesar de esta demostración, algunos expertos discuten que los computadores cuánticos solo han demostrado la superioridad de los computadores cuánticos frente a los clásicos en operaciones matemáticas y que tal vez podrían no tener un valor práctico más allá de investigaciones académicas.

Uno de los mayores retos actuales del paradigma es hacer frente al "ruido", que puede perjudicar la precisión de los cálculos. Con ruido nos referimos a cualquier molestia o perturbación en el entorno del qubit que pueda causar pérdida de información, como luz, radiación electromagnética, calor, el campo magnético de la tierra, etc. La solución conceptualizada a este problema ha sido desarrollar los procesadores cuánticos tolerantes a los fallos o ejecutar algoritmos cuánticos libres de errores. Ambas ideas requieren al menos diez mil qubits, una cantidad que actualmente queda lejos de la capacidad actual de los procesadores cuánticos.

El equipo de investigadores cuánticos de Google publicó un artículo en febrero de 2023 afirmando que sus investigadores eran capaces de demostrar que **incrementar el número de qubits podía**, progresivamente**, reducir el porcentaje de error** en los

cálculos de un computador cuántico [2]. En junio de 2023, un equipo liderado por los investigadores cuánticos de IBM presentó los resultados de un experimento para, aparentemente, “manipular de forma controlada” el ruido cuántico, logrando el éxito en un entorno experimental. Sin embargo, la mejora lograda con este experimento es todavía pequeña, y algunos expertos sostienen que para que la computación cuántica sea una tecnología fiable, es necesario una corrección completa de los errores cuánticos [3].

Hasta ahora, el computador cuántico más avanzado existente es propiedad de IBM; el chip llamado Condor, posee 1121 qubits superconductores organizados en forma de panal. La misma compañía ya poseía el récord anteriormente con su chip de 433 qubits llamado Osprey [4].

Además, también IBM, ha formado un acuerdo junto a la universidad de Tokio y la de Chicago por valor de 100 millones de dólares para autofinanciarse con el objetivo de desarrollar un supercomputador cuántico de 100000 qubit. A su vez, Google también se encuentra compitiendo, tratando de desarrollar un procesador de 1000 qubits para 2025 y con objetivos a largo plazo más ambiciosos, apuntando hacia un computador cuántico de un millón de qubits, corregido de errores, para dentro también de una década [5].

2.3 Introducción a la computación cuántica

2.3.1 Naturaleza del qubit

A continuación, indagaremos en algunos conceptos clave básicos para entender los contenidos que vamos a tratar. El **qubit**, análogo al bit, es la **unidad básica de información de los ordenadores cuánticos**. Al igual que este, puede tomar el valor 0 y 1, llamados en este caso $|0\rangle$ y $|1\rangle$. Esta notación, "$|\rangle$", se llama **"Notación de Dirac"**, la usaremos como estándar para los estados en computación cuántica.

El qubit tiene una serie de características que la vuelven una unidad más compleja y con mayor potencial que el bit, una de ellas es el **principio de superposición**: un qubit puede existir en una superposición de estados, representado simultáneamente $|0\rangle$ y $|1\rangle$, eventualmente con más probabilidad de uno que de otro y, en el momento que es medido, colapsará hacia un valor o hacia otro, teniendo en cuenta su amplitud de

probabilidad hacia un estado u otro. Debido a este principio de superposición, para describir un **estado general**, lo haremos de la siguiente forma: **$|\psi\rangle$= α|0⟩ y β|1⟩**.

$|\alpha|^2$ será la **probabilidad de que el resultado que obtengamos del qubit sea 0** cuando realicemos la medida y $|\beta|^2$ de que obtengamos uno.

Para describir el **estado de un qubit en superposición de forma vectorial** denotaremos de la siguiente forma: $\begin{bmatrix}\boldsymbol{\alpha}\\ \boldsymbol{\beta}\end{bmatrix}$, de tal forma que $|0\rangle$ representa $\begin{bmatrix}1\\0\end{bmatrix}$ y $|1\rangle$ representa $\begin{bmatrix}0\\1\end{bmatrix}$.

Para representar geométricamente a un qubit y su estado, usamos la **esfera de Bloch**:

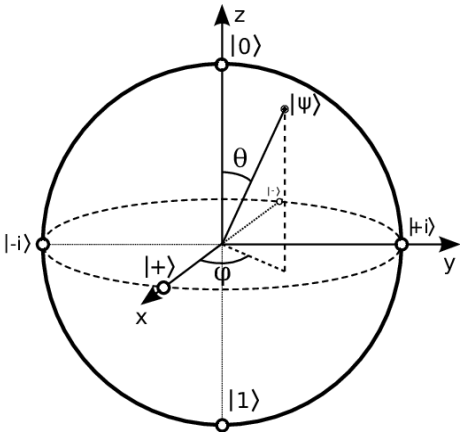

Figura 5 Representación geométrica de la esfera de Bloch

En esta esfera, **cada punto representa un estado cuántico posible**. Tenemos el polo norte que representa el estado $|0\rangle$ y el polo sur el estado $|1\rangle$, la superficie de la esfera representa todos los estados cuánticos posibles y, el interior representaría estados que son combinaciones probabilistas de estados puros, llamados estados cuánticos superpuestos o mixtos. **La clave de esta representación geométrica es que el estado de un qubit es más probable que colapse hacia |0⟩ cuanto más al norte esté y más probable que colapse hacia |1⟩ cuanto más al sur esté**. Por lo que, si el estado se encuentra situado exactamente en el polo norte, no habría posibilidad de que colapsara hacia $|1\rangle$ y viceversa, aquí entrarán en juego las puertas cuánticas para manipular el estado de los qubits antes de realizar mediciones.

Vamos a profundizar un poco más en los principios característicos del qubit:

- Para entender bien el **principio de superposición** podemos tomar como referencia el famoso ejemplo de **"El gato de Schrödinger"**, propuesto en 1935 por Erwin Schrödinger para remarcar las peculiaridades de la teoría cuántica. En el experimento, imaginamos a un gato encerrado en una caja con un átomo radiactivo, un martillo, un frasco de veneno y un contador Geiger. Si el contador

detecta radiación, el mecanismo que acciona el martillo se activa, rompiendo el frasco de veneno y muriendo el gato de esta forma, si no, el gato vive. La teoría dice que el átomo está en un estado de superposición, lo que significa que está simultáneamente en estado de desintegración y no, lo que a su vez quiere decir que el propio sistema que tenemos en la caja también está en un estado de superposición en el que el gato también está simultáneamente vivo y muerto. La superposición colapsará cuando abramos la caja (realicemos la medición) para comprobar si el gato está vivo o muerto.

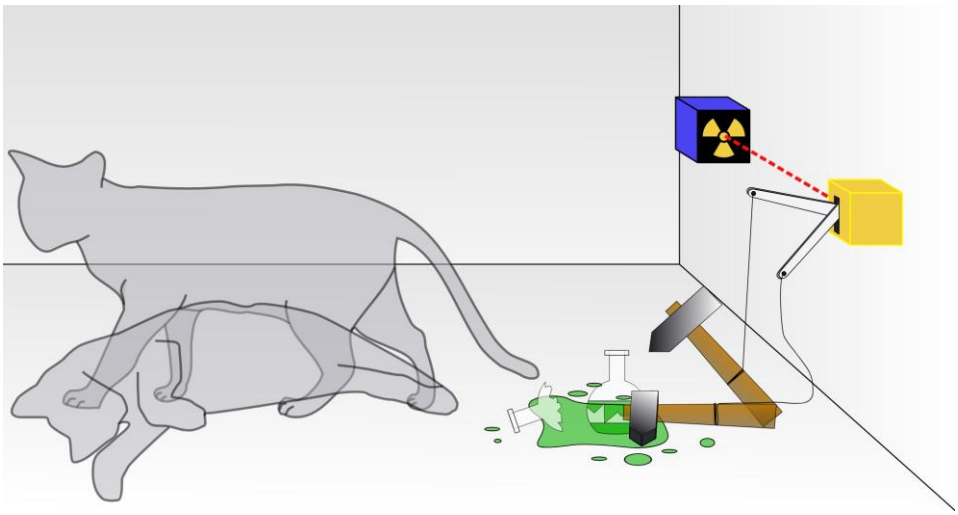

Figura 6 Experimento del gato de Schrödinger

- Otro principio importante para entender la naturaleza del qubit es, **el principio de entrelazamiento**. El entrelazamiento es un fenómeno en el que el estado cuántico de una partícula puede estar intrínsecamente vinculado al estado cuántico de la otra, indiferentemente respecto a la distancia que las separe. Supongamos que superponemos dos qubits tal que, al ser observados, con la misma probabilidad los dos colapsan a 0 o a 1. Si cogemos uno de los qubits y nos lo llevamos al otro lado del mundo, con la misma probabilidad sería 0 o 1 y, al medirlo, tomaría su valor correspondiente e instantáneamente el otro qubit que hemos dejado tomaría el mismo valor.

2.3.2 Puertas cuánticas

Las **puertas cuánticas, son bloques básicos para la construcción de algoritmos cuánticos, que se encargarán de manipular los qubits llevándolos de un estado a otro**. Son análogos a las puertas lógicas de la computación clásica.

Para la representación de las puertas cuánticas, nos valemos de matrices; por ejemplo, para la puerta cuántica "NOT", vamos a definir una matriz A que la represente:

$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Por otro lado, tenemos el estado de un qubit $\alpha|0\rangle + \beta|1\rangle$ escrito en forma de vector tal que: $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$, correspondiendo "α" a la amplitud de $|0\rangle$ y "β" a la de $|1\rangle$, la salida que obtendríamos después de pasarla por nuestra puerta cuántica NOT sería: $\begin{bmatrix} \beta \\ \alpha \end{bmatrix}$.

La respuesta procesada al pasar un qubit por una puerta cuántica la vamos a entender como el producto de la matriz que representa la puerta cuántica por el estado de nuestro qubit.

La operación de la puerta cuántica NOT ha sido reemplazar el estado $|0\rangle$ por la primera columna de la matriz A y el estado $|1\rangle$ por la segunda.

Vista la forma en la que se representan las puertas cuánticas, podemos pensar que combinando matrices de 2x2 de formas diferentes obtendríamos puertas cuánticas diferentes, sin embargo, tenemos una restricción para formar estas matrices: **la matriz debe ser unitaria**, es decir, que sea una matriz cuadrada cuya adjunta sea la inversa de la matriz original: $U^{\dagger} U = I$. Además, tanto la entrada como la salida deben ser estados de un qubit, por lo que la suma de los módulos al cuadrado de los elementos del vector columna deben ser igual a uno también después de la salida.

