



UNIVERSIDAD DE GRANADA

Facultad de Ciencias

GRADO EN FÍSICA

TRABAJO FIN DE GRADO

SUPERCOMPUTACIÓN Y SIMULACIÓN DE ALTO RENDIMIENTO DE CIRCUITOS CUÁNTICOS

Presentado por:
D^a. Noelia Sánchez Gómez

Curso Académico 2023/2024

Resumen

El avance en la computación cuántica requiere herramientas de simulación clásicas para evaluar los nuevos ordenadores cuánticos. Motivado por este reto, este trabajo se centra en métodos de simulación de circuitos cuánticos aleatorios usando los algoritmos de Schrödinger y Schrödinger-Feynman, evaluando la eficiencia en tiempo y memoria en hardware de supercomputación, específicamente en un nodo de superordenador y una GPU. Se han buscado los parámetros adecuados para cada método, como el número de puertas fusionadas (3-4) y la paralelización en hebras, cuyo comportamiento óptimo depende del circuito. Con esto se han comprobado los tamaños de sistema para los que mejor se aprovechan los recursos de las arquitecturas y algoritmos. Los resultados muestran que, aunque el algoritmo de Schrödinger es mejor para sistemas pequeños, el de Schrödinger-Feynman es superior para sistemas mayores en memoria y tiempo. Además, la GPU exhibe una reducción en los tiempos respecto al nodo en el método de Schrödinger para sistemas grandes, gracias a su alta capacidad de paralelización. Estas observaciones amplían el conocimiento para la simulación eficiente de circuitos, destacando el potencial del algoritmo de Schrödinger-Feynman y abriendo las puertas a nuevas cuestiones.

Abstract

The advancement in quantum computing requires classical simulation tools to evaluate new quantum computers. Motivated by this challenge, this work focuses on methods for simulating random quantum circuits using the Schrödinger and Schrödinger-Feynman algorithms, assessing efficiency in terms of time and memory on supercomputing hardware, specifically on a supercomputer node and a GPU. Appropriate parameters for each method were sought, such as the number of fused gates (3-4) and thread parallelization, with optimal performance depending on the circuit. This allowed for verifying the system sizes that best utilize the resources of each algorithm and architecture. The results show that while the Schrödinger algorithm is better for small systems, the Schrödinger-Feynman algorithm is superior for larger systems in terms of memory and time. Additionally, the GPU demonstrates a reduction in times compared to the node in the Schrödinger method for large systems, thanks to its high parallelization capability. These observations expand the knowledge for efficient circuit simulation, highlighting the potential of the Schrödinger-Feynman algorithm and opening the door to new questions.

Índice

1	Introducción	4
1.1	Orígenes de la computación cuántica	4
1.2	Progreso experimental y ventaja cuántica	5
1.3	Necesidad de la computación de alto rendimiento	6
1.4	Estructura del trabajo	7
2	Preliminares técnicos	8
2.1	Qubits. Esfera de Bloch y operadores unitarios.	8
2.2	Modelo Cuántico de Computación: Circuitos	10
2.2.1	Universalidad	11
2.2.2	Fusión de puertas cuánticas	12
2.3	Eficiencia y complejidad algorítmica	12
2.3.1	Notación O	12
2.3.2	Clases de complejidad	13
3	Computación de alto rendimiento	14
3.1	Unidad central de procesamiento	14
3.2	Unidad de procesamiento gráfico	17
3.3	Factores que afectan al rendimiento	19
4	Problema práctico	20
4.1	Objetivo	20
4.2	Descripción del circuito simulado	20
4.3	Métodos de simulación	22
4.3.1	Qsim. Algoritmo de Schrödinger	22
4.3.2	Qsimh. Algoritmo híbrido Schrödinger-Feynman	23
5	Resultados y discusión	24
5.1	Algoritmo de Schrödinger	25
5.1.1	Requisitos de memoria RAM	25
5.1.2	Eficiencia de la paralelización en hebras	26
5.1.3	Optimización del parámetro de fusión de puertas	28
5.1.4	Comparación CPU vs GPU	30
5.1.5	Efecto del entrelazamiento cuántico	31
5.2	Algoritmo Schrödinger-Feynman. Comparación con Schrödinger	31
6	Conclusiones	34
A	Profiler: Scalene	39
B	Incertidumbres. Desigualdad de Hoeffding	40

1 Introducción

El campo de la computación cuántica, que combina la física y la informática, ha evolucionado con fuerza desde su concepción en los años 80. Su objetivo es aprovechar las propiedades de la mecánica cuántica, como la superposición o el entrelazamiento, para realizar cálculos que son imposibles en una computadora clásica.

1.1 Orígenes de la computación cuántica

La analogía cuántica de los ordenadores clásicos fue tratada por varios investigadores, entre ellos el matemático Yuri Manin en 1980 [1], que compartió su idea de un autómata cuántico que aprovechara las propiedades de la superposición y el entrelazamiento. A su vez, Paul Benioff presentó un hamiltoniano mecánico-cuántico microscópico como modelo de máquinas de Turing [2]. En los años siguientes, surgirían modelos teóricos y nuevos algoritmos basados en estas ideas.

En 1985 un físico de la Universidad de Oxford, David Deutsch, también describió una máquina de Turing cuántica [3]. Demostró que la teoría cuántica y la computadora cuántica universal son compatibles con el principio físico subyacente a la hipótesis de Church-Turing: ‘todo sistema físico realizable finitamente puede ser perfectamente simulado por una máquina de cálculo universal que opera por medios finitos’. Una computadora cuántica podría entonces realizar los mismos cálculos que una clásica, además de poder resolver algunos problemas de manera más eficiente.

Años después, comenzaron a surgir los primeros algoritmos cuánticos que superaban a sus versiones clásicas. Peter Shor, matemático, desarrolló en 1994 un algoritmo para la factorización de números enteros [4]. Es un problema que clásicamente no puede resolverse en un tiempo polinomial¹ y en base al cual se han diseñado muchas de las técnicas de criptografía actuales. En 1996 nació el algoritmo de Grover [5], que supuso una mejora cuadrática en los tiempos de búsqueda de datos en bases desordenadas. Este algoritmo es un caso especial de *Quantum Walks* o caminatas cuánticas [6], [7], las cuales aprovechan la superposición y la interferencia para mejorar la eficiencia de búsqueda. También tienen un *hitting time* exponencialmente más rápido que en el caso clásico. El *hitting time* es el tiempo medio que se tarda en alcanzar una posición concreta. Con estos algoritmos se puede trabajar en espacios abstractos como los hipercubos, pero para entender la gran diferencia que suponen respecto a los métodos clásicos nos trasladamos a una dimensión.

En la Figura 1 se comparan las caminatas aleatorias clásica y cuántica. En ambas el caminante tiene una moneda clásica (cuántica) y el resultado de lanzarla (operar con ella) numerosas veces definirá su camino a seguir. Como en una caminata aleatoria la probabilidad de ir a la derecha o a la izquierda es de $1/2$ (cara o cruz), se obtiene una distribución gaussiana, mientras que en el caso cuántico la distribución final está determinada tanto por la moneda empleada como por el

¹Este tipo de problemas se denominan en teoría de complejidad como tipo NP. Se describirán en la sección 2 debido a su importante relación con la demostración de la superioridad cuántica.

estado inicial. De esta manera, el caminante clásico se encontrará generalmente en posiciones cercanas a la posición de partida, mientras que el caminante cuántico llegará sobre todo a los extremos de la línea.

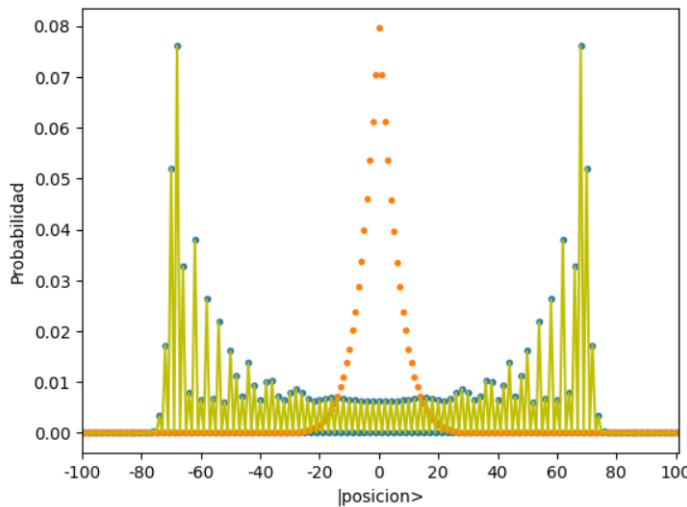


Figura 1: Distribución de probabilidad de una caminata aleatoria clásica (naranja) y cuántica (azul y verde) con 100 pasos. La gráfica ha sido obtenida por la autora.

1.2 Progreso experimental y ventaja cuántica

Las aplicaciones computacionales a las distintas ramas de la física se expanden a pasos agigantados desde hace años. Un vistazo a los Trabajos de Fin de Grado en Física propuestos para el curso 2023/2024 [8], nos da la pista de que tener una base en programación y simulación será indispensable en el futuro. Concretamente, una de las ramas que más se han visto implicadas es la Mecánica Cuántica.

La computación cuántica ha conseguido grandes logros y, a día de hoy, es posible hacer uso de ordenadores cuánticos de unos pocos qubits, la unidad de información mínima de estos sistemas. Sin embargo, queda un largo camino por delante, ya que la capacidad de estos computadores se ve afectada por la decoherencia cuántica debido a interacciones entre los qubits físicos y su entorno [9]. Podemos ver un ejemplo del efecto de la decoherencia en los resultados en la Figura 2, en la que se ha calculado la distribución de probabilidad de una caminata cuántica clásicamente y en un ordenador cuántico de IBM. Se observa que aparecen muchas más posiciones accesibles en la segunda (muchas más barras) y los datos no concuerdan. En ausencia de máquinas cuánticas que funcionen correctamente y dada la reducida disponibilidad de las que están en desarrollo, los simuladores cuánticos son herramientas cruciales para el diseño de los algoritmos cuánticos y la evaluación de su comportamiento en comparación con los ordenadores cuánticos reales.