Antes de seguir, hay que **mencionar otros dos estados fundamentales de los qubits** (aparte de $|0\rangle$ y $|1\rangle$) son **$|+\rangle$**, que está representado en el ecuador, exactamente entre los polos norte y sur en un estado de máxima superposición y **$|-\rangle$**, igual que $|+\rangle$ pero en el lado opuesto. Se representan tal que así:

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

$$\mathbf{|-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)}$$

Vamos a ver algunas de las puertas cuánticas más importantes:

- Correspondiendo al ejemplo de puerta NOT que hemos puesto antes, en computación cuántica, su análogo sería la puerta de Pauli-X o simplemente **puerta X**. Representado matricialmente por:

  $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

  Tiene la propiedad de que si **es aplicada a un qubit que está en el estado |1⟩, cambiará al |0⟩ y viceversa**.

  Como hemos visto en el ejemplo anterior, básicamente transforma el estado $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ en $\begin{bmatrix} \beta \\ \alpha \end{bmatrix}$. Resumidamente su acción es:

  $X|0\rangle = |1\rangle$

  $X|1\rangle = |0\rangle$

  La representamos gráficamente tal que:

  

  Figura 7 Puerta Pauli-X

- La puerta de Pauli-Y o **puerta Y** es otra de las puertas cuánticas fundamentales, esta **realiza una rotación de 180 grados alrededor del eje Y en la esfera de Bloch**, siendo representada matricialmente por:

  $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$, $siendo\ i\ la\ unicidad\ imaginaria.$

  En forma de ecuación, podríamos describir la acción de la puerta Y sobre el estado de un qubit tal que:

  $Y|0\rangle = i|1\rangle$

  $Y|1\rangle = -i|0\rangle$

  La representamos gráficamente tal que:

  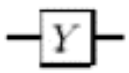

  Figura 8 Puerta Pauli-Y

- La última puerta de Pauli fundamental es la puerta de Pauli-Z o **puerta Z**, representada matricialmente por:

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

Al igual que la puerta Y, esta puerta también **realiza una rotación de 180 grados en la esfera de Bloch, pero alrededor del eje Z**.

La acción que realiza esta puerta sobre el estado de un qubit en forma de ecuación se describe tal que:

$Z|0\rangle = |0\rangle$

$Z|1\rangle = -|1\rangle$

, o bien:

$Z|+\rangle = |-\rangle$

$Z|-\rangle = |+\rangle$

La representamos gráficamente tal que:

Figura 9 Puerta Pauli-Z

- Probablemente la puerta más importante de todas, pues se suele utilizar al principio de todos los algoritmos cuánticos; la **puerta de Hadamard**.

  Representada matricialmente por:

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

  Esta puerta **coloca al qubit en una mezcla cuántica de los estados |0⟩ y |1⟩**; la acción que realiza para los estados principales del qubit, son las siguientes:

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

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

  La puerta de Hadamard siempre colocará el estado del qubit en el ecuador de la esfera de Bloch en una superposición equitativa, dependiendo del estado de partida.

  La representamos gráficamente tal que:

Figura 10 Puerta de Hadamard

- Otra puerta fundamental que veremos es la **puerta CNOT** (Controlled NOT), esta, además, **opera en dos qubits**. Bajo determinada perspectiva podría considerarse

análoga a la puerta XOR en computación clásica, y matricialmente se representa por:

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

Los dos qubits que usa esta puerta como entrada los llamaremos qubit de control y qubit objetivo. Las acciones que realiza sobre las posibles combinaciones de estados entre los dos qubits de entradas son las siguientes:

CNOT$|00\rangle$ = $|00\rangle$

CNOT$|01\rangle$ = $|01\rangle$

CNOT$|10\rangle$ = $|11\rangle$

CNOT$|11\rangle$ = $|10\rangle$

La forma de actuar que tiene es la siguiente: **si el estado del qubit de control es $|0\rangle$, no se realiza ninguna acción sorbe el qubit objetivo, pero si es $|1\rangle$, al estado del qubit objetivo se le aplica una puerta X. El qubit de control no varía.**

La representamos gráficamente tal que:

Figura 11 Puerta C-NOT

Veremos más acerca de cómo proceder con múltiples qubits.

- Otra puerta importante es la **puerta de medición**, que como su propio nombre indica, **se usa para medir el estado del qubit**, por tanto, al ser observado, colapsará hacia un estado u otro, y a partir de ese momento ese qubit pasará a ser observado como un bit.

  La representamos gráficamente tal que:

Figura 12 Puerta de medición

[6], [7]

2.3.3 Múltiples qubits

Como hemos visto al analizar el efecto de la puerta cuántica CNOT, al tener dos qubits consideramos 4 estados base denotados por sus posibles combinaciones de estados: $|00\rangle$, $|01\rangle$, $|10\rangle$ y $|11\rangle$. **Si tenemos n qubits, tendríamos 2^n estados base denotados por sus combinaciones de estados.**

Por supuesto, cada uno de los 4 anteriores con su representación matricial:

$$|00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix},\ |01\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix},\ |10\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix},\ |11\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Para llegar a estos estados de forma matemática tenemos que, "*el estado básico de n qubits es el producto de n estados básicos simples*": $|01\rangle = |0\rangle \otimes |1\rangle$.

El producto tensorial multiplica todos los elementos del vector de la izquierda por todos los elementos del vector de la derecha, tal que, con el ejemplo anterior, se realizarían las siguientes operaciones:

$$|01\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1 & \begin{bmatrix}0\\1\end{bmatrix}\\ 0 & \begin{bmatrix}1\\0\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

Cuando tratamos con la superposición de estos cuatro estados, nos podemos encontrar con que el coeficiente que tenemos que asociar a algún estado computacional base del par de qubits es complejo (a veces llamado **amplitud**), y lo será más conforme debamos tratar la superposición de más estados. El vector que describe el par de qubits y sus estados con sus coeficientes asociados sería el siguiente:

$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$

Igual que vimos cuando tratábamos con un solo qubit en el que la suma de los módulos al cuadrado de las probabilidades de que el resultado que obtengamos cuando midamos el qubit tienda a un estado o a otro debe ser uno, obviamente se sigue aplicando la misma regla, es decir:

$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$

[8], [9]

2.4 RAM clásica

A medida que los computadores empezaban a utilizar aplicaciones más intensas con cada vez más datos, nació como solución al problema la instalación de memorias de acceso aleatorio (RAM, Random Access Memory), las cuales se utilizarían para **servir de almacenamiento temporal de datos** para que la unidad central de procesamiento **(CPU) pueda recuperar los datos que necesite de forma rápida** mientras se ejecutan programas. Esta memoria es volátil, por lo que si la computadora se apaga los datos almacenados en ella se pierden.

Una memoria RAM **consta de una matriz de memoria, un registro de entrada y un registro de salida**. Esta matriz consta de unas celdas de memoria, conteniendo cada una un bit de datos. Cuando la CPU quiere acceder a la RAM, envía la dirección de memoria de la celda en cuestión, a través de las líneas de dirección accediendo mediante el registro de entrada. La dirección de memoria se usa para saber la ubicación exacta de la celda de memoria que contiene los datos deseados. Entonces se iniciará el proceso de decodificación de la dirección de memoria para saber la ubicación física de la celda de memoria a la que hace referencia. Dependiendo de las señales que se hayan enviado a través de las líneas de control, la acción será bien almacenar datos en la celda de memoria (operación de escritura) o bien recuperar datos (operación de lectura) [10]. Hay dos tipos principales de memoria RAM, con diferentes propiedades que las hacen útiles en diversas misiones: SRAM (Static Random Access Memory) y DRAM (Dynamic Random Access Memory) [11].

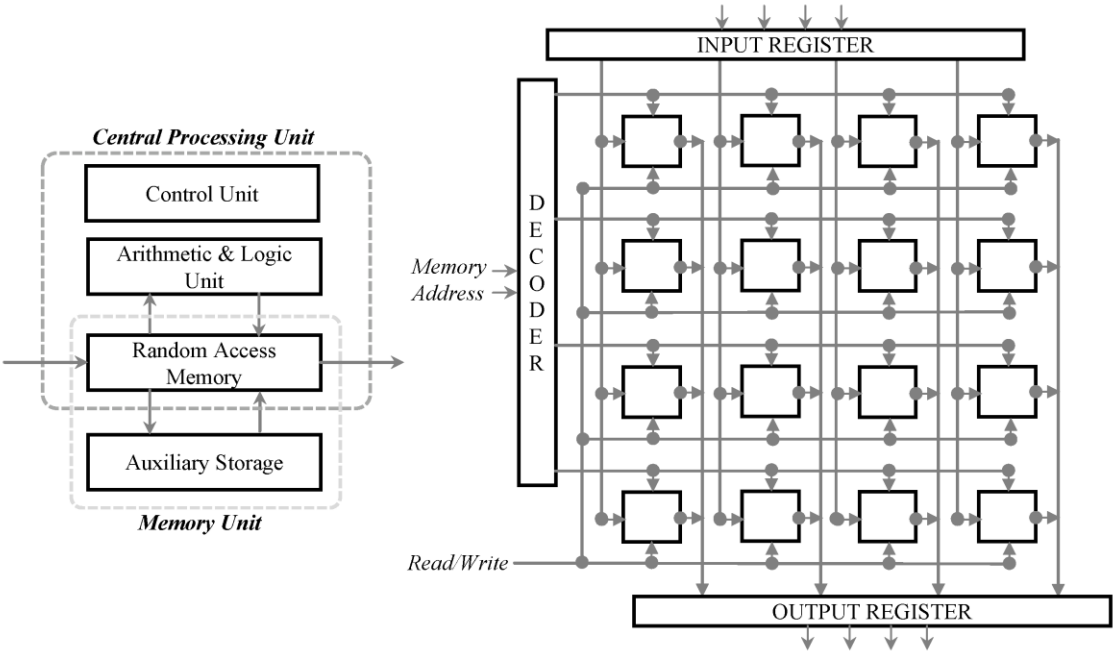

Figura 13 Posición de la RAM en la jerarquía del computador y diagrama funcional del interior de una memoria RAM y sus elementos clave

Como vemos en la jerarquía, la RAM sirve de puente entre la CPU y los sistemas que almacenen la memoria. El registro de entrada almacena temporalmente la información que se va a almacenar en una celda de memoria durante una operación de escritura y el registro de salida almacena la información temporalmente que se lee de una celda de memoria durante la operación de lectura.

Capítulo 3

Quantum RAM y algoritmo FF-QRAM

3.1 Introducción

En este apartado, teniendo presentes cierto conocimientos de la materia, habiendo asentado algunos conceptos tanto de computación arquitectura Von-Newmann como de computación cuántica y tras la revisión de las memorias RAM, profundizaremos en el concepto de RAM cuántica (QRAM); cómo funciona, cómo se estructura y cómo permite la codificación de datos clásicos a cuánticos. También se verán diversas aplicaciones prácticas de la QRAM y qué ventajas aporta.

Seguidamente, se profundizará en el algoritmo Flip-Flop QRAM, primero teóricamente y después se estudiará cómo se implementaría en circuitos cuánticos de forma óptima. También se explicará la técnica de la "bifurcación cuántica" (*Quantum Forking)* para la mejora del rendimiento de algoritmos cuánticos que acceden a datos.