Una cuestión crítica en el campo de la computación cuántica es si los ordenadores que consigan sobrepasar la barrera de la decoherencia podrán realizar tareas que superen las capacidades de las computadoras clásicas de alto nivel, alcanzando la ventaja cuántica. Estos recursos tan poderosos no estarán exentos de la discusión ética sobre su uso, como ha sucedido con otros grandes inventos a lo largo de la historia. En las manos incorrectas, podrían ser usados para romper los

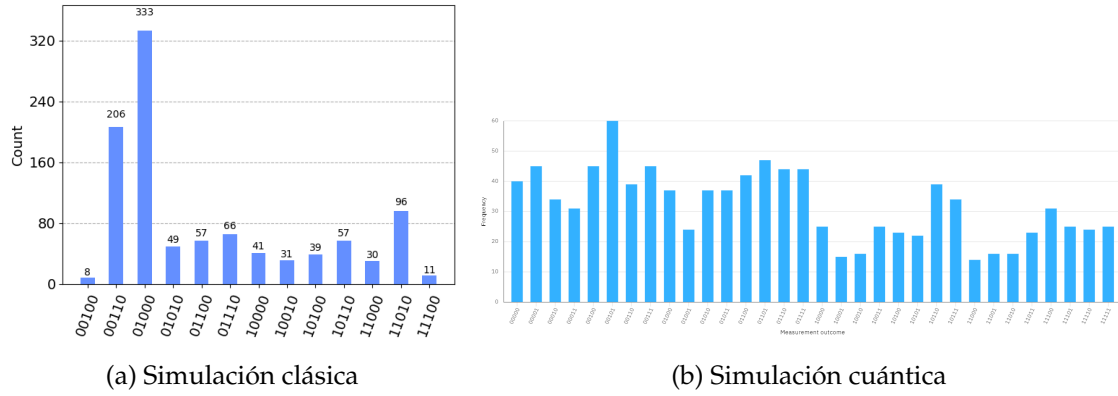


Figura 2: Amplitudes de probabilidad obtenidas a) en un ordenador clásico y b) en un ordenador cuántico. Cada barra corresponde a un estado.

algoritmos que aseguran las claves y firmas digitales de todo el mundo; la criptografía clásica se basa en la suposición de que nadie puede resolver cierto problema matemático difícil en un tiempo realista, lo que si sería posible cuánticamente [10].

Uno de los indicios de esta ventaja es que, a pesar de un siglo de investigación, aún se desconoce la manera de simular en un ordenador clásico ciertos sistemas cuánticos de manera eficiente. Es el caso del cálculo de amplitudes de circuitos cuánticos aleatorios. En 2019, el equipo de Google [11] demostró experimentalmente la aceleración exponencial en la simulación de estos circuitos al ser implementados en su ordenador cuántico Sycamore. Estimaron que tomaría aproximadamente 10000 años ejecutar la tarea en la supercomputadora clásica más potente del mundo, Summit, mientras que solo se requirieron 200 segundos en el Sycamore.

1.3 Necesidad de la computación de alto rendimiento

Como la simulación de sistemas cuánticos es un problema computacional costoso, se requiere el uso de sistemas de computación de alto rendimiento. Estos sistemas permiten hacer simulaciones físicas de una complejidad que no se había alcanzado hasta la fecha. Para hacer una comparación justa con los sistemas cuánticos, las simulación en los sistemas clásicos deben estar perfectamente optimizada, algo que no puede conocerse con seguridad. Por ejemplo, en [12] (2020), se emplea un algoritmo clásico basado en contracción de tensores con el que se estima que la tarea antes mencionada toma únicamente 20 días, en contraposición a los 10000 años que tomaba en el Sumit con el método de Google. Además, en casos de un tamaño moderado se consigue superar al ordenador cuántico Sycamore.

La dificultad radica en comprender como aprovechar eficientemente estas arquitecturas. Para un físico que dedique parte de su trabajo a las simulaciones, sería de una gran utilidad conocer la estructura interna de la máquina que emplea, permitiéndole optimizar su código de manera que se adecúe a una arquitectura concreta. Sin entrar en muchos detalles, ya que en la sección 3 se habla más en profundidad sobre arquitecturas, se muestra un ejemplo. Los ordenadores cuentan con muchos tipos de memoria diferentes, que siguen una jerarquía en la que

las memorias más cercanas a los núcleos donde se procesan los datos son más pequeñas, pero también más rápidas. Además, cuanto más lejos se encuentre la memoria donde está la información, mayor es el tiempo de latencia que transcurre hasta que la información llega a los núcleos. Por lo tanto, una estrategia que puede mejorar la eficiencia es dividir los datos en bloques que se ajusten al tamaño de las memorias más cercanas. Si hablamos de un caso de multiplicación de matrices de grandes dimensiones, una tarea bastante común, se podría enfocar esta idea de 'bloque' dividiendo las matrices en submatrices de menor tamaño. Esto no solo mejoraría el rendimiento, sino también el uso de recursos o la paralelización de la tarea. El uso de GPUs es cada vez más frecuente en los diferentes campos de la física, visto su gran potencial para paralelizar. Por ejemplo, hay trabajos en astrofísica [13], electromagnetismo [14], neurofísica [15], estado sólido [16] o física de la atmósfera [17].

1.4 Estructura del trabajo

Antes de entrar en la parte técnica se procede a enunciar la estructura de este Trabajo de Fin de Grado: en la sección 2, se dan las herramientas necesarias para entender el concepto de qubit y su función en los circuitos cuánticos, abordando la importancia de la universalidad de sus elementos. Además, da una pincelada de teoría de complejidad algorítmica para comprender la importancia del estudio de ciertos problemas, como el de este trabajo. En la sección 3 se introduce al lector en la computación de alto rendimiento (HPC), esencial para simular circuitos de gran tamaño. En esta sección se ofrece una pequeña inmersión en las arquitecturas del hardware empleado. Se explican los principales componentes de las unidades centrales de procesamiento y la unidad de procesamiento gráfico utilizadas, CPUs y GPU respectivamente, de manera que se comprendan sus jerarquías de acceso a la información.

En la sección 4 se presentan los circuitos cuánticos aleatorios de interés, los mismos utilizados por Google para su demostración de la ventaja cuántica [11], así como los métodos de simulación seguidos para calcular las amplitudes resultantes. El objetivo de este enfoque es identificar las limitaciones inherentes a la simulación clásica de estos circuitos y optimizar los parámetros involucrados a través de herramientas HPC.

Finalmente, en la sección 5 de Resultados y Discusión se exponen las distintas simulaciones realizadas y la optimización de los parámetros de estudio, justificando los resultados a través del conocimiento de las arquitecturas empleadas y comparando los sistemas.

2 Preliminares técnicos

2.1 Qubits. Esfera de Bloch y operadores unitarios.

De la misma manera que el concepto de bit es la base de la computación clásica, el bit cuántico o qubit es esencial en la computación cuántica [18]. Actualmente, los qubits pueden encontrarse en forma de sistemas físicos reales gracias al uso de técnicas diversas. Sin embargo, en este trabajo es de interés el concepto de qubit como objeto matemático, abordando a continuación sus propiedades.

Análogamente a un bit, un qubit puede encontrarse en el estado $|0\rangle$ o en el estado $|1\rangle$, pero al contrario que clásicamente, estos no son los únicos estados posibles. Concretamente, un qubit puede encontrarse en una combinación lineal, o superposición, de los estados mencionados:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle. \quad (2.1)$$

Los estados $|0\rangle$ y $|1\rangle$ conforman una base ortonormal para este espacio vectorial:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (2.2)$$

Para n qubits se procede de manera similar, siendo el orden en la base el valor en binario de los elementos. Se toma como ejemplo el caso de un sistema de n qubits. La base tendrá $2^3 = 8$ elementos $\{|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle\}$ en el orden señalado por convenio.

Los términos α y β en 2.1 son números complejos, y $|\psi\rangle$ es el estado del qubit. Acorde con los postulados fundamentales de la mecánica cuántica, al realizar una medida la función de onda colapsa a uno de los estados de la base. Los dos resultados posibles son el valor 0, con una probabilidad $|\alpha|^2$, y el valor 1, con probabilidad $|\beta|^2$. Además, el estado debe estar normalizado, cumpliéndose $|\alpha|^2 + |\beta|^2 = 1$. Gracias a esta condición, se puede expresar la Ecuación 2.1 como

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \right), \quad (2.3)$$

donde θ , ϕ y γ son números reales. Desde un punto de vista observacional, el factor de fase global $e^{i\gamma}$ puede ignorarse, visto que no afecta a las propiedades observadas del sistema físico, obteniendo

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle. \quad (2.4)$$

Para el caso tratado, en el que únicamente se trabaja con un qubit, este estado se puede representar de manera sencilla en lo que se conoce como esfera de Bloch, como se muestra en la Figura 3. Dicha esfera está definida por los parámetros θ y ϕ , ofreciendo una representación geométrica intuitiva del estado.

Para aplicar una rotación a nuestro estado en la esfera de Bloch se hace uso de operadores unitarios conocidos como puertas cuánticas, análogas a las puertas

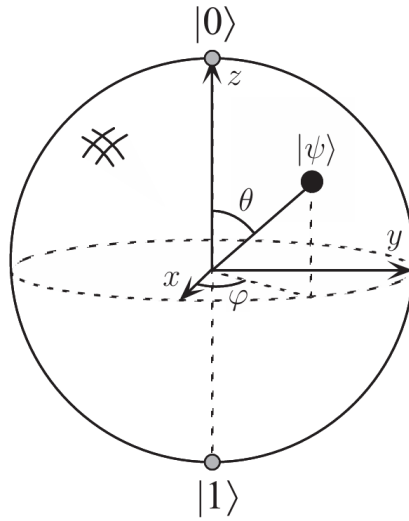


Figura 3: Representación de un qubit en la esfera de Bloch [18].

lógicas clásicas. Estas puertas se representan mediante matrices, con la única restricción de que sean unitarias. Además, la idea de rotación se puede extender a un sistema de n qubits, ya que se conserva el módulo del vector de estado. Un sistema de n qubits consta de una base de la forma $|x_1 x_2 \dots x_n\rangle$, es decir, es un producto tensorial entre los espacios de Hilbert de cada qubit, $\mathbb{C}_1^2 \otimes \mathbb{C}_2^2 \otimes \dots \otimes \mathbb{C}_n^2 = \mathbb{C}^{2^n}$. Entonces, la dimensión de la base aumenta según 2^n . Para entenderlo de manera intuitiva, se propone un ejemplo: si se trabaja en un sistema de dos qubits, el resultado final tras evolucionar la función de onda, antes de medir, será una superposición de los posibles estados accesibles. En el vector de salida se guardará la probabilidad de que la función de onda colapse a cada uno de los estados, por lo que habrá una entrada por cada uno de ellos. La cantidad de estados que pueden surgir de n qubits es 2^n ; en nuestro sistema de dos qubits cada uno puede colapsar a 0 o a 1, pudiendo dar lugar a los estados $|00\rangle$, $|01\rangle$, $|10\rangle$ y $|11\rangle$, un total de 2^2 estados. Esta propiedad será recurrente a lo largo de este trabajo, visto que afecta directamente en la cantidad de qubits máxima que puede ser simulada, debido a las limitaciones de memoria RAM (ver sección 3.1).

Al trabajar con un sistema de múltiples qubits puede observarse el efecto del entrelazamiento cuántico, un recurso elemental cuya relevancia en la computación cuántica se debe principalmente a sus propiedades completamente alejadas de las conocidas clásicamente. Se dice que un sistema está entrelazado cuando su estado cuántico no es separable en producto de sus estados monoparticulares. Por ejemplo, si el estado de n partículas $|\psi\rangle \in \mathcal{H}$ no puede expresarse de la forma $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \dots \otimes |\psi_n\rangle$, siendo $|\psi_i\rangle \in \mathcal{H}_i$ donde \mathcal{H}_i es el espacio de Hilbert asociado al qubit q_i , entonces $|\psi\rangle$ es un estado entrelazado. Actualmente se ha llegado a la conclusión de que el entrelazamiento cuántico es condición necesaria para obtener la ventaja cuántica partiendo de estados puros [19], pero no suficiente [20].

2.2 Modelo Cuántico de Computación: Circuitos

Los dos modelos principales de computación son la máquina de Turing y el modelo de circuitos [18], inicialmente originados desde un punto de vista clásico hasta desarrollar posteriormente sus análogos cuánticos. En la práctica, es muy útil el uso del modelo de circuitos cuánticos en el estudio de la computación cuántica y será desarrollado en esta sección.

Los ordenadores reales tienen dimensión finita, mientras que el modelo de Turing es un modelo idealizado en el que se supone tamaño ilimitado. Por otro lado, el modelo de circuitos es más realista, de ahí su conveniencia. De manera general, un circuito cuántico está formado por qubits de entrada, bits donde se registra la salida, puertas e hilos o cables, a través de los cuales se transmite la información.

El circuito se lee de izquierda a derecha. Los cables del circuito no tienen por que representar un cable físicamente real, sino más bien simboliza el paso del tiempo o, incluso, el movimiento de una partícula como un fotón. Por convenio, se suele tomar el estado inicial de entrada como un estado de la base, siendo el más común el estado compuesto por todo $|0\rangle$ s. Además, el conjunto de puertas que se aplican en el mismo instante temporal, es decir, que se encuentran en la misma vertical del circuito, se dice que conforman una capa. El número total de capas se mide con la profundidad d . Por ejemplo, las dos puertas de la Figura 4 forman una, dando lugar a un circuito de profundidad $d = 1$.

Formalmente, una operación sobre n qubits, realizada a través de las puertas lógicas, es una matriz unitaria en $\mathbb{C}^{2^n \times 2^n}$, es decir, de dimensión $2^n \times 2^n$. El hecho de ser unitaria es porque debe preservar la norma. Por tanto las operaciones cuánticas son lineales y reversibles. Esto último implica que, conociendo la salida de un circuito, se puede conocer la entrada aplicando el circuito inverso.

En la Figura 4 se muestra un ejemplo de un circuito cuántico genérico. Las líneas representan los qubits 1 y 2. Los qubits y las puertas cuánticas (recuadros) se componen vía producto tensorial: la matriz asociada con la puerta cuántica del ejemplo $\mathcal{R} = \mathcal{U} \otimes \mathcal{S}$ se puede calcular como el producto tensorial de las dos matrices cuánticas básicas de 2×2 , \mathcal{U} y \mathcal{S} . Este es un caso particular de nuestro ejemplo, porque la puerta no entrelaza los qubits, pudiendo separar el circuito en dos partes fácilmente. El caso más complejo, en el que los qubits se encuentren entrelazados por la puerta, se aborda en la sección 4.3.2.

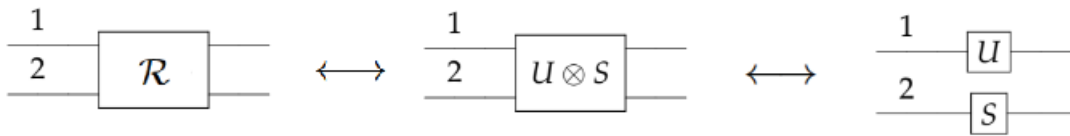


Figura 4: Ejemplo de circuito cuántico genérico.

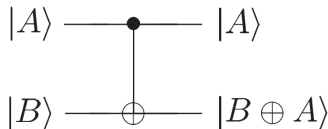
Para poder medir el estado final del sistema se aplica la operación de medición. Transforma el estado de un qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ en un bit clásico que toma valor 0 con probabilidad $|\alpha|^2$, o 1 con probabilidad $|\beta|^2$. El proceso de medida es

irreversible en la mayoría de casos, visto que la función de onda colapsa al estado medido.

2.2.1 Universalidad

Mientras que en los circuitos clásicos se trabaja con las puertas lógicas binarias, tales como AND, NAND, XOR, OR, NOR,... en los circuitos cuánticos la cantidad de puertas es mayor, visto que una puerta cuántica es cualquier operador unitario. Recordando la esfera de Bloch, para llevar un estado inicial a cualquier otro vemos que es necesario número infinito de rotaciones, equivalentemente se necesitaría un número infinito de puertas. Sin embargo, las propiedades de este conjunto al completo pueden comprenderse con un conjunto reducido de puertas, con cualquier nivel de precisión deseado. De esta manera, es posible construir una operación sobre n qubits haciendo uso de un conjunto finito de puertas cuánticas, denominado universal. Por ejemplo, en el caso clásico cualquier función sobre los bits puede calcularse con la composición de únicamente puertas NAND, por lo que esta puerta es conocida como una puerta universal.

El equivalente cuántico se sigue de este importante resultado de universalidad: cualquier puerta cuántica multi-qubit puede ser construida a partir de puertas CNOT y operaciones sobre un qubit. Aunque en este trabajo se hará uso de otro conjunto universal, es interesante introducir superficialmente esta puerta prototípica para una comprensión más completa. Una CNOT tiene dos qubits de entrada, uno de ellos es el de control y sobre el otro (target qubit) se registra la salida. Se muestra su representación en la Figura 5.



$$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figura 5: Representación de la puerta multi-qubit prototípica CNOT, junto con su representación matricial U_{CN} .

El hilo superior representa el qubit de control y el inferior el target qubit. Esta puerta modifica el estado de entrada de la siguiente manera:

$$|00\rangle \rightarrow |00\rangle; |01\rangle \rightarrow |01\rangle; |10\rangle \rightarrow |11\rangle; |11\rangle \rightarrow |10\rangle. \quad (2.5)$$

En definitiva, cuando el qubit de control toma valor 1, el target qubit rota a su otro posible valor. Si por el contrario toma valor 0, el target qubit no cambia. Estas transformaciones están descritas por las columnas de la matriz U_{CN} .

Como se ha explicado en este apartado, no existe un único conjunto universal de puertas cuánticas. En este trabajo se ha hecho uso del conjunto de puertas

empleado en el experimento de Google de 2019 [11], en el procesador cuántico Sycamore, con el que se busca demostrar la supremacía cuántica. Tanto el conjunto de puertas empleado, como la estructura de los circuitos simulados, son presentados en la sección 4.

2.2.2 Fusión de puertas cuánticas

La fusión de puertas cuánticas puede ser útil [21], [22], [23] en términos de rendimiento en los simuladores cuánticos de vector de estado, como el que es usado. Puede aplicarse tanto espacialmente como temporalmente, es decir, tanto entre puertas consecutivas en la misma vertical del circuito como entre puertas seguidas horizontalmente.

Tal y como se ha mencionado previamente, dos puertas que actúan consecutivamente en dos qubits pueden componerse mediante un producto tensorial. Además, cuando dos puertas cuánticas actúan una a continuación de la otra sobre el mismo qubit, pueden fusionarse simplemente multiplicando sus matrices asociadas. En la Figura 6 se recogen ambos casos.

$$\begin{array}{c}
 0 \times 1 \text{ --- } \boxed{F} \text{ --- } \boxed{G} \text{ --- } F \times G \\
 \begin{bmatrix} f_{1,1}g_{1,1} + f_{1,2}g_{2,1} & f_{1,1}g_{1,2} + f_{1,2}g_{2,2} \\ f_{2,1}g_{1,1} + f_{2,2}g_{2,1} & f_{2,1}g_{1,2} + f_{2,2}g_{2,2} \end{bmatrix}
 \end{array}
 \qquad
 \begin{array}{c}
 \begin{array}{c}
 0 \times 1 \text{ --- } \boxed{F} \text{ ---} \\
 0 \times 2 \text{ --- } \boxed{G} \text{ ---}
 \end{array}
 F \otimes G \\
 \begin{bmatrix} f_{1,1}g_{1,1} & f_{1,1}g_{1,2} & f_{1,2}g_{1,1} & f_{1,2}g_{1,2} \\ f_{1,1}g_{2,1} & f_{1,1}g_{2,2} & f_{1,2}g_{2,1} & f_{1,2}g_{2,2} \\ f_{2,1}g_{1,1} & f_{2,1}g_{1,2} & f_{2,2}g_{1,1} & f_{2,2}g_{1,2} \\ f_{2,1}g_{2,1} & f_{2,1}g_{2,2} & f_{2,2}g_{2,1} & f_{2,2}g_{2,2} \end{bmatrix}
 \end{array}$$

Figura 6: Fusión de dos puertas cuánticas uni-qubit actuando sobre el mismo qubit (derecha) y dos puertas operando en paralelo (izquierda) [21].