Finalmente, se introducirá el kit de desarrollo software de computación cuántica Qiskit.

3.2 Quantum Random Access Memory

La RAM cuántica, de forma análoga a la RAM clásica, pretende almacenar y manejar datos, pero en un formato cuántico. También **se compone principalmente de un registro de entrada, otro de salida y los vectores de memoria**, sin embargo, **los registros se componen de qubits y los vectores de memoria pueden ser clásicos o**

cuánticas dependiendo del uso que se le vaya a dar a la QRAM [12]. En [13], a modo de ejemplo, podemos ver dos enfoques diferentes para implementar la arquitectura **Fan-Out QRAM**; en uno la QRAM está compuesta de celdas de memoria clásicas de 1 bit que cambian la polarización de los fotones en el registro de salida según el valor almacenado en la celda. En el otro enfoque, las celdas de memoria involucran dos qubits superconductores, uno para almacenar información y otro para extraer.

La **diferencia clave entra la RAM y QRAM es la forma en que la QRAM se aprovecha de la propiedad cuántica de la superposición** para acceder de forma simultánea a múltiples localizaciones de memoria; primero, se utiliza el espacio de Hilbert cuántico (espacio matemático que describe el estado cuántico de un sistema) para cargar todas las direcciones de memoria en un estado de superposición, es decir, tendremos un sistema cuántico que representará de forma simultánea todas las direcciones de memoria posibles; posteriormente, este estado general se pasa a través de la QRAM para obtener otra superposición combinando ahora las direcciones de memoria y los datos de la memoria.

Si tenemos *n* qubits tendremos $N = 2^n$ líneas de dirección, todas las direcciones son representadas como estados base desde |0⟩ hasta |N−1⟩, y son almacenadas en un registro de direcciones *r*. Cada una de estas direcciones |*i*⟩ posee una amplitud de probabilidad asociada α_i. Una superposición efectiva de las direcciones se expresaría como: $\sum_{i=0}^{N-1} \alpha_i \ |i\rangle_r$, siendo α_i la amplitud de la dirección *i* y $|i\rangle_r$ el estado del registro de direcciones. Cuando esta superposición de direcciones se envía a la QRAM, el resultado es otro estado de superposición. Este nuevo estado no solo contiene las direcciones |*i*⟩, sino también los datos asociados a esas direcciones almacenados en un registro de datos *O*. Si X_i (que se codifica en el estado cuántico $|X_i\rangle_o$) representa el dato unidimensional almacenado en la dirección *i*, y $|X_i\rangle_o$ representa el estado cuántico del dato X_i en el registro de datos *O*, el estado de salida de la QRAM se puede expresar como $\sum_{i=0}^{N-1} \alpha_i \ |i\rangle_r \, |X_i\rangle_o$.

Supongamos que tenemos n = 2 qubits en el registro de direcciones. Tendríamos 4 posibles direcciones: |00⟩, |01⟩, |10⟩ y |11⟩.

La superposición de estas direcciones con amplitudes $\alpha_0, \alpha_1, \alpha_2 \, y \, \alpha_3$ sería:

$$\alpha_0|00\rangle_{\mathrm{r}} + \alpha_1|01\rangle_{\mathrm{r}} + \alpha_2|10\rangle_{\mathrm{r}} + \alpha_3|11\rangle_{\mathrm{r}}$$

Supongamos que los datos almacenados en cada dirección son X_1, X_2, X_3 y X_4. La QRAM transforma la anterior superposición en un estado que incluye tanto las direcciones como los datos correspondientes:

$$\alpha_0|00\rangle_r|X_1\rangle_O + \alpha_1|01\rangle_r|X_2\rangle_O + \alpha_2|10\rangle_r|X_3\rangle_O + \alpha_3|11\rangle_r|X_4\rangle_O$$

Este proceso permite que la QRAM acceda y manipule múltiples datos de manera simultánea gracias a la superposición cuántica.

A pesar de la complejidad de la operación de escritura en la QRAM**, el verdadero reto llega en la operación de lectura debido al teorema de no clonación**, que establece que es imposible crear una copia idéntica de un estado cuántico arbitrario sin destruir el estado original [14], por lo que no podemos simplemente copiar el estado cuántico que representa la información almacenada. Para sobrepasar esta barrera, recurrimos a operaciones de entrelazamiento entre los qubits contenidos en las celdas de memoria y los qubits del registro de salida.

3.3 Uso de la QRAM

El problema de los estados cuánticos es que son extremadamente sensibles, y si son expuestos a determinadas perturbaciones externas se genera lo que llamamos decoherencia, [15] que puede ocasionar la pérdida de coherencia de estos estados cuánticos y por tanto termine en la inhabilitación para su uso.

A los computadores cuánticos de hoy en día se les conoce como "***NISQ (Noisy intermediate-scale quantum) computers***", que se les conozca así es debido a que no son lo suficientemente avanzados como para evitar ser susceptibles a los errores causados por ruido, que puede provenir de diferentes fuentes: interferencias electromagnéticas, fluctuaciones térmicas, etc.

En un computador cuántico cuya fidelidad se ha visto degradada por los errores antes comentados, además ocurre que el proceso general de carga de datos puede ir acompañado de ruido, e incluso se podrían almacenar datos de manera inadecuada.

Con una **QRAM se pretende gestionar estos datos de forma fiable y segura.**

Gracias al principio de superposición, una QRAM podría acceder a los datos de forma paralela, lo que reduciría el tiempo de acceso aumentando la resiliencia hacia el ruido.

Es muy importante almacenar y recuperar los estados cuánticos de los bloques de datos para ejecutar de forma eficiente los algoritmos cuánticos.

¿Podríamos usar una RAM para almacenar estados cuánticos? Requeriríamos el colapso de la función de onda con una operación de medición para que asuma un valor clásico (0 o 1) [16], así se podría almacenar en una RAM clásica, pero dejaría de ser útil dentro de un algoritmo cuántico, ya que al colapsar el estado cuántico se pierde mucha información (superposición, entrelazamiento, información cuántica y la escalabilidad se limitaría) y dejaría por tanto de ser valioso.

Por este motivo la QRAM se puede considerar una solución potencial a esta carencia que presenta la RAM clásica, ya que **podríamos almacenar y recuperar estados cuánticos sin la necesidad del colapso de su superposición de estados**. Con este propósito intentamos valernos de técnicas de mecánica cuántica para codificar la información tratando de reducir en la medida de lo posible las fuentes de decoherencia y ruido, [17] consiguiendo almacenar y recuperar estados cuánticos con un error mínimo, lo que convertiría esta RAM cuántica en un componente viable e incluso esencial en computación cuántica.

Una utilidad importante de las QRAM es la capacidad de cargar datos clásicos en un formato cuántico en el espacio de Hilbert, característica muy deseable pues, la gran mayoría de los conjuntos de datos se encuentran actualmente en un formato clásico, siendo inutilizables en algoritmos de *Quantum Machine Learning*. Gracias a las QRAM, se podría establecer un puente entre conjuntos de datos en un formato clásico y algoritmos de *Quantum Machine Learning*.

Existen otros **métodos de codificación** cómo codificación de ángulo (*Angle Embedding*) o codificación de base (*Basis Embedding*) [18] que también han sido planteados para cargar datos clásicos en un formato cuántico. Sin embargo, estos métodos no suelen tener la capacidad de abarcar correctamente este tipo de conjuntos de datos, es por ello por lo que se propone el uso de QRAM para la carga de datos para abordar este problema, pues permite preparar los estados cuánticos de forma que represente de una manera más fidedigna estos datos clásicos.

En QML es crucial que el proceso de codificación de un conjunto de datos clásicos a un marco cuántico sea hecho de la manera más eficiente posible, ya que aprovechamos la velocidad de los computadores cuánticos frente a los clásicos, así como la escalabilidad y una mejor economía de los recursos, pero si la codificación nos deja un conjunto de datos con mucho ruido y/o decoherencia, o que requiera más qubits u operaciones

cuánticas de las necesarias, la complejidad y el costo de los algoritmos cuánticos aumentaría e incluso nuestro conjunto de datos podría quedar inutilizable.

3.4 Flip-Flop QRAM

En [19] se habla de un modelo de QRAM basada en **circuitos cuánticos y flip-flops**, sin la necesidad de algoritmos de enrutamiento, **para codificar un conjunto de datos básicos en una base de datos cuántica de forma flexible y sistemática**. El FF-QRAM funciona como una interfaz entre datos clásicos y cuánticos; puede leer datos clásicos no ordenados de las celdas de memoria y superponerlos o actualizarlos en un formato cuántico, lo que sería muy efectivo a la hora de manipular grandes bases de datos.

El coste de escribir o modificar M datos clásicos representados como el conjunto $D = \{\vec{d}^{(l)} \mid 0 \leq l < M\}$ (donde $\vec{d}^{(l)}$es un representa *n* bits de información) es *O(n)* y *O(Mn)* operaciones cuánticas.

Para cuantificar el coste espacial, teniendo un conjunto de datos establecido por M vectores conteniendo N datos cada uno, son suficientes *n + m* qubits para codificar el conjunto de datos inicial en una base de datos cuántica, donde $\mathrm{m} = \lceil \log_2(\mathrm{M}) \rceil$ y $\mathrm{n} = \lceil \log_2(\mathrm{N}) \rceil$. Como ejemplo, la base de datos que se ha empleado para el seguimiento de la implementación en Qiskit presenta 5 vectores de 4 elementos cada uno. Sabemos que un qubit puede estar en dos estados posibles, $|0\rangle$ o $|1\rangle$, es decir, puede representar dos estados. Si aumenta el número de qubits, aumenta el número de estados que podemos representar, si con 1 qubit podemos representar $|0\rangle$ y $|1\rangle$, con 2 qubits podemos representar $|00\rangle, |01\rangle, |10\rangle\ y\ |11\rangle$. En el caso de la implementación que se ha realizado, si la base de datos presenta 20 datos en total (5 vectores y 4 elementos por vector), se necesitan m qubits tal que $\mathrm{m} = \lceil \log_2(5) \rceil$ y n qubits tal que $\mathrm{n} = \lceil \log_2(4) \rceil$. El resultado de *m* será 3 qubits y el de *n* será 2 qubits, por lo que serán necesarios 5 qubits para codificar el conjunto de datos.