Aplicar técnicas de fusión en un circuito puede mejorar la eficiencia en cuanto a multiplicación de matrices, a expensas de reducir la paralelización disponible. Por lo tanto, es de utilidad encontrar el valor óptimo de puertas a fusionar.

2.3 Eficiencia y complejidad algorítmica

2.3.1 Notación O

Para comparar el rendimiento relativo entre algoritmos y caracterizar su eficiencia se define el orden de crecimiento del tiempo de ejecución [24]. Este orden toma sentido cuando los tamaños de entrada son lo suficientemente grandes como para que solo el orden de crecimiento del tiempo de ejecución sea relevante; estamos estudiando la eficiencia asintótica de los algoritmos.

Existen diferentes notaciones asintóticas, de las que se va a abordar la notación O, útil cuando se tiene únicamente una cota asintótica superior. Para una función

dada $g(n)$, se denota por $O(g(n))$ el conjunto de funciones que verifican

$$O(g(n)) = \{f(n) \mid \exists c, n_0 > 0 : 0 \leq f(n) \leq cg(n), \forall n \geq n_0\} \quad (2.6)$$

Es decir, la función $g(n)$ por una constante es una cota para la función $f(n)$. Se escribe $f(n) = O(g(n))$ para indicar que una función $f(n)$ pertenece al conjunto presentado. Con esta notación a menudo podemos describir el tiempo de ejecución de un algoritmo simplemente conociendo su estructura general, acotando con el peor caso posible de tiempo de ejecución.

2.3.2 Clases de complejidad

Para comprender el poder de los ordenadores cuánticos se introduce brevemente la teoría de complejidad computacional, que clasifica la dificultad de los problemas computacionales [18]. Estos son divididos en clases de complejidad acorde con los recursos computacionales necesarios para que sean resueltos.

Las clases más importantes son la clase **P** y la clase **NP**. **P** es la clase de problemas computacionales que pueden ser resueltos por una máquina de Turing determinista (ordenador clásico) en tiempo polinomial. **NP** es la clase de problemas para los cuales, si existe una solución afirmativa, esta solución puede ser verificada en tiempo polinomial. Un ejemplo muy usado para entender la diferencia entre estos conceptos es el problema de encontrar los factores primos de un entero n . Por un lado, visto que no hay una manera conocida de resolver este problema en un ordenador clásico, no se trataría de un caso **P**. Por otro, es sencillo comprobar si un número p es o no un factor de n dividiendo n entre p , por lo que sí es un caso **NP**.

Las dos clases presentadas se encuentran dentro de una clase mayor, **PSPACE**. Esta clase engloba en esencia a aquellos problemas que para ser resueltos requieren recursos de poco tamaño espacial, pero no necesariamente en tiempo. El hecho de que **NP** sea estrictamente mayor que **P** y que **PSPACE** sea estrictamente mayor que estos últimos no ha sido probado aún, sin embargo.

Respecto a las clases de complejidad cuántica, se tiene la clase **BQP**, que engloba a los problemas que pueden ser resueltos en tiempo polinomial, en un ordenador cuántico con un error acotado por $1/3$ en todas las instancias. No se conoce la relación concreta entre esta clase y las clases clásicas anteriores, pero si es sabido que los ordenadores cuánticos pueden resolver todos los problemas tipo **P** eficientemente, pero que no hay problemas fuera de **PSPACE** que puedan resolver de forma eficiente. Por lo tanto, **BQP** se relaciona con **PSPACE** y **P** como se ilustra en la Figura 7.

La investigación en el campo de la computación cuántica, que busca probar la supremacía de los ordenadores cuánticos, se interesa por encontrar los problemas que pudieran pertenecer a **BQP** sin estar contenidos **P**. La existencia de estos problemas demostraría que **P** no es equivalente al **PSPACE**. En este trabajo se aborda un problema candidato a poder demostrarlo.

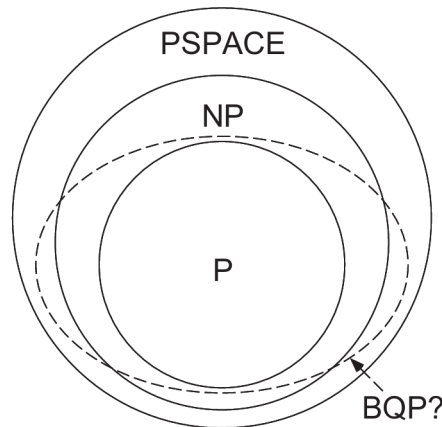


Figura 7: Relación entre las clases clásicas y cuánticas [18].

3 Computación de alto rendimiento

La computación de alto rendimiento o HPC (High Performance Computing) es la rama de la informática que se dedica al desarrollo de infraestructuras con gran potencia de procesamiento para resolver problemas complejos [25]. Los supercomputadores consisten en redes de computadores coordinados para resolver problemas que no son asequibles en un único ordenador. Las principales diferencias entre un sistema HPC y otros clusters de computadores -conjunto de ordenadores o nodos que trabajan conjuntamente- están en el número total de nodos, la potencia individual de cada nodo y la interconexión entre ellos.

Las simulaciones y las comparaciones de rendimiento que se presentan en la sección 5 se han realizado en uno de los nodos del servicio de HPC Proteus, del Instituto Carlos I de Física Teórica y Computacional. Concretamente, se ha trabajado en el nodo Orion, de la serie EPYC 9004 de AMD. Además, este nodo cuenta con una tarjeta gráfica o GPU Nvidia A100, también empleada en el estudio.

Para hacer estudios de rendimiento, es de gran utilidad conocer la estructura y funcionamiento de los sistemas empleados.

3.1 Unidad central de procesamiento

La Unidad Central de Procesamiento o CPU es un componente electrónico en forma de chip, instalado en la placa base a través de un socket [26]. Es un elemento esencial, que realiza todos los cálculos aritméticos lógicos necesarios para los programas y el sistema operativo. Para comprender su funcionamiento se va a hablar de su arquitectura, es decir, de su estructura interna. No se trata de una descripción de tamaño o forma, sino de la localización y organización de las distintas unidades lógicas y físicas.

- **Núcleos o cores:** Los núcleos son las entidades de procesamiento de información. Cada core puede trabajar en una tarea, de manera que cuantos más cores haya más eficiente será el procesador. En un pasado, los procesadores tenían un único núcleo. En nuestro caso, el nodo cuenta con 96, divididos en

dos grupos de 48 núcleos siguiendo la estructura mostrada en la Figura 8. Los cores corresponden con las casillas denominadas Z4, divididos en conjuntos de 6 cores conocidos como dados (dies), en los que comparten una memoria caché L3. Cada uno de los núcleos tiene una memoria caché L2 propia, además de una L1 [26].



Figura 8: Detalle de una de las CPUs del procesador, correspondiente a uno de los sockets, de 48 cores.

- **Memoria caché:** Este elemento es de una gran importancia, ya que en las memorias caché se almacenan los datos que van a ser utilizados inmediatamente por el procesador. La importancia de esta cercanía a los cores reside en que se reduce la espera, conocida como tiempo de latencia, que requiere acceder a la memoria RAM, explicada también en esta sección. Existen tres niveles de memoria caché, siendo L1, L2 y L3, en orden de cercanía a los núcleos. Entre más cercana sea la memoria, más pequeña y más rápida es -lo que equivale a un mayor ancho de banda-, siendo todas mucho más rápidas que la RAM.
- **IOD:** Las siglas IOD significan Input/Output Die, refiriéndose al dado que contiene las comunicaciones entre el procesador y el exterior. Aquí el ancho de banda también es importante, ya que a mayor cantidad de cores mayor ancho de banda se requiere.
- **Memoria RAM:** La Random Access Memory almacena temporalmente la información del proceso que se esté realizando en ese momento. A pesar de no

estar al nivel de las memorias caché, es significativamente rápida en comparación con otros medios de almacenamiento de acceso directo, como los discos duros. Se denomina de acceso aleatorio debido a que no se requiere un acceso ordenado a los datos almacenados. El nodo Orion cuenta con 1536GB de RAM.

Es posible paralelizar una tarea haciendo uso de varios núcleos. La paralelización es posible a través de las hebras o hilos de procesamiento. Estas hebras dividen las tareas, si es posible, en varios cálculos más simples, de manera que cada uno de estos subprocesos puede ser ejecutado en un núcleo.

Como se ha mencionado, el nodo está compuesto por dos CPUs. Esto se denomina configuración de dos sockets, dando lugar a un multiprocesador. En la Figura 9 se esquematiza esta configuración. No se entra en detalles de cada uno de los elementos dado que no es requisito para la comprensión de este trabajo. De modo general, las dos CPUs están conectadas por links específicos a cada componente del socket. En el esquema se identifican los cores Z4 en las esquinas, además del I/O Die en el centro. Además, se muestra la distribución de los 12 controladores de memoria D5.

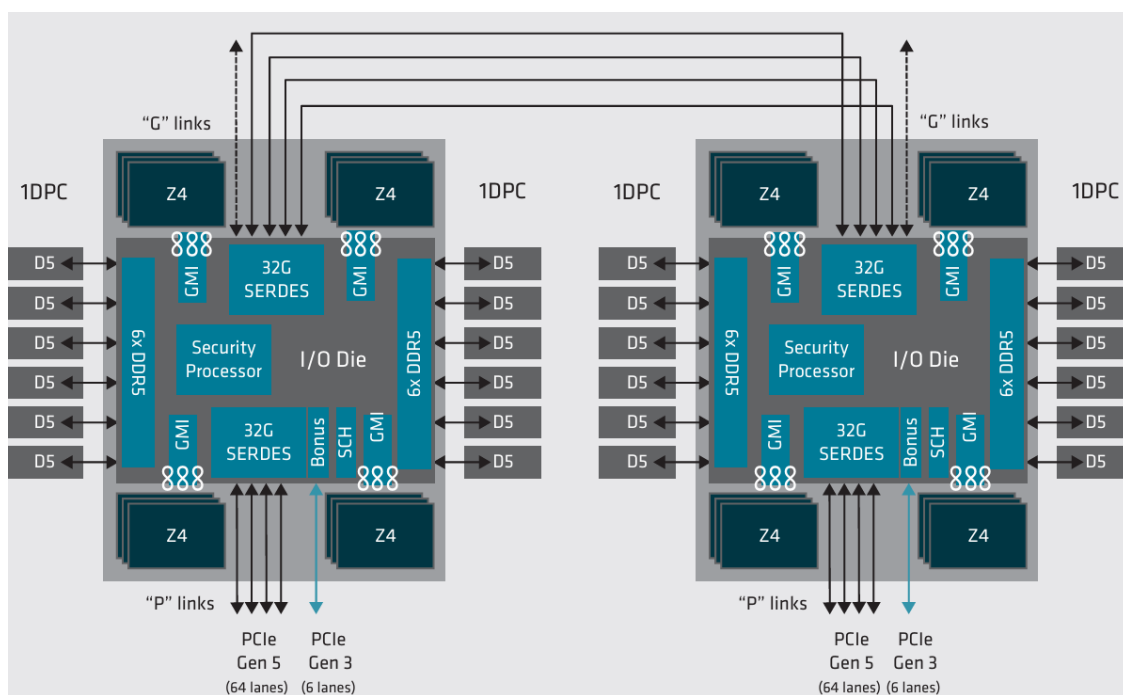


Figura 9: Arquitectura de un procesador AMD EPYC 9004 Series en una configuración de dos sockets [26].

Por último, en este tipo de arquitectura multi-chip debe considerarse la existencia de variaciones en el tiempo de latencia de acceso a la RAM, que depende de las conexiones de los controladores de memoria y la CPU. Esto se denomina NUMA o acceso no uniforme a la memoria.

3.2 Unidad de procesamiento gráfico

Gracias a su capacidad de paralelismo masivo y su facilidad de programación, la unidad de procesamiento gráfico (GPU) se ha convertido en un motor de cómputo crucial para la computación de alto rendimiento. En 2021, 7 de los 10 supercomputadores de clase mundial están siendo impulsados por GPUs [27]. En esta sección se brinda una comprensión básica de la arquitectura de las GPU.

El modelo en el que se procesa y ejecuta la información en una GPU sigue una jerarquía. La función de entrada del programa se llama kernel [28]. El kernel se ejecuta en múltiples bloques de hebras o grupos de trabajo. Cada bloque de hebras consta de múltiples warps, donde el warp es un grupo de hebras (32 hebras en el caso de la GPU empleada) que ejecutan la misma instrucción simultáneamente. Las hebras dentro del mismo bloque pueden compartir datos a través de la memoria en chip. Por otro lado, los distintos bloques se ejecutan de manera independiente, lo que significa que las hebras en diferentes bloques no pueden comunicarse entre sí.

Mientras que una CPU realiza muchas tareas en diferentes núcleos, la GPU divide la tarea en subrutinas mucho más simples y en una cantidad mayor de núcleos. Dicho de otra manera, la CPU trabaja de manera independiente en diferentes tareas y la GPU trabaja de forma paralela en las mismas tareas. Por lo tanto, la GPU es más indicada para trabajos que se puedan dividir en subrutinas sencillas, como el análisis de datos. En otro caso, su eficiencia se reduce.

Una GPU contiene múltiples streaming multiprocessors (SM), cuya arquitectura para la GPU A100 se muestra en la Figura 10. Los SM contienen una serie de conjuntos de cores o núcleos que pueden ejecutar operaciones de enteros (INT32), de coma flotante de simple y doble precisión (FP32 y FP64) y tensoriales (Tensor core) en cada ciclo. Las operaciones complejas como funciones trigonométricas o raíces cuadradas se ejecutan en la unidad de funciones especiales SFU. Finalmente, Las operaciones de memoria son manejadas por las unidades de carga/almacenamiento LD/ST. El conjunto de todos estos elementos forma parte de una unidad física capaz de correr un bloque de hebras. Como se observa en la figura, la SM estudiada cuenta con cuatro de estas unidades computacionales. Esto implica que en cada SM pueden ejecutarse paralelamente cuatro bloques de hebras.

Además de las unidades computacionales, un SM consta de planificadores de warps, unidades de asignación y archivos de registro. Como se ha mencionado, las instrucciones se ejecutan por warps. La GPU sigue una ejecución ordenada, lo que puede dar lugar a detenciones en la ejecución de los warps si surge algún conflicto, hasta que sea solucionado. El planificador de warps selecciona los warps que estén listos en cada ciclo, ocultando el tiempo de latencia por las demoras. Cada planificador de warps está asociado a una unidad de asignación y cada una puede emitir un warp listo. Estos warps son entonces asignados a una de las unidades presentadas, como la INT32 o la SFU. Por último, los archivos de registro son un almacenamiento temporal para datos durante la ejecución de una instrucción. Mientras que en una CPU estos registros son pequeños, en la GPU

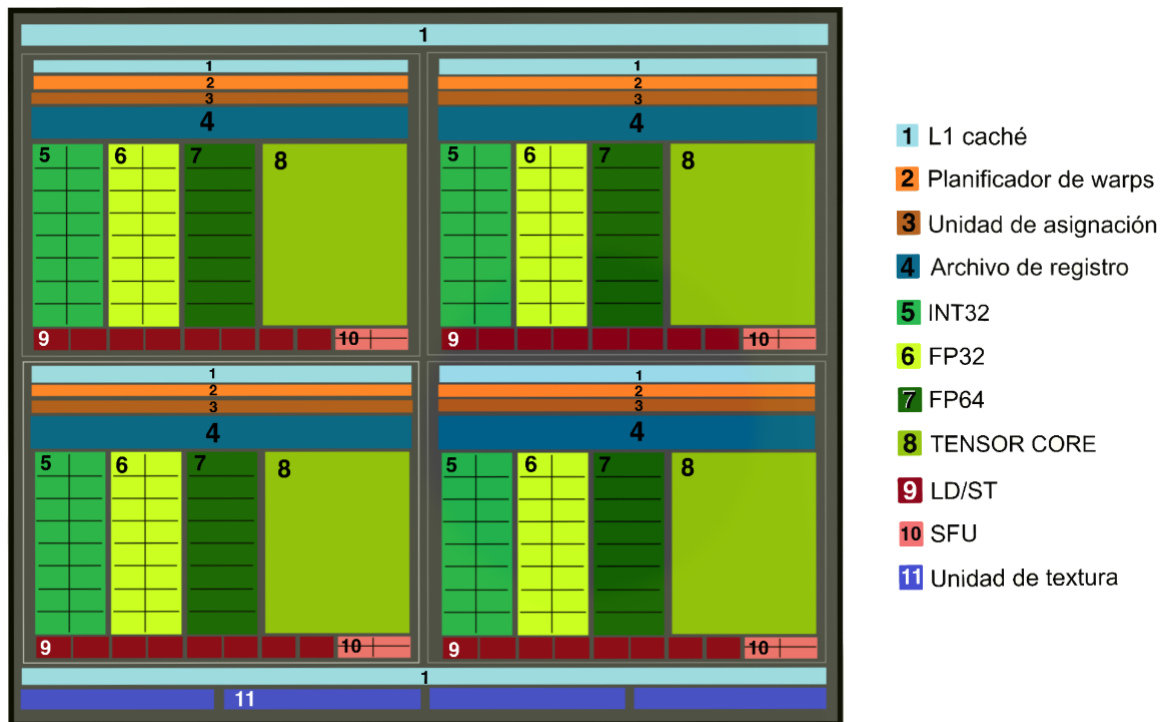


Figura 10: Arquitectura de un streaming multiprocessor de una GPU Nvidia A100.

son mayores para poder almacenar la información de decenas de warps.

Los archivos de registro son la memoria más pequeña y rápida, encabezando una jerarquía de memorias de mayor complejidad que en una CPU [29]. A continuación se da un esbozo de cada una de estas memorias en orden ascendente en tamaño (descendente en rapidez).

- **Memoria compartida:** Es el espacio compartido por las hebras que realizan el mismo proceso, es decir, que pertenecen al mismo bloque. Facilita un intercambio rápido de información entre los cores si las comunicaciones son de tamaño reducido.
- **Memorias caché:** Las GPUs suelen utilizar una jerarquía de caché de dos niveles. Cada SM cuenta con una caché L1. Además, hay dos cachés L2, cada una compartida por 64 SMs a través de una red de interconexiones, como se observa en la Figura 11. Esto hace que la latencia de acceso a la L2 sea mucho mayor que a la caché L1.
- **Memoria global:** Es la memoria principal de la GPU. Es la región de lectura y escritura de mayor tamaño. Es accesible por la CPU para enviar datos de entrada y recibir datos de salida.
- **Memoria RAM:** Tiene la misma función que en una CPU. La GPU Nvidia A100 utilizada cuenta con 80GB de memoria, un espacio mucho menor comparado con el de la CPU. Esto limita el número de qubits a simular, reduciéndolo a 33 qubits como máximo.