Para referirnos a un estado cuántico de superposición preparado con respecto a datos clásicos usamos el término "*Quantum Database*" (QDB), y constituye nuestro objetivo mediante el uso del algoritmo FF-QRAM. Lo expresamos de esta manera:

$|\psi\rangle_{QDB} = \sum_{l=0}^{M-1} \alpha_l |\vec{d}^{(l)}\rangle$,

donde M es el número de datos y $\vec{d}^{(l)} = \mathrm{d}_0^{(\mathrm{l})}\mathrm{d}_1^{(\mathrm{l})} \ldots \mathrm{d}_{\mathrm{n}-1}^{(\mathrm{l})} \in \{0,1\}^{\mathrm{n}}$ es una cadena de qubits representando un dato clásico (para grandes conjuntos de datos, podemos descomponer $|\vec{d}^{(l)}\rangle$ en $|\vec{y}^{(l)}\rangle|\mathrm{l}\rangle$, siendo $\vec{y}^{(l)}$ una entrada de datos y *l* representa el índice (posición) de dicha entrada de datos en el conjunto de datos). Como ejemplo, se dispone de una base de datos clásica simple con los siguientes datos: [[3.6312], [7.0201]]. $|\vec{y}^{(0)}\rangle$ denota el estado cuántico de la entrada de datos de "[3.6312]" y $|\mathrm{l}\rangle$ representaría el estado cuántico de la posición de la entrada de datos en el conjunto de datos.
La variable α_l corresponde a la amplitud de probabilidad.

3.4.1 Concepto del FF-QRAM y circuito cuántico

Para codificar una base de datos clásicos pensamos en un ordenador cuántico con un estado de los registros de *n+m* qubits.
Representamos una operación para superponer un conjunto de datos clásicos en un estado del bus de qubits tal que:

$$QRAM(D)\sum_{v}\alpha_v|v\rangle_B|0\rangle_R \equiv \sum_{l}\alpha_l\left|\vec{d}^{(l)}\right\rangle_B|b_l\rangle_R$$

- El subíndice *B* se refiere al bus de qubits y el subíndice R al qubit registro.
- α_v corresponde a las amplitudes de probabilidad con que los estados base computacionales del bus de qubits $|v\rangle_B$ son accedidos. Cada v es un índice que recorre todos los estados posibles del bus de qubits.
- $|0\rangle_R$ representa el estado del qubit registro inicialmente como $|0\rangle$. El qubit registro es un qubit auxiliar que se utiliza en una fase posterior del algoritmo, la fase registro. En esta fase, una puerta multicontrolada por los qubits del bus rotará el qubit registro χ radianes sobre un determinado eje de la esfera de Bloch. Esta puerta solo se activará si los qubits de control se encuentran en el estado $|1\rangle$.
- $\vec{d}^{(l)}$ es la representación binaria de la entrada de datos que señala el índice *l*. $\left|\vec{d}^{(l)}\right\rangle_B$ denota una cadena de qubits en su estado base computacional que representa $\vec{d}^{(l)}$ en el conjunto de datos *D*. α_l denotaría las amplitudes de

probabilidad asociada a los estados base computacionales de $\left|\vec{d}^{(l)}\right\rangle_B$, igual que en $|v\rangle_B$.

- $|b_l\rangle_R$ representa un estado arbitrario del qubit registro, tras haberle aplicado la rotación multicontrolada o no.

A continuación, se verá la lógica del algoritmo en un proceso de superponer dos cadenas de bits independientes $\vec{d}^{(l)}$ y $\vec{d}^{(l+1)}$ (como ejemplo, supongamos que $\vec{d}^{(l)}$ es 10000 y $\vec{d}^{(l+1)}$ es 10001, acompañado de ecuaciones matemáticas que representan el estado del circuito cuántico en un determinado paso. Podemos definir el estado inicial como:

$$|\Psi_0\rangle_l = \alpha_{\vec{d}^{(l)}} \left|\vec{d}^{(l)}\right\rangle |0\rangle_R + \sum_{\mathrm{v} \neq \vec{d}^{(l)}} \alpha_v \, |v\rangle |0\rangle_R$$

$|\Psi_0\rangle_l$ representa el estado del bus de qubits y el qubit registro cuando se procede a la codificación de la entrada de datos del conjunto de datos señalada por el índice *l*. A este estado se llega tras determinar los registros cuánticos necesarios para codificar el conjunto de datos. Por ejemplo, tenemos la cadena binaria “10000”, calculada a partir del número de qubits que se haya usado para codificar el conjunto de datos, de tal forma que, si la cadena binaria es “10000”, en este caso serían 5 qubits y estaría representando de forma binaria la iteración actual de la base de datos, que en nuestro conjunto de de datos sería el primer elemento del quinto vector.

- $\vec{d}^{(l)}$ (10000) es la representación binaria de la entrada de datos que señala el índice *l.* $\left|\vec{d}^{(l)}\right\rangle$ denota una cadena de qubits en su estado base computacional que representa esa entrada de datos $\vec{d}^{(l)}$. $\alpha_{\vec{d}^{(l)}}$ denotaría las amplitudes de probabilidad asociada al estado base computacional de $\left|\vec{d}^{(l)}\right\rangle$.
- $|0\rangle_R$ representa el estado del qubit registro inicialmente como $|0\rangle$, explicado anteriormente.
- La suma $\sum_{\mathrm{v} \neq \vec{d}^{(l)}} \alpha_v \, |v\rangle |0\rangle_R$ representaría exactamente el mismo proceso que lo descrito antes de la suma, pero para el resto de entradas de datos, omitiendo $\vec{d}^{(l)}$.

Para el siguiente estado, se aplican puertas X a los qubits del bus, siendo activadas si el bit de control de los elementos controladores de $\vec{d}^{(l)}$ es 0. Es decir, reorganizaría los

estados base computacionales de los qubits del bus convirtiendo $|\vec{d}^{(l)}\rangle$ en $|1\rangle^{\otimes n}$ y, consecuentemente el resto de qubits cambia (realiza "flips", en el caso de que el bit de control sea "0") coherentemente:

$$|\Psi_1\rangle_l = \alpha_{\vec{d}^{(l)}}|1\rangle^{\otimes n}|0\rangle_R + \sum_{|v\oplus\vec{d}^{(l)}\rangle\neq|1\rangle^{\otimes n}} \alpha_v \overline{|v\oplus\vec{d}^{(l)}\rangle}|0\rangle_R$$

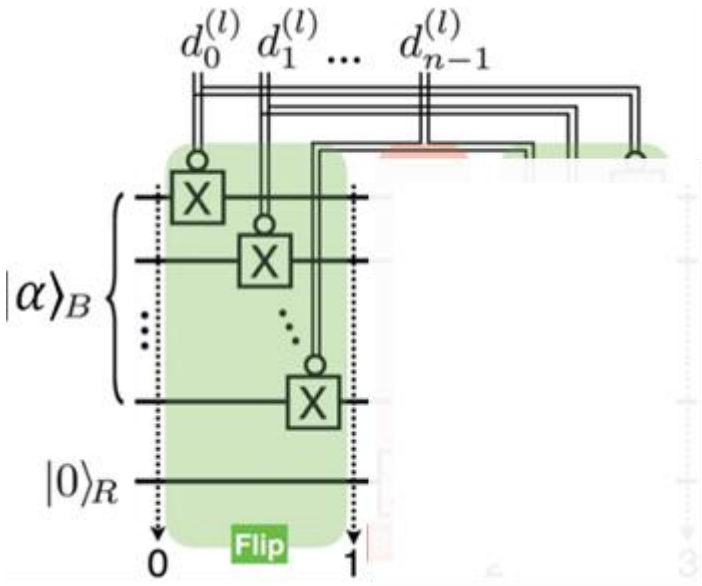

Figura 14 Circuito que genera el estado $|\Psi_1\rangle_l$

En $|\Psi_1\rangle_l$, la barra superior se usa para denotar que el flip sucede si el bit de control es 0. A continuación, se aplica la rotación multicontrolada (calculada realizando el arcoseno por dos del dato clásico que se esté codificando) por los qubits del bus (representado como $\chi^{(l)}$) al qubit registro, si los qubits de control se encuentra en el estado $|1\rangle$:

$$|\Psi_2\rangle_l = \alpha_{\vec{d}^{(l)}}|1\rangle^{\otimes n}\left|\chi^{(l)}\right\rangle_R + \sum_{|v\oplus\vec{d}^{(l)}\rangle\neq|1\rangle^{\otimes n}} \alpha_v \overline{|v\oplus\vec{d}^{(l)}\rangle}|0\rangle_R,$$

$$|\chi\rangle = cos\,\chi|0\rangle + sin\,\chi|1\rangle$$

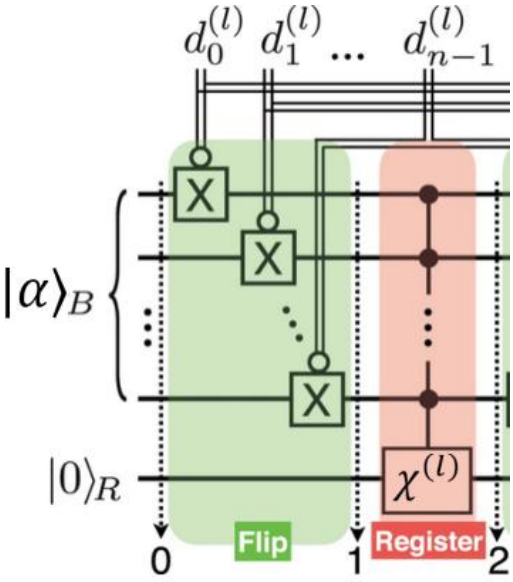

Figura 15 Circuito que genera el estado $|\Psi_2\rangle_l$

Volvemos a hacer uso de puertas X controladas por los elementos de $\vec{d}^{(l)}$, para revertir (flop) el estado de los qubits del bus:

$$|\Psi_3\rangle_l = \alpha_{\vec{d}^{(l)}} \left|\vec{d}^{(l)}\right\rangle \left|\chi^{(l)}\right\rangle_{\mathrm{R}} + \sum_{\mathrm{v} \neq \vec{d}^{(l)}} \alpha_v \, |\mathrm{v}\rangle|0\rangle_R$$

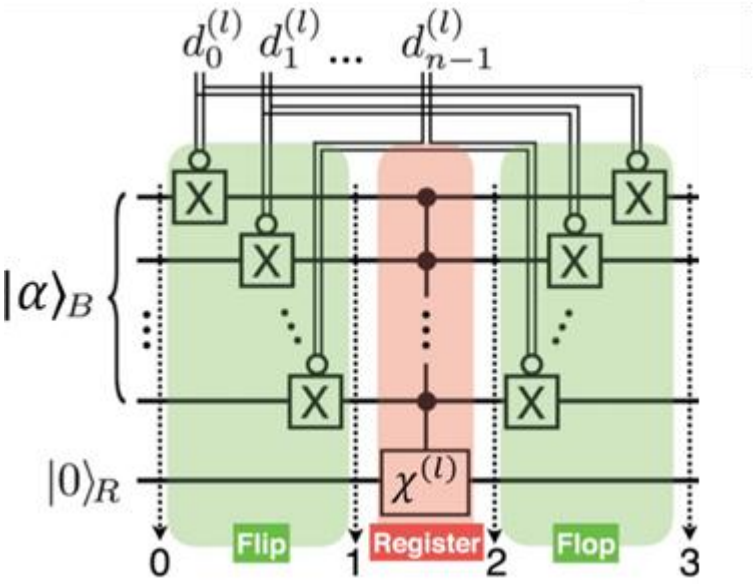

Figura 16 Circuito que genera el estado $|\Psi_3\rangle_l$

Repetimos este proceso con la siguiente entrada de datos $\vec{d}^{(l+1)}$ y la siguiente puerta que aplique una rotación multicontrolada al qubit registro $\chi^{(l+1)}$:

$$|\Psi_4\rangle_l = \alpha_{\vec{d}^{(l)}} \left|\vec{d}^{(l)}\right\rangle \left|\chi^{(l)}\right\rangle_{\mathrm{R}} + \alpha_{\vec{d}^{(l+1)}} \left|\vec{d}^{(l+1)}\right\rangle \left|\chi^{(l+1)}\right\rangle_{\mathrm{R}} + \sum_{\mathrm{v} \neq \vec{d}^{(l)}, \vec{d}^{(l+1)}} v \, |\mathrm{v}\rangle|0\rangle_R$$