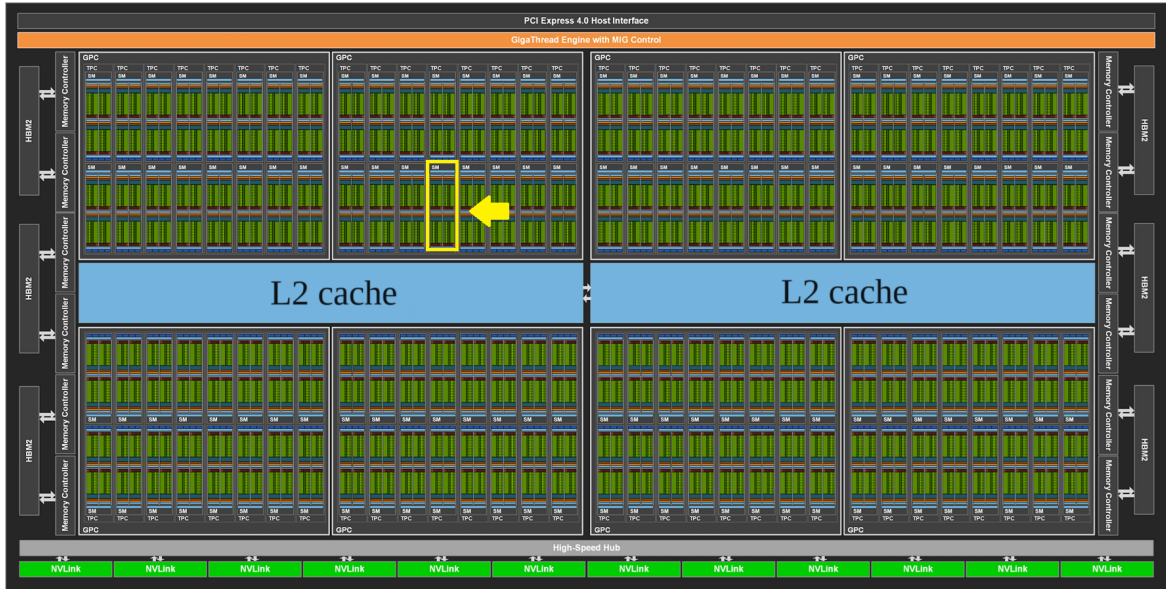


Figura 11: Arquitectura de una GPU Nvidia A100. La zona señalizada con una flecha amarilla corresponde con un SM [30].

El acceso no uniforme a la memoria o NUMA, mencionado para la CPU, toma mayor importancia en los tiempos de latencia si además se incorpora una GPU al sistema. La GPU se conecta a una de las CPUs que componen el nodo de trabajo. Esto implica que una de las CPUs tendrá un acceso más directo que la otra, creando una asimetría. Si la información es enviada desde la CPU que no está conectada directamente, dicha información primero debe ser transmitida a la otra CPU, resultando en un aumento del tiempo de latencia.

Tipos de tiempo de simulación

Al lanzar un trabajo a una CPU o GPU se diferencian distintos tipos de tiempo, en función de su origen:

- **Tiempo de cómputo:** Este tiempo es el que el intérprete dedica a la ejecución de funciones y operaciones, por ejemplo, el tiempo empleado en ejecutar librerías como numpy, scipy o qsim, o multiplicar matrices. Se puede ver afectado por la división excesiva en hebras.
- **Tiempo de sistema:** Es el tiempo que se invierte en el sistema operativo en hacer llamadas al sistema y operaciones de entrada y salida, como leer o escribir en el disco. Cuando hay un número excesivo de hebras pueden darse luchas por los recursos, dando lugar a cuellos de botella en el dado I/O.

3.3 Factores que afectan al rendimiento

Cuando se trabaja con sistemas físicos, como una computadora, el rendimiento teórico esperado no coincide con el observado. Tanto la CPU como la GPU se ven expuestas a condiciones físicas que reducen dicho rendimiento. Es una labor

complicada la de controlar todas las fuentes que producen estas variaciones, que difieren, además, de simulación en simulación.

Una de las fuentes es el NUMA. Al haber un acceso asimétrico, no todas las simulaciones seguirán el mismo patrón de acceso a la memoria. Esto se ve acentuado al trabajar con la GPU, visto que una de las CPUs tiene un acceso más directo a ella. Además, usar la GPU implica un tiempo de latencia extra por la carga de datos, ya que la información debe recorrer un camino mayor al que recorre cuando solo se trabaja con la CPU. Por otro lado, también se puede ralentizar el procesamiento de los trabajos si la temperatura del nodo es alta. Como el nodo es accesible para diferentes usuarios, no siempre estará a la misma temperatura porque su uso es irregular y esto se refleja en una variación relativa del rendimiento. Estas variaciones de temperatura son elevadas haciendo necesaria la instalación de máquinas de refrigeración. A lo largo del trabajo el nodo se apagó en dos ocasiones, una debida al sobrecalentamiento por la parada de los aires acondicionados.

En resumen, la tarea de conocer y controlar todas las posibles fuentes de estas variaciones es muy compleja. Las simulaciones realizadas pueden requerir ser repetidas en los casos en los que el rendimiento fuera excesivamente bajo, ya que no es tan importante la obtención del cómputo de la simulación como el estudio del tiempo que requiere.

4 Problema práctico

4.1 Objetivo

El objetivo de este trabajo es realizar simulaciones de alto rendimiento de circuitos cuánticos. Con esto lo que se pretende es ver cuáles son las limitaciones de simular clásicamente estos circuitos y optimizar los parámetros implicados. Se han estudiado diferentes estrategias para simular estos circuitos a través de herramientas HPC. Para ello, se han comparado distintas arquitecturas hardware con diferentes elementos, lanzando trabajos a un nodo en modo single-core o multi-core, además de a una GPU.

En resumen, en este trabajo se estudiará la optimización de los parámetros de los simuladores para que se adecúen a la arquitectura concreta del hardware empleado. A continuación se presentan los circuitos de interés, las diferentes estrategias seguidas para simularlos y los parámetros involucrados en la optimización.

4.2 Descripción del circuito simulado

Los circuitos simulados en este trabajo son circuitos cuánticos aleatorios, los cuales han sido ejecutados en el chip cuántico Sycamore para demostrar la superioridad cuántica [31]. Cada circuito aleatorio está compuesto de m ciclos. Cada ciclo consiste en una o dos capas. La primera capa consta de puertas 1-qubit² y la segunda capa está formada por puertas que actúan de dos en dos qubits consecuti-

²Puertas que actúan sobre un qubit. A partir de ahora se usará la notación puerta n-qubit para referirnos a una puerta que actúa sobre n qubits.

vos.

El algoritmo que genera los circuitos aleatorios sigue un patrón que se repite cada 8 ciclos A, B, C, D, C, D, A, B, del que depende el número de capas por ciclo. Este patrón especifica cómo actúan las puertas 2-qubits, mientras que las puertas 1-qubit son escogidas uniformemente de manera aleatoria, excluyendo la puerta que se haya aplicado en el ciclo previo.. La estructura que tendría un circuito de 8 ciclos se muestra en la Figura 12. Finalmente, antes de realizar una medida, se añade una capa extra de puertas sobre un qubit. Las capas de puertas 2-qubits se construyen de manera intercalada: si en el ciclo m hay una capa de puertas 2-qubits que se aplica empezando sobre los qubits n_0 y n_1 , en el ciclo $m + 1$, si también hay una capa de puertas 2-qubits, se aplican empezando sobre los qubits n_1 y n_2 .

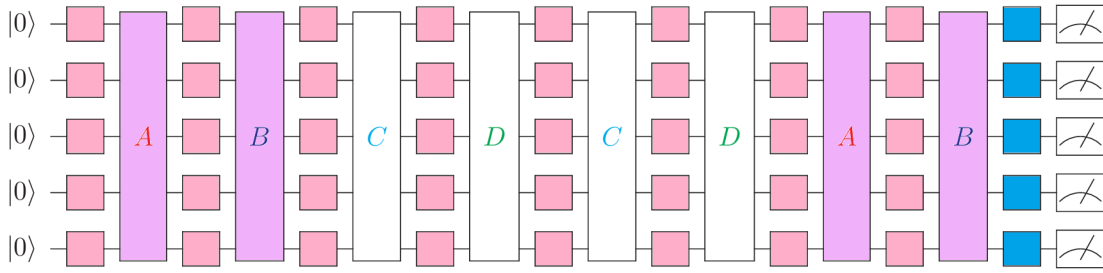


Figura 12: Esquema de un circuito de 8 ciclos [12], simulado en este trabajo. Cada ciclo contiene una capa de puertas 1-qubit y una segunda capa de puertas 2-qubits (etiquetadas como A, B, C, D). Las puertas 2-qubits coloreadas (A y B) están 'activadas' en nuestro circuito, mientras que las puertas blancas no aparecen, dando lugar a ciclos de una sola capa. Los cubos rosas son las puertas 1-qubit mencionadas. Los cubos azules representan la capa final de puertas 1-qubit.

En la capa de puertas 1-qubit se escoge entre las puertas del conjunto $\{X^{1/2}, Y^{1/2}, W^{1/2}\}$, donde

$$X^{1/2} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}, \quad Y^{1/2} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \quad W^{1/2} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -\sqrt{i} \\ \sqrt{i} & 1 \end{pmatrix} \quad (4.1)$$

Las puertas 2-qubits se descomponen en torno a cuatro rotaciones en torno a Z determinadas por el índice del ciclo y

$$\mathbf{fSim}(\theta, \phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -i \sin \theta & 0 \\ 0 & -i \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & e^{-i\phi} \end{pmatrix}, \quad (4.2)$$

donde los parámetros θ y ϕ están determinados por el par de qubits sobre los que se actúa. La descomposición de Schmidt (ver sección 4.3.2) de la mayoría de las puertas 2-qubits de estos circuitos aleatorios consta de cuatro términos, por lo que son difíciles de simular con cualquier método conocido y generan mucho en-

trelazamiento. Todas las puertas presentadas en esta sección forman un conjunto universal (demostrado en [31]).

4.3 Métodos de simulación

Para el desarrollo de este trabajo se ha hecho uso de las librerías Cirq [32] y Qsim [33], necesarias para la simulación de los circuitos cuánticos. Cirq es empleada para escribir, manipular y optimizar circuitos cuánticos. Qsim permite emplear diferentes métodos para simular los circuitos generados con Cirq.