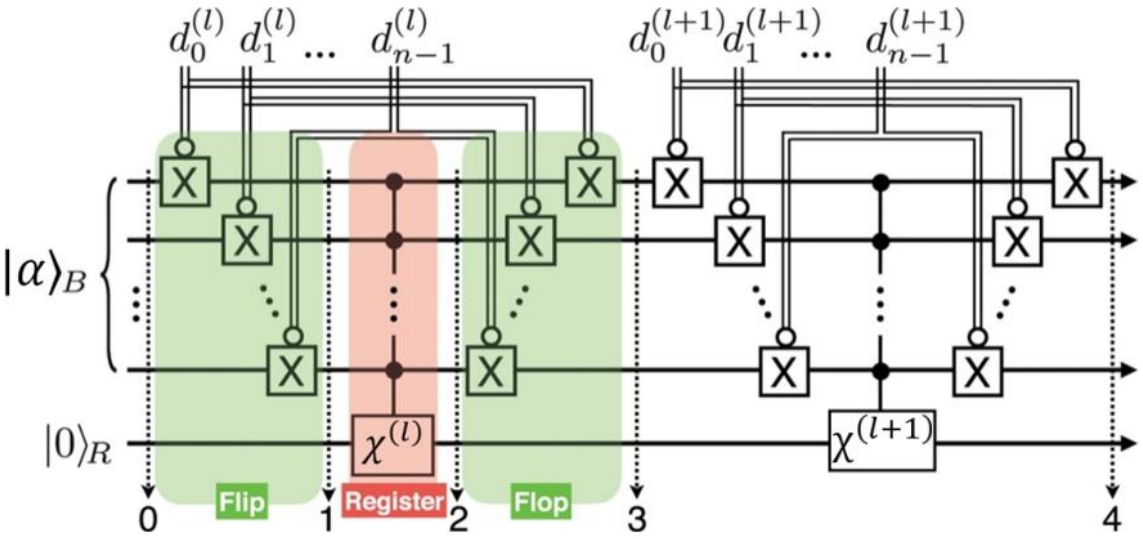

Figura 17 Circuito que genera el estado $|\Psi_4\rangle_l$

Repitiendo este proceso podemos registrar *M* entradas de datos que queramos con pesos no uniformes; podríamos representar el proceso repetido para M entradas mediante la ecuación:

$$\sum_{l=0}^{M-1} \alpha_{\vec{d}^{(l)}} \left|\vec{d}^{(l)}\right\rangle [cos\chi^{(l)}|0\rangle_{\mathrm{R}} + sin\chi^{(l)}|1\rangle_{\mathrm{R}}\,] + \sum_{v \notin \{\vec{d}^{(l)}\}} \alpha_v\,|\mathrm{v}\rangle|0\rangle_R$$

Eligiendo un ángulo adecuado para $\chi^{(l)}$ haciendo que coincida la amplitud de probabilidad querida α_v y post-seleccionando el resultado de la medición de $|1\rangle_{\mathrm{R}}$ podemos obtener la QDB consultada derivada de $|\psi\rangle_{QDB} = \sum_{l=0}^{M-1} \alpha_l |\vec{d}^{(l)}\rangle$.

La probabilidad de medición de $|1\rangle_{\mathrm{R}}$ viene dada por la expresión:

$$P(1) = \sum_{l=0}^{M-1} |\alpha_{\vec{d}^{(l)}} sin\chi^{(l)}|^2$$

La post-selección aumenta el tiempo total de ejecución en un factor aproximado de *1/P(1) (*al tener que repetir el proceso), aunque sujeto a mejora, por ejemplo, preprocesando los datos clásicos para que $\chi^{(l)}$ quede cerca de *kπ/2* para todas las entradas de datos en las que *k* sea un número entero impar.

Para los datos que se encuentren en una superposición uniforme, es decir, con amplitudes de probabilidad iguales, podemos reemplazar las rotaciones controladas con

puertas X, implicando que los datos clásicos solo se codifican en forma digital. Sin embargo, hay que tener en cuenta que si el bus de qubits no se encuentra en un estado base que corresponda a una entrada de datos que se encuentre en $\vec{d}^{(l)}$, la *l* entrada de datos no puede ser escrita en la QDB consultada. Si la misma cadena de bits se repite, se acumula la rotación en el qubit registro.

Si queremos actualizar unas determinadas entradas de datos de la QDB, habría que tomar el estado de la QDB como el bus de qubits y solo se realizaría el flip-flop selectivo con los estados base que se quisieran actualizar.

En cuanto al costo del algoritmo, la eficiencia durante la fase de registro ($\chi^{(l)}$), es crucial para la practicabilidad del modelo. Si dicha fase es eficiente, nuestro método podría codificar amplitudes de probabilidad complejas. El número de puertas básicas que se empleen para implementar dicho paso puede marcar la diferencia entre que el algoritmo posea un tiempo de ejecución elevado y que no sea así. En [19] se mencionan algunas implementaciones de la fase de registro para la optimización de recursos del algoritmo, aunque por supuesto, dependerá de nuestra configuración experimental específica.

3.4.2 Quantum Forking

Se menciona el concepto "**Quantum Forking**" (bifurcación cuántica) [20] para solucionar el problema de tener que repetir procesos de preparación de un mismo estado cuántico para varios fines, ya sea porque es necesario en diferentes algoritmos o simplemente porque se requiere muestrear la respuesta del estado debido al principio de superposición (que nos impide reutilizar un estado cuántico para otra tarea una vez medido) y al teorema de no clonación (que declara la imposibilidad de crear una copia idéntica del estado cuántico preparado). Tener que repetir procesos de preparación de un mismo estado cuántico, aunque sea para fines diferentes, provoca un aumento notable en la redundancia; mediante la bifurcación cuántica se consigue que un qubit experimente procesos simultáneos e independientes de forma coherente en superposición, reduciendo las consultas a la QRAM.

Vamos a ver un ejemplo en el que el QF lograría una mejora de rendimiento en el cálculo del producto interno: tenemos el estado cuántico $|\Psi_0\rangle = |0\rangle|\Phi\rangle|\chi\rangle$ donde $|0\rangle$ representa un qubit inicial en el estado $|0\rangle$, $|\Phi\rangle$ corresponde a un estado de una QDB

de *n* qubits originado por la QRAM y $|\chi\rangle$ a un estado arbitrario de *n* qubits. Empezamos pasando el qubit en el estado $|0\rangle$ por una puerta Hadamard para que su estado sea $(|0\rangle + |1\rangle)/\sqrt{2}$, partiendo de una superposición uniforme; ahora usaremos este qubit como controlador en un intercambio entre $|\Phi\rangle$ y $|\chi\rangle$ (proceso conocido como *swap test*), formando el estado entrelazado $|\Psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle|\Phi\rangle|\chi\rangle + |1\rangle|\chi\rangle|\Phi\rangle)$

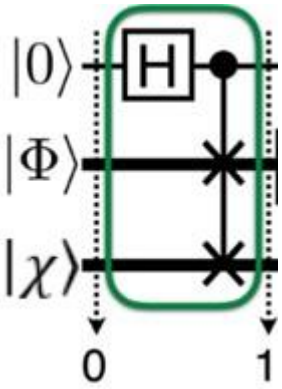

Figura 18 Representación del estado $|\Psi_1\rangle$ de QF

Aplicamos dos evoluciones unitarias (proceso por el cual un estado cuántico evoluciona en el tiempo bajo la influencia de un operador unitario) activadas por diferentes estados base computacionales del qubit de control tanto a $|\chi\rangle$ como a $|\Phi\rangle$. Con esto, obtendremos una superposición de dos estados diferentes a modo de bifurcación: $|\Psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle \mathrm{U}_1|\Phi\rangle|\chi\rangle + |1\rangle|\chi\rangle U_2|\Phi\rangle)$

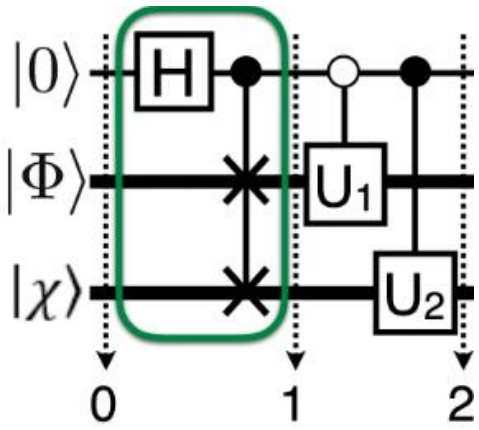

Figura 19 Representación del estado $|\Psi_2\rangle$ de QF

Sin realizar bifurcación cuántica, sería necesario preparar los estados respectivos de las evoluciones unitarias ($|\Phi_1\rangle = \mathrm{U}_1|\Phi\rangle$ y $|\Phi_2\rangle = \mathrm{U}_2|\Phi\rangle$) por separado para evaluar su producto interno, requiriendo, en este caso específico, realizar el doble de consultas a la QRAM. Con QF, partiendo desde $|\Psi_2\rangle$, se vuelve a utilizar el qubit de control para otro

intercambio entre $|\Phi\rangle$ y $|\chi\rangle$ y además se le aplica al qubit de control otra puerta de Hadamard, obteniendo $|\Psi_3\rangle = \frac{1}{2}[|0\rangle\ (|\Phi_1\rangle + |\Phi_2\rangle) + |1\rangle(|\Phi_1\rangle - |\Phi_2\rangle)]\,|\chi\rangle$

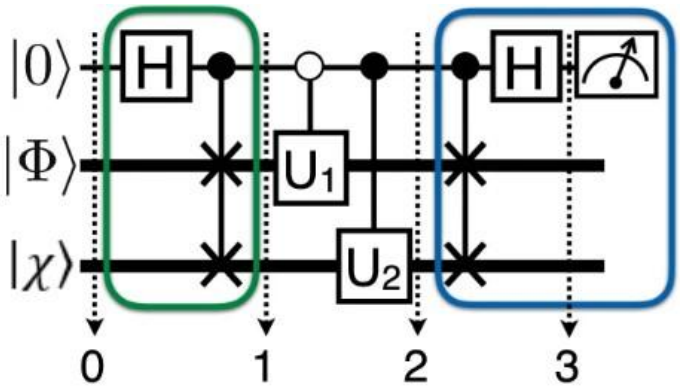

Figura 20 Representación del estado $|\Psi_3\rangle$ de QF

Obtenemos la probabilidad de que el qubit de control colapse a $|0\rangle$ dada por la expresión: $P(0) = \frac{1+\mathrm{Re}(\langle\Phi_1|\Phi_2\rangle)}{2}$

El empleo de bifurcación cuántica tiene un costo de *O(n)* puertas adicionales, impedimento que merece la pena ya que reduce las consultas a la QRAM en un factor aproximado de ½, gracias a no tener que repetir procesos de preparación de un mismo estado cuántico. Adicionalmente, la bifurcación cuántica permite la identificación del signo del producto interno. Esto es gracias a la capacidad que ofrece el circuito de QF de poder aplicar operadores unitarios diferentes a cada rama (subespacio), de forma que la fase global que introduce un operador unitario se vuelve indistinguible.

3.5 Qiskit

Para nuestras implementaciones y simulaciones, vamos a usar el **kit de desarrollo software** creado por IBM llamado Qiskit. Qiskit es utilizado para trabajar con computadores cuánticos a nivel de circuitos, algoritmos y módulos de aplicación, permitiéndonos programar en computadores cuánticos reales desde nuestro ordenador. Es el kit de desarrollo software que usaremos para nuestra implementación del algoritmo FF-QRAM para su uso en *Quantum Machine Learning.*

Capítulo 4

Implementación

4.1 Introducción

En este apartado se verán de forma detallada, las 2 implementaciones que culminan este trabajo, así como explicaciones concretas y numerosos experimentos para poder evaluar resultados, finalmente unas conclusiones respecto a los resultados obtenidos y a los experimentos realizados junto con el desempeño obtenido.

4.2 Análisis y explicación del código

Para la implementación del algoritmo, he empleado una base de datos ad-hoc para poder representar de forma gráfica los procesos y manipulaciones que sus datos van sufriendo y así poder realizar un seguimiento del código visualmente accesible. El código, como ya se ha dicho, ha sido implementado con el kit de desarrollo software Qiskit en Python, en un cuaderno de Jupyter.

4.2.1 Importaciones y módulos

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister,
transpile
from qiskit.visualization import plot_state_qsphere
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram, plot_state_qsphere
from sklearn.preprocessing import MinMaxScaler
import numpy as np
```

```
import matplotlib.pyplot as plt
import math

data = [
    [3.6574, 7.2431, 5.1223, 2.4924],
    [6.8273, 1.9453, 9.1743, 4.8321],
    [5.0324, 3.7123, 8.5214, 6.1232],
    [4.9212, 9.2345, 3.4213, 7.3421],
    [8.2314, 2.3912, 5.8921, 4.7345]
]
```

Se han empleado las importaciones mostradas en la figura superior, a su vez, se han instalado módulos para su uso tales como “Qiskit”, “Numpy”, “Math” y “Sklearn”. Para ver su efectividad, he probado con la base de datos “Iris”, sin embargo, como he dicho antes, para el correcto seguimiento de la implementación y con el fin de hacerlo entendible y accesible, en esta memoria se verá con la base de datos mostrada en la figura superior, mucho más simple, pero más demostrativa.

4.2.2 Paso 1: Normalización

```
# PRIMER PASO: normalizamos los datos clasicos para un mejor tratamiento,
haciendo que esten comprendidos entre -1 y 1
def normalizing_step(data):
    myScaler = MinMaxScaler(feature_range=(-1, 1))
    normalized_data = myScaler.fit_transform(data)
    return normalized_data
```

La primera función a la que se llama en el código tiene como parámetro la base de datos que se le quiera pasar, en nuestro caso ya se ha mostrado. La función consiste en pasar nuestra base de datos por un “MinMaxScaler”, que los normalizará, escalando los datos individualmente dentro del rango -1 y 1 para su posterior tratamiento.

4.2.3 Paso Opcional: Proyección estereográfica inversa.

```
def inverse_stereo_transform(data):
    k = len(data[0]) #nº elementos en cada array
    l = len(data) #nº de arrays
    adapted_data = []
    for i in range(l): #iteramos sobre la lista de arrays
         s = sum(data[i]**2) #suma los cuadrados de los componentes del
array en s
```

```
        for j in range(k):
            data[i][j] = 2*data[i][j] #duplica los elementos
        adapted_data.append([x for x in data[i]]) #crea una copia de data[i]
y se añade en adapted_data
         adapted_data[i] = np.append(data[i], np.array(s-1)) #añadimos el
valor calculado en s -1
        adapted_data[i] = adapted_data[i]/(s+1) # y lo normalizamos
        print("adapted_data["+str(i)+"] = "+ str(adapted_data[i]))
```

Dependiendo del algoritmo de *Quantum Machine Learning* (sería ideal uno relacionado con clustering que, consiste en agrupar un conjunto de datos en subgrupos, llamados clusters, de manera que los objetos dentro de un mismo cluster posean más similitudes entre sí que con los objetos de otros clusters), podríamos pasar nuestros datos por otro tipo de normalización llamado **transformación o proyección estereográfica inversa** [21]. Este tipo de normalización se basa en un tipo de enfoque que se aplica en algoritmos de *Machine Learning*, en el que los sets de datos son vistos como conjuntos de datos distribuidos en un espacio o dimensión; en el caso concreto de los algoritmos de clustering, por ejemplo, para la Proyección Estereográfica Inversa, mapearíamos colecciones de datos de un espacio dimensional *N* hacia una esfera en un espacio dimensional *N+1*. Una dimensión de un orden mayor nos ayudará a preservar los ángulos y las similitudes entre los datos de la colección original, de una mejor manera que otras técnicas como la distancia Euclídea en la que se conserva el espacio dimensional, y con la que, probablemente, algunos puntos de diferentes colecciones de datos queden solapados.

4.2.4 Paso 2: Circuito cuántico

```
def initializing_circuit(data):
    K = len(data) #nº de arrays postadaptacion
    print("\n")
    print("Tamaño de K // Nº de arrays postadaptacion: " + str(K))

    L = len(data[0]) #nº de elementos de datos en cada array postadaptacion
    print("\n")
    print("Tamaño de L // Nº de elementos en cada array postadaptacion: "
+ str(L))

    print("\n")

    #para codificar una base de datos de K vectores y L elementos en cada
vector se necesitara encontrar #el numero minimo de qubits tal que 2^k >=
K, por ello hacemos k >= log2(K) y redondeamos hacia arriba #para encontrar
este numero minimo, y lo mismo con L

    k = math.ceil(math.log2(K))
    print("k: numero de qubits necesario para indexar K arrays: " + str(k))
    l = math.ceil(math.log2(L))
    print("l: numero de qubits necesario para indexar L elementos en cada
array: " + str(l))

    r = QuantumRegister(1, 'R') #qubit registro al que aplicamos la rotacion
controlada
    i = QuantumRegister(k, 'K') #registros para indexar los K arrays
    j = QuantumRegister(l, 'l') #registros para indexar los L elementos de
cada array
    c = ClassicalRegister(k+l, 'c') #registros clásicos

    Q = QuantumCircuit(j,i,r,c) #circuito cuantico para ver nuestros datos
pasados por FF-QRAM

    FFQRAM(Q, data, j, i, r)



    #display(Q.draw(output='mpl', reverse_bits=False))

    return Q
```

Seguidamente, pasamos nuestros datos a la siguiente función para empezar a inicializar el circuito cuántico. Así tanto para su seguimiento como para poder corregir el código de forma eficiente, se opta por obtener impresiones tratando de equilibrar la obtención de información e impedir mucha sobrecarga de resultados en la salida del código.

El propósito general del cuaderno es crear una base de datos cuántica con los datos de la base de datos clásica pasada:

$$|\psi\rangle_{QDB} = \sum_{l=0}^{M-1} \alpha_l |\vec{d}^{(l)}\rangle$$

- $\vec{d}^{(l)} = \mathrm{d}_0^{(\mathrm{l})}\mathrm{d}_1^{(\mathrm{l})} \ldots \mathrm{d}_{\mathrm{n}-1}^{(\mathrm{l})} \in \{0,1\}^{\mathrm{n}}$ es una cadena de qubits en la base computacional representando un dato clásico.
- *l* representa el índice (posición) de dicha entrada de datos en el conjunto de datos.
- *M* es la cantidad de datos clásicos.
- α_l, representa la amplitud de probabilidad para la entrada de datos del índice *l*.

La función comienza obteniendo el número de vectores que hay en la matriz de datos y el número de elementos que tiene cada vector, para posteriormente calcular el número de qubits necesario para codificar estos datos. Siguiendo la explicación matemática proporcionada en la sección 3.4 de este mismo trabajo, se obtienen las fórmulas para obtener *k + l* qubits: $k = \lceil log_2(\mathrm{K}) \rceil$ y $l = \lceil log_2(\mathrm{L}) \rceil$, siendo K el número de vectores y L el número de elementos por vector

Posteriormente, se definen los registros cuánticos siendo estos todos los qubits obtenidos con el cálculo anterior más un qubit adicional, el qubit registro, cuya utilidad se comentará más adelante. Definimos nuestros registros clásicos donde volcaremos los resultados de las mediciones de los qubits. En este caso concreto no se realizarán mediciones, ya que carece sentido colapsar los qubits hacia un estado cuando tienen que estar preparados para ser utilizados en un algoritmo de *Quantum Machine Learning* que se aplique posteriormente. Finalmente, declaramos nuestro circuito cuántico con los elementos previamente definidos.

Seguidamente, sometemos nuestro circuito cuántico y datos al algoritmo FF-QRAM en su respectiva función (explicada en su sección).

4.2.5 Paso 3: Función FF-QRAM

```
def FFQRAM(circuit, data, l_qregisters, k_qregisters, qubitRY):
    #list_crtl_qubits = [l_qregisters, k_qregisters]
    list_crtl_qubits = l_qregisters[:] + k_qregisters[:]
    circuit.h(l_qregisters)
    circuit.h(k_qregisters)
    for iteratingK, array in enumerate(data):
        #flip stage para los registros cuanticos K que representa el nº de
arrays
        binary_pattern1 = '{:0%sb}' % len(k_qregisters)
        encoded_data1 = binary_pattern1.format(iteratingK)
        print("\n")
        print("encoded_data for flipK: " + str(encoded_data1))
        for pos, bit in enumerate(encoded_data1):
            if not int(bit): #si el bit es 0
                circuit.x(k_qregisters[pos]) #le aplicamos una puerta X al
registro cuantico cuya posicion
                                             #corresponda con la del bit
en la cadena