4.3.1 Qsim. Algoritmo de Schrödinger

Qsim es un simulador de estado completo de Schrödinger. Calcula todas las 2^n amplitudes del vector de estado, donde n es el número de qubits. La evolución de la función de onda bajo la ecuación de Schrödinger puede interpretarse como la interferencia entre las amplitudes de probabilidad de los estados posibles. Esencialmente, el simulador realiza productos de matriz por vector repetidamente. Un producto de matriz por vector corresponde a la aplicación de una puerta. Si se aplica el conjunto de las puertas en cada ciclo U_m en orden cronológico, deben multiplicarse de derecha a izquierda sobre el estado de entrada:

$$U_m U_{m-1} \dots U_1 |\psi\rangle \quad (4.3)$$

Para calcular este producto matricial se deben representar las puertas en la dimensión total del sistema. Si se tiene la puerta T_2 aplicada sobre el qubit 2, se tiene que su representación matricial es $I_1 \otimes T_2 \otimes I_3 \otimes \dots \otimes I_n$. Si se tuviera, además, una puerta T_4 , entonces la representación es $I_1 \otimes T_2 \otimes I_3 \otimes T_4 \otimes \dots \otimes I_n$.

El tiempo total de ejecución es

$$O(g2^n) = O(mn2^n), \quad (4.4)$$

donde $g = mn$ es proporcional al número de puertas 2-qubits. Para acelerar el simulador, puede utilizarse la fusión de puertas. La computadora clásica almacena la función de onda completa y la propaga bajo la evolución inducida por el circuito cuántico. En términos de memoria, esto implica que se guarda una información de tamaño $O(2^n)$, ya que el vector de estado tiene dicha dimensión. Esto limita el número de qubits que puedan simularse. Los requerimientos de memoria en gigabytes (GB) pueden calcularse con la siguiente regla general:

$$\text{Memoria requerida (GB)} : 2^n \times 8 \times \frac{1 \text{ kB}}{1024 \text{ bytes}} \times \frac{1 \text{ MB}}{1024 \text{ kB}} \times \frac{1 \text{ GB}}{1024 \text{ MB}} \quad (4.5)$$

Por lo tanto en el caso del ordenador empleado, cuya memoria RAM tiene un tamaño de 1536 GB, como máximo pueden simularse circuitos de 37 qubits, equivalente a 1024 GB, ya que 38 qubits (equivalente a 2048 GB) supera el espacio permitido por los recursos.

La función que calcula la evolución del sistema mediante el algoritmo de Schrödinger depende de los siguientes parámetros:

- **Hebras:** Indica el número de divisiones, en subtareas, del cálculo de la evolución del vector de estado. Envía las subtareas a núcleos diferentes para acelerar el proceso, aunque procesar las subtareas también requiere un coste, por lo que puede buscarse el valor óptimo. El valor máximo en este caso es 96, es decir, el número máximo de núcleos que pueden ser usados en las CPUs. La GPU regula automáticamente los núcleos que va a utilizar, sin posibilidad de alterarlo.
- **Fusión de puertas:** Señala el número de puertas máximo a fusionar en el circuito, según el procedimiento explicado en la sección 2.2.2. El estudio de este parámetro permite adecuar el tamaño de estas nuevas matrices fusionadas al tamaño de las memorias caché, acelerando la simulación.

4.3.2 Qsimh. Algoritmo híbrido Schrödinger-Feynman

Qsimh es un simulador híbrido Schrödinger-Feynman³ [34]. A continuación se presenta brevemente el método de Feynman.

Integral de camino de Feynman

Este método se basa en las integrales de camino de mismo nombre. La base de estas integrales es la siguiente: en un espacio de Hilbert n -dimensional, la amplitud de probabilidad de pasar de un estado de la base a otro se obtiene sumando las amplitudes de probabilidad de todos los caminos posibles para llegar a ese estado:

$$\begin{aligned} \langle y | U_{p(n)} \cdots U_2 U_1 | 0 \rangle &= \sum_{x_1, \dots, x_{p(n)-1}} \langle y | U_{p(n)} | x_{p(n)-1} \rangle \\ &\quad \langle x_{p(n)-1} | U_{p(n)-1} \cdots U_2 | x_1 \rangle \langle x_1 | U_1 | 0 \rangle. \end{aligned} \quad (4.6)$$

En términos de complejidad computacional, la representación de Schrödinger lleva a una simulación en espacio exponencial ya que es necesario calcular toda la evolución de los vectores de estado. Por otro lado, la integral de camino de Feynman, que suma todos los caminos, conduce a una simulación en espacio polinómico.

De esta manera, un camino de Feynman toma un estado de entrada $|x_{n-1}, x_{n-2}, \dots, x_0\rangle$ hasta un estado de salida específico $|y_{n-1}, y_{n-2}, \dots, y_0\rangle$ a través de un conjunto de estados intermedios $|z_{n-1}, z_{n-2}, \dots, z_0\rangle$. Traducido al tratamiento del circuito, lo que se hace es ir dividiendo sucesivamente el circuito por la mitad, y estas mitades se vuelven a dividir, hasta llegar a un qubit por subcircuito. Esto lleva al cálculo de una secuencia de puertas 1-qubit aplicadas en $|x\rangle$, que se obtiene fácilmente computando gn valores binarios, en otras palabras, una integral de camino. Gracias a este método, cuando se simula la evolución del vector de estado

³A partir de ahora se alterna entre la denominación SF y Schrödinger-Feynman.

de un circuito con entrada $|x_{n-1}, x_{n-2}, \dots, x_0\rangle$, es posible calcular la integral de camino para un estado de salida esperado $|y_{n-1}, y_{n-2}, \dots, y_0\rangle$ sin la necesidad de calcular todas las posibles amplitudes de la superposición de estados final.

El algoritmo de Feynman tiene un coste de tiempo de $O(4^g)$, mayor que el de Schrödinger, pero un coste en memoria mucho inferior, de $O(g + n)$. Al tratarse de una suma de coeficientes, es altamente paralelizable en una GPU. Este método se suele emplear de manera híbrida con el de Schrödinger [35].

Método híbrido

Usando el método híbrido, el circuito se divide únicamente en dos partes, con n_1 y n_2 qubits respectivamente. Cada subcircuito evoluciona con el algoritmo de Schrödinger. Para descomponer las puertas 2-qubits que se encuentran en el corte del circuito, se utiliza la descomposición de Schmidt, de manera que cada subcircuito pueda ser simulado de manera independiente y los resultados puedan sumarse al final. Por ejemplo, la descomposición de Schmidt de una puerta CZ es la siguiente:

$$\begin{aligned} CZ = \text{diag}(1, 1, 1, -1) &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\ &= P_0 \otimes I + P_1 \otimes Z. \end{aligned} \quad (4.7)$$

Si el rango de Schmidt de cada puerta es s y el número de puertas en el corte es k , entonces hay s^k caminos. El tiempo total de ejecución es $O(2^{n_1} + 2^{n_2})s^k$, y la memoria requerida pasa de $O(2^n)$ a $O(2^{n/2+1})$, ya que se reduce a la mitad el tamaño del sistema. También se puede aproximar

$$O(2^{n_1} + 2^{n_2})s^k \approx O(2 \times 2^{n/2})4^{m/4}, \quad (4.8)$$

ya que se corta por la mitad y, entonces, $2^{n_1} + 2^{n_2} \approx 2 \times 2^{n/2}$. Se ha tomado $s = 4$ y $k = m/4$. Esto último es porque de los m ciclos, solo la mitad cuentan con capas de puertas 2-qubits, y entre ellos solo la mitad tienen una puerta que se encuentre en el corte debido a su estructura intercalada.

5 Resultados y discusión

Todos los resultados presentados en esta sección han sido realizados por la autora. Los códigos utilizados se encuentran en este [36] repositorio de github. Están brevemente explicados, complementando a lo expuesto en la sección 4⁴. Para la obtención de los tiempos de simulación y los requisitos de memoria se ha empleado Scalene [37], un *profiler* o perfilador para Python orientado a la computación de alto rendimiento en CPU y GPU. Se explica su uso en el apéndice A.

Las incertidumbres se han obtenido siguiendo la Desigualdad de Hoeffding, la cual permite la estimación del tamaño de muestra necesaria para que la media

⁴Si alguien quisiera reproducir y comprobar los resultados y no pudiera, puede contactarme a través de noesg@correo.ugr.es

experimental no se desvíe más de un ϵ del valor real, con un nivel de confianza α . Para más detalles, consultar el apéndice B.

5.1 Algoritmo de Schrödinger

5.1.1 Requisitos de memoria RAM

Antes de comenzar con las simulaciones para optimizar la eficiencia del método de Schrödinger, es necesario conocer cuál es la limitación en el número de qubits del circuito. Para ello, se ha obtenido el perfil de memoria, enviando los trabajos únicamente a las CPUs del nodo (ver Fig. 13). Debido a que el profiler no detecta tamaños inferiores a 10MB, se han obtenido los requisitos de memoria a partir de 21 qubits. Mediante la ecuación 4.5, se puede calcular que el valor teórico de memoria para 20 qubits es 8MB, mientras que para 21 qubits es 16MB. El mismo procedimiento podría haberse seguido enviando las simulaciones a la GPU. Sin embargo, se habrían obtenido iguales resultados hasta 33 qubits, ya que los requisitos de memoria en función del número de qubits son independientes de la máquina empleada.