        #display(circuit.draw(output='mpl', reverse_bits=False))

        for iteratingL in range(len(array)):
            #flip stage para los registros cuanticos L que representa el
nº de elementos en cada array
            binary_pattern2 = '{:0%sb}' % len(l_qregisters)
            encoded_data2 = binary_pattern2.format(iteratingL)
            print("\n")
            print("encoded_data for flipL: " + str(encoded_data2))
            for pos, bit in enumerate(encoded_data2):
                if not int(bit): #si el bit es 0
                    circuit.x(l_qregisters[pos]) #le aplicamos una puerta
X al registro cuantico cuya posicion
                                                 #corresponda con la del
bit en la cadena

            circuit.barrier()
                                        #display(circuit.draw(output='mpl',
reverse_bits=False))

            #aplicamos la rotacion multicontrolada al qubit registro
            #empezamos calculando "theta" que es el numero de grados que
vamos a rotar el eje y de la esfera de Bloch
            theta = np.arcsin(array[iteratingL])*2
            circuit.mcry(theta, list_crtl_qubits, qubitRY)
            circuit.barrier()

            #display(circuit.draw(output='mpl', reverse_bits=False))
```

```
            #flop stage para los registros cuanticos L que representa el
nº de elementos en cada array
            binary_pattern2 = '{:0%sb}' % len(l_qregisters)
            encoded_data2 = binary_pattern2.format(iteratingL)
            print("\n")
            print("encoded_data for flopL: " + str(encoded_data2))
            for pos, bit in enumerate(encoded_data2):
                if not int(bit): #si el bit es 0
                    circuit.x(l_qregisters[pos]) #le aplicamos una puerta
X al registro cuantico cuya posicion corresponda con la del bit en la cadena

            #display(circuit.draw(output='mpl', reverse_bits=False))
            circuit.barrier()

#flop stage para los registros cuanticos K que representa el nº de arrays
        binary_pattern3 = '{:0%sb}' % len(k_qregisters)
        encoded_data3 = binary_pattern3.format(iteratingK)
        print("\n")
        print("encoded_data for flopK: " + str(encoded_data3))
        for pos, bit in enumerate(encoded_data3):
            if not int(bit): #si el bit es 0
                circuit.x(k_qregisters[pos]) #le aplicamos una puerta X al
registro cuantico cuya posicion corresponda con la del bit en la cadena
        #display(circuit.draw(output='mpl', reverse_bits=False))
```

El algoritmo FF-QRAM encargado de convertir nuestra base de datos clásica en cuántica. Su lógica queda representada mediante la siguiente ecuación matemática:

$$QRAM(D)\sum_{v}\alpha_v|v\rangle_B|0\rangle_R \equiv \sum_{l}\alpha_l\left|\vec{d}^{(l)}\right\rangle_B|b_l\rangle_R$$

donde:

- α_v corresponde a la amplitud de probabilidad con que los estados base computacionales del bus de qubits $|v\rangle_B$ son accedidos. Cada *v* es un índice que recorre todos los estados posibles del bus de qubits.
- El subíndice *B* se refiere al bus de qubits y el subíndice R al qubit registro.
- $|0\rangle_R$ representa el estado del qubit registro inicialmente como $|0\rangle$. El qubit registro es un qubit auxiliar que se utiliza en una fase posterior del algoritmo, la fase registro. En esta fase, una puerta multicontrolada por los qubits del bus rotará el qubit registro χ grados sobre un determinado eje de la esfera de Bloch.

Esta puerta solo se activará si los qubits de control se encuentran en el estado $|1\rangle$.

- $\left|\vec{d}^{(l)}\right\rangle_B$ denota una cadena de qubits en su estado base computacional que representa un dato clásico del conjunto de datos *D*. $\vec{d}^{(l)}$ es la representación binaria del dato específico que señala el índice *l*. α_l denotaría la amplitud de probabilidad asociada a los estados base computacionales de $\left|\vec{d}^{(l)}\right\rangle_B$, igual que en $|v\rangle_B$.
- $|b_l\rangle_R$ representa un estado arbitrario del qubit registro, tras haberle aplicado la rotación multicontrolada o no.

Es importante empezar el circuito cuántico aplicando puertas Hadamard a los registros cuánticos del bus para partir de una superposición uniforme. Si no partimos de una superposición uniforme, no se ejecutaría la fase registro.

El algoritmo, como ya se ha visto, consta de tres pasos fundamentales: fase flip, fase registro y fase flop. En el código se realiza la fase flip para los registros cuánticos correspondientes al número de vectores del conjunto de dato. Para cada iteración del bucle de la fase flip (correspondiendo cada iteración a uno de los vectores del conjunto de datos) realizamos las tres fases seguidas (flip, rotación multicontrolada al qubit registro y flop) para cada elemento contenido dentro del vector al que esté señalando el bucle. Una vez iterados todos los elementos del vector, se termina el bucle correspondiente, y termina también una iteración del bucle inicial realizando la fase flop al vector.

4.2.6 Ejecución del código y resultados

```
data = normalizing_step(data)
print(data)
print("\n")
#data= inverse_stereo_transform(data)
print("\n")
print(data)
q_circuit = initializing_circuit(data)


#Ejecución del circuito
#backend = Aer.get_backend('qasm_simulator')
```

```
backend = Aer.get_backend('statevector_simulator')
transpiled_circ = transpile(q_circuit, backend)
result = backend.run(transpiled_circ, shots=1024).result()

#Mostramos el circuito
display(q_circuit.draw(output='mpl', reverse_bits=False))


#Obtenemos los resultados

counts = result.get_counts()
print("\nResultados de la ejecución del circuito:")
print(counts)
plot_histogram(counts)

statevector = result.get_statevector()
statevector_array = np.asarray(statevector)

print("Statevector:\n", statevector)
# Calcular las probabilidades
probabilities = np.abs(statevector)**2

# Obtener los estados (bits)
states = [bin(i)[2:].zfill(5) for i in range(len(statevector_array))]

#Gráfico de barras
fig, ax = plt.subplots(figsize=(20, 10))
bar_width = 0.4
positions = np.arange(len(states))
ax.bar(positions, probabilities, width=bar_width, color='blue')
ax.set_xticks(positions)
ax.set_xticklabels(states, rotation=90)
plt.xlabel('Estados')
plt.ylabel('Probabilidad')
plt.title('Probabilidades de los Estados Cuánticos')
plt.show()
```

En el siguiente bloque de código, los datos serán procesados por las diferentes funciones de este cuaderno, explicadas previamente. Para ejecutar el circuito:

- Primero, hay que declarar el backend que se va a usar. Un backend es un simulador de un ordenador cuántico. En este caso, el backend empleado ha sido “statevector_simulator”. Este simulador cuántico permite la simulación de circuitos cuánticos mediante la representación del estado cuántico del circuito como un vector de estado.

- El siguiente paso es transpilar el circuito. La transpilación es el proceso de transformar un circuito cuántico a una forma específica que sea compatible con un simulador cuántico específico.
- Entonces se ejecuta el circuito, realizando 1024 ejecuciones. Se realiza un número elevado de ejecuciones con el fin de obtener una estimación de las probabilidades de los diferentes estados cuánticos.

- Seguidamente, se mostrará tanto un esquema del propio circuito cuántico como una gráfica visual de los resultados.

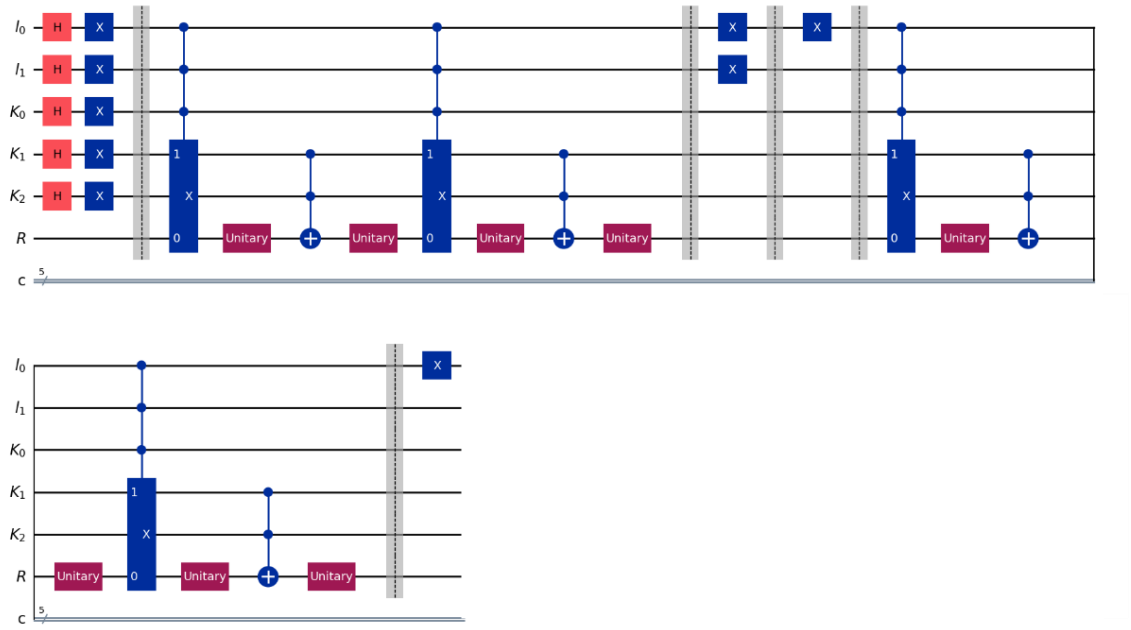

Figura 21 Fragmento del esquema del circuito cuántico

La figura previa corresponde a un fragmento del circuito cuántico producido por la implementación realizada en este trabajo, que representa la preparación de los registros cuánticos y la aplicación del algoritmo sobre las dos primeras entradas de datos. No se ha optado por mostrar el esquema completo producido en el cuaderno, pues es demasiado grande y la idea fundamental ya está aquí representada.

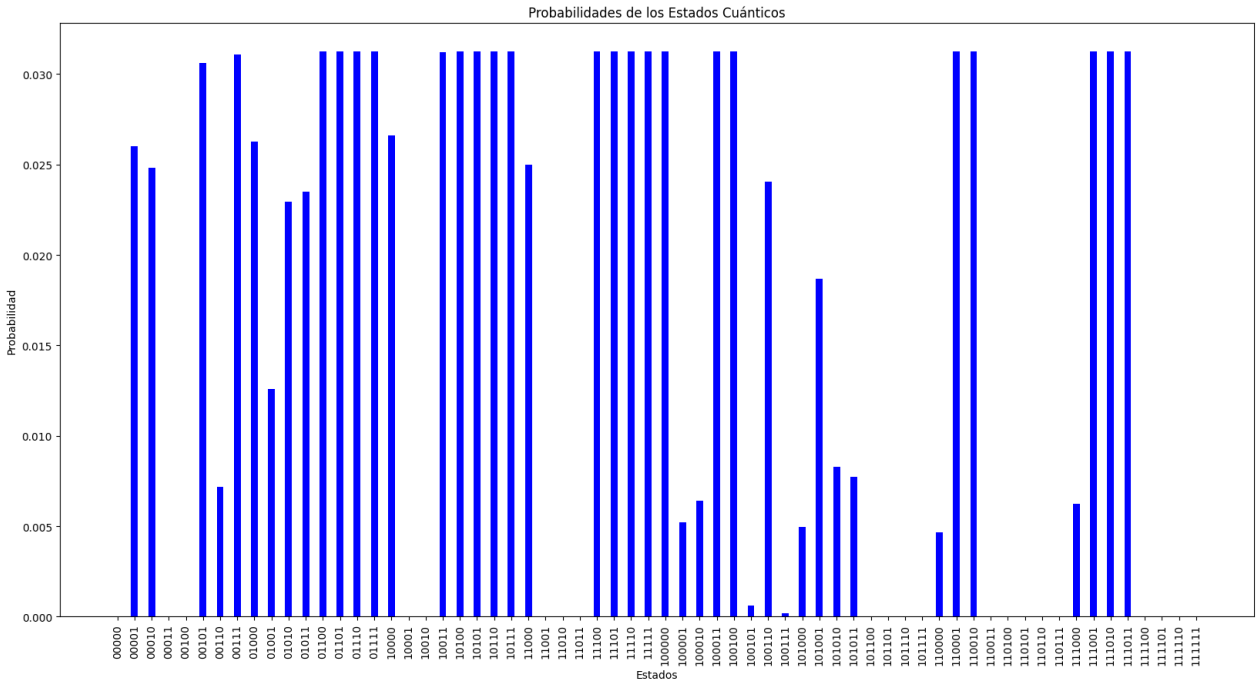

Figura 22 Gráfico de barras de las amplitudes de probabilidad de los estados del vector de estado del circuito

Para el gráfico de barras, se ha calculado la probabilidad de cada estado cuántico mediante el cuadrado del módulo de las amplitudes de probabilidad mostradas en el vector de estado. Este vector de estado se obtiene gracias al backend "statevector_simulator".

Mediante este gráfico de barras que muestra la probabilidad de cada estado cuántico, obtenida con el cálculo descrito anteriormente, podemos confirmar que se ha codificado cada uno de los valores del conjunto de datos clásico, por lo que quedarían preparados para utilizarlos en un algoritmo de *Quantum Machine Learning*.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1 Conclusiones

El objetivo principal de este trabajo ha sido estudiar y entender el algoritmo FF-QRAM e implementarlo para su uso en *Quantum Machine Learning*. La idea nace de los presentes desafíos significativos relacionados con la representación eficiente de datos y la optimización de algoritmos cuánticos, retos que se pretenden abordar con este algoritmo de *encoding* específico.

Las aportaciones realizadas mediante este trabajo Fin de Grado han sido las siguientes:

- Una descripción razonada y didáctica del algoritmo FF-QRAM. Esta descripción está dirigida a ingenieros en informática, con el objetivo de que puedan comprender los conceptos y el funcionamiento del algoritmo sin necesidad de tener un conocimiento profundo en computación cuántica.
- Se ha llevado a cabo la implementación práctica del algoritmo FF-QRAM. Esta implementación está orientada para su posible uso en el futuro en algoritmos de *Quantum Machine Learning*. Se ha utilizado una base de datos de tamaño pequeño en la implementación del algoritmo y se ha obtenido un esquema del circuito cuántico que ha resultado de la implementación. El motivo de utilizar una base de datos pequeña es que, con el tamaño, requiere una cantidad

recursos computacionales de los que no disponemos. También se ha obtenido un gráfico de barras que muestra la distribución de probabilidad de cada estado cuántico, habiendo simulado la ejecución 1024 veces, debido a la naturaleza probabilística de este paradigma de cómputo.

A pesar del arduo estudio y trabajo que este paradigma requiere, y específicamente el algoritmo FF-QRAM, la codificación de datos clásicos para su uso en algoritmos de *Quantum Machine Learning* sigue enfrentando retos y presenta desafíos significativos por superar, como puede ser la escalabilidad.

Aunque la curva de aprendizaje del paradigma de computación cuántica resulta dura al principio, las ventajas que aporta y sus perspectivas sobradamente compensan el esfuerzo.

5.2 Trabajo futuro

Concluimos este capítulo y el TFG con los trabajos futuros que creemos pueden ser interesantes tras desarrollar este documento:

- Implementar otros algoritmos de QRAM y compararlos con FF-QRAM.
- Utilizar en la práctica FF-QRAM para manipular y analizar una base de datos clásica con un algoritmo de QML. Idealmente, podría ser uno relacionado con clustering aprovechando que se ha implementado la “Proyección Estereográfica Inversa” que es un tipo de normalización de datos que funciona especialmente bien con este tipo de algoritmos.
- Complementar y mejorar el código generado, mediante técnicas como la “bifurcación cuántica” con el objetivo de reducir el número de veces que hay que preparar un estado cuántico.

5.3 Competencias

Con este Trabajo Fin de Grado, se han desarrollado las siguientes competencias de la tecnología específica de computación:

- **[CM1]** Capacidad para tener un conocimiento profundo de los principios fundamentales y modelos de la computación y saberlos aplicar para

interpretar, seleccionar, valorar, modelar, y crear nuevos conceptos, teorías, usos y desarrollos tecnológicos relacionados con la informática.

Dicha competencia se trabaja:

- En la necesaria comprensión de los principios y conceptos “poco intuitivos” y extremadamente rigurosos de la Computación Cuántica.
- En la comprensión del algoritmo FF-QRAM hasta traducirlo al dominio de un no-experto en Computación Cuántica.
- Implementar y analizar las prestaciones del algoritmo FF-QRAM.
- Aplicar los conceptos de algoritmia al dominio de Qiskit.

- **[CM3]** Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.
  - El propio algoritmo FF-QRAM tiene como objetivo generar una base de datos cuántica a partir de una base de datos clásica para que pueda ser procesada por algoritmos de QML. Uno de los fines de emplear técnicas y algoritmos de QML es obtener potenciales ventajas computacionales.
  - Se estudia la complejidad computacional del algoritmo en función del uso de registros cuánticos y posibles optimizaciones usando técnicas como la “bifurcación cuántica” para reducir el número de consultas a la QRAM.
- **[CM7]** Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.
  - Aunque su uso sea opcional en el código, la implementación de la “Proyección Estereográfica Inversa” y su aplicación en el algoritmo FF-QRAM, denota el conocimiento en técnicas de aprendizaje computacional, siendo ideal para algoritmos de clustering cuántico.

Bibliografía

[1] L. Chen, S. Zhang, and L. Li, “A brief introduction to quantum algorithms,” *CCF Transactions on High Performance Computing*, vol. 4, no. 1, pp. 53–62, Mar. 2022, doi: 10.1007/S42514-022-00090-3.

[2] Y. Kim *et al.*, “Evidence for the utility of quantum computing before fault tolerance,” *500 | Nature |*, vol. 618, 2023, doi: 10.1038/s41586-023-06096-3.

[3] D. Castelvecchi, “Google’s quantum computer hits key milestone by reducing errors,” *Nature*, Feb. 2023, doi: 10.1038/D41586-023-00536-W.

[4] D. Castelvecchi, “IBM releases first-ever 1,000-qubit quantum chip,” *Nature*, vol. 624, no. 7991, p. 238, Dec. 2023, doi: 10.1038/D41586-023-03854-1.

[5] L. Zhu, “Quantum Computing: Concepts, Current State, and Considerations for Congress,” 2023, Accessed: Jun. 30, 2024. [Online]. Available: https://crsreports.congress.gov

[6] G. Alonso, “(47) Introducción a las puertas cuánticas - YouTube.” Accessed: Jul. 01, 2024. [Online]. Available: https://www.youtube.com/watch?v=73iQffY35VI&t=260s&ab_channel=Ket.G

[7] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2000.

[8] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2000.

[9] G. Alonso, “(47) Trabajando con varios Qubits - YouTube.” Accessed: Jul. 01, 2024. [Online]. Available: https://www.youtube.com/watch?v=qAkg4PEnyNE&ab_channel=Ket.G

[10] D. A. Patterson and J. L. Hennessy, “In Praise of Computer Organization and Design: The Hardware/ Software Interface, Fifth Edition,” 2014.

[11] William. Stallings, *Computer organization and architecture : designing for performance*. Prentice Hall, 2010.

[12] V. Giovannetti, S. Lloyd, and L. MacCone, “Quantum random access memory,” *Phys Rev Lett*, vol. 100, no. 16, p. 160501, Apr. 2008, doi: 10.1103/PHYSREVLETT.100.160501/FIGURES/3/MEDIUM.

[13] V. Giovannetti, S. Lloyd, and L. MacCone, “Architectures for a quantum random access memory,” *Phys Rev A*, vol. 78, no. 5, p. 052310, Nov. 2008, doi: 10.1103/PHYSREVA.78.052310/FIGURES/7/MEDIUM.

[14] W. K. Wootters and W. H. Zurek, “A single quantum cannot be cloned,” *Nature 1982 299:5886*, vol. 299, no. 5886, pp. 802–803, 1982, doi: 10.1038/299802a0.

[15] M. Schlosshauer, “Quantum decoherence,” *Phys Rep*, vol. 831, pp. 1–57, Oct. 2019, doi: 10.1016/J.PHYSREP.2019.10.001.

[16] J. Von Neumann, *Mathematical foundations of quantum mechanics*, vol. 613. Princeton, N.J. : Princeton University Press, 1955.

[17] C. T. Hann, G. Lee, S. M. Girvin, and L. Jiang, “Resilience of Quantum Random Access Memory to Generic Noise,” *PRX Quantum*, vol. 2, no. 2, p. 020311, Apr. 2021, doi: 10.1103/PRXQUANTUM.2.020311/FIGURES/11/MEDIUM.

[18] M. Schuld and F. Petruccione, “Quantum Science and Technology Supervised Learning with Quantum Computers”, Accessed: Jul. 01, 2024. [Online]. Available: http://www.springer.com/series/10039

[19] D. K. Park, F. Petruccione, and J. K. K. Rhee, “Circuit-Based Quantum Random Access Memory for Classical Data,” *Scientific Reports 2019 9:1*, vol. 9, no. 1, pp. 1–8, Mar. 2019, doi: 10.1038/s41598-019-40439-3.

[20] D. K. Park, I. Sinayskiy, M. Fingerhuth, F. Petruccione, and J. K. K. Rhee, “Parallel quantum trajectories via forking for sampling without redundancy,” *New J Phys*, vol. 21, no. 8, p. 083024, Aug. 2019, doi: 10.1088/1367-2630/AB35FB.

[21] K. Eybpoosh, M. Rezghi, and A. Heydari, “Applying inverse stereographic projection to manifold learning and clustering,” *Applied Intelligence*, vol. 52, no. 4, pp. 4443–4457, Mar. 2022, doi: 10.1007/S10489-021-02513-0/TABLES/5.