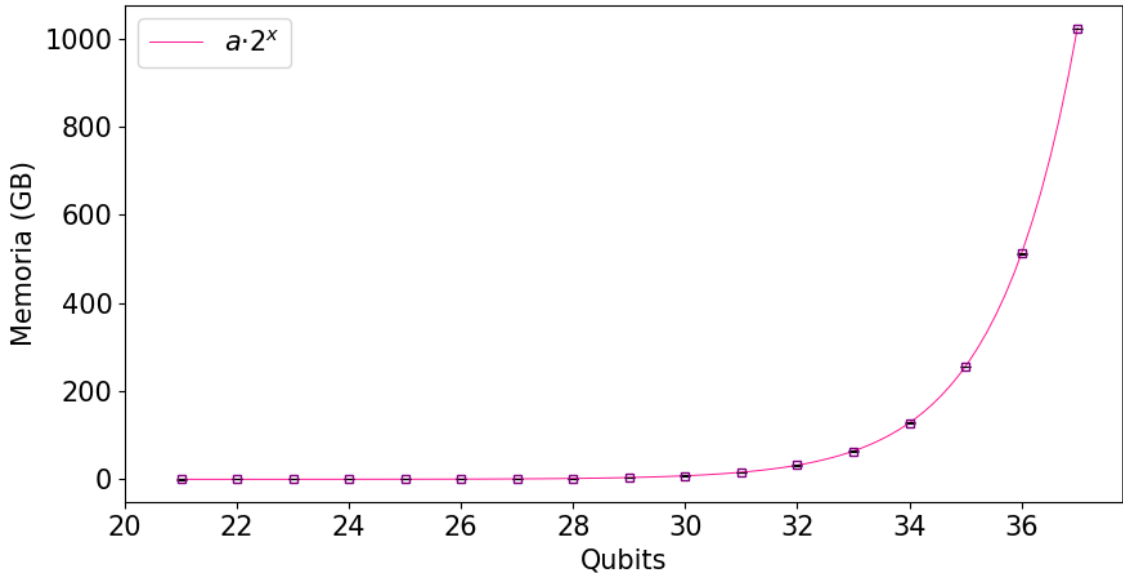


Figura 13: Ajuste según la función $f(x) = a \cdot 2^x$ de la memoria requerida en la simulación de n qubits mediante el método de Schrödinger. Las barras de error son despreciables respecto a los valores medidos.

Los datos se han ajustado según la función $f(x) = a \cdot 2^x$, donde si se toma

$$a = 8 \times \frac{1 \text{ kB}}{1024 \text{ bytes}} \times \frac{1 \text{ MB}}{1024 \text{ kB}} \times \frac{1 \text{ GB}}{1024 \text{ MB}}$$

se ve que la función de ajuste corresponde con la ecuación 4.5. Los datos siguen por tanto los valores teóricos esperados, con una tendencia exponencial. Con este resultado se confirma lo que ya se conocía: un sistema de n qubits es un producto

tensorial entre los espacios de Hilbert de cada qubit, $\mathbb{C}_1^2 \otimes \mathbb{C}_2^2 \otimes \dots \otimes \mathbb{C}_n^2 = \mathbb{C}^{2^n}$, de manera que la dimensión de la base del espacio resultante aumenta según $O(2^n)$. Esto se traduce en que el tamaño del vector de estado de la función de onda seguirá dicha tendencia.

El crecimiento exponencial con el tamaño del sistema se conoce como maldición de la dimensión, siendo un problema presente en física y otras ramas [38]. Es un obstáculo que limita enormemente la capacidad de simular circuitos cuánticos clásicamente, ya que una computadora no tiene memoria RAM ilimitada. Recordando las características de los sistemas empleados, un nodo con RAM de 1536GB y una GPU con RAM de 80GB, y atendiendo a la Fig. 13, se ve que los límites del sistema de n qubits son de 37 qubits para el nodo y de 33 qubits para la GPU.

5.1.2 Eficiencia de la paralelización en hebras

En la sección 3.1 se ha señalado la importancia de la paralelización en sistemas HPC. Las CPUs empleadas cuentan con 96 cores, por lo que un trabajo puede dividirse hasta en 96 hebras o subtareas. Para comparar la mejora en los tiempos, se ha realizado la misma simulación para sistemas de 20, 25, 26 y 27 qubits variando el número de hebras en potencias de dos hasta llegar al total de 96 (ver Figura 14). En todos los casos representados, excepto para 20 qubits, se cumple que el tiempo de simulación mejora conforme aumenta el número de hebras, incluso en la región ampliada.

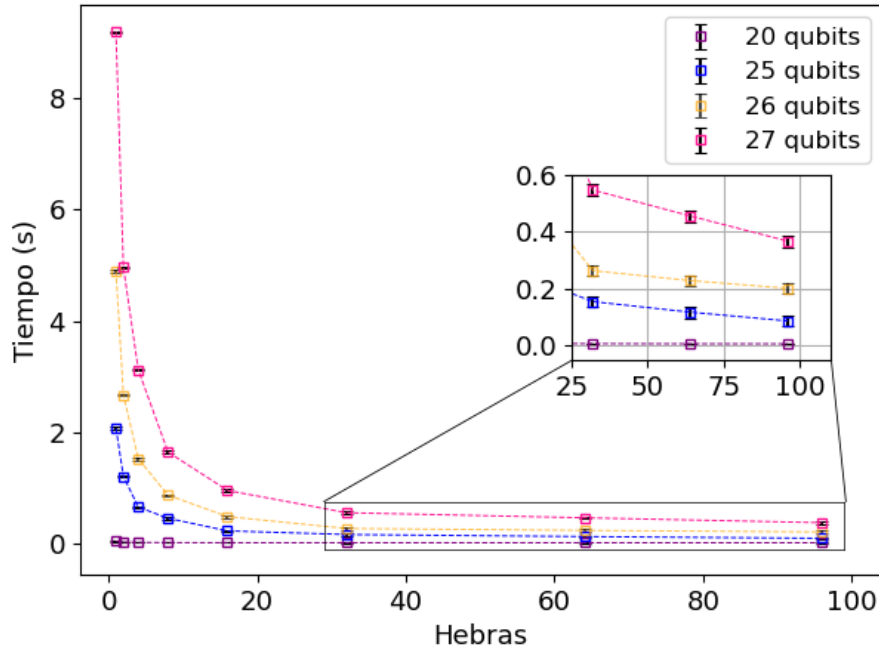


Figura 14: Tiempo obtenido para cada número de hebras, en sistemas de 20, 25, 26 y 27 qubits. La profundidad de los circuitos es $m = 0.7n$.

El sistema de 27 qubits muestra un mayor tiempo de simulación debido a su

mayor tamaño, que supone una mayor complejidad de cálculo. Para los tres sistemas el tiempo disminuye rápidamente, hasta que comienza a estancarse en torno a las 20 o 30 hebras. Para encontrar el valor óptimo de hebras, es decir, el número de hebras tras el cual el tiempo de simulación comienza a estabilizarse, se han representado los datos en la Figura 15, en la que los ejes están en escala logarítmica de base 2.

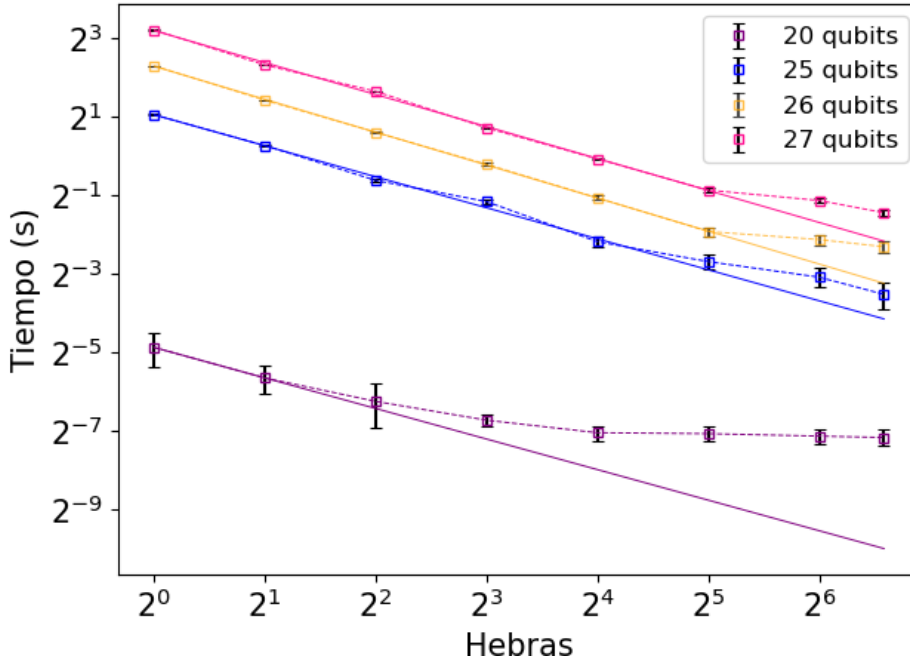


Figura 15: Tiempo obtenido para cada número de hebras. Los ejes están en escala logarítmica de base 2. Las líneas continuas son resultantes de ajustar los seis primeros puntos de las series de 26 y 27 qubits, los cinco primeros de la serie de 25 qubits, y los dos primeros de la serie de 20 qubits, según la escala logarítmica.

Vamos a definir el valor óptimo de hebras como a partir del cual deja de haber una tendencia exponencial. Es óptimo en el sentido de eficiencia energética de los recursos y para el caso de querer simular varios circuitos a la vez. Por ejemplo, es más eficiente simular tres circuitos de 27 qubits con 32 hebras cada uno a la vez, que simular esos tres mismos circuitos con 96 hebras por separado.

Para 25, 26 y 27, se observa una disminución lineal en el intervalo de los cinco o seis primeros puntos, hasta alcanzar las $2^5 = 32$ hebras el valor óptimo para 26 y 27 qubits, o $2^4 = 16$ para 25 qubits. A partir de ahí, los valores se desvían hacia arriba. Como en la escala utilizada una recta implica una relación exponencial entre los ejes, la desviación de los valores se traduce en una reducción en la mejora del tiempo en función del número de hebras. Una mayor diferencia se observa para 20 qubits, donde a partir de 4 hebras deja de haber una mejora exponencial. Este resultado coincide con lo indicado por los desarrolladores de qsim en [39], que afirman que el número de hebras de CPU no afecta al rendimiento para circuitos pequeños (de menos de 17 qubits).

Los cambios en la eficiencia de la paralelización del trabajo son debidos a varios factores. Por un lado, dividir el trabajo en subtarear supone un coste adicional, a parte del tiempo de cómputo de la simulación. Además, un valor alto de hebras puede derivar en una lucha por los recursos compartidos, dejando a algunas hebras en estado de espera; como se explica en la sección 3.1 los núcleos comparten, entre otros elementos, las memorias L3 o el IOD. Por otro lado, la división excesiva de las operaciones numéricas también supone una ralentización en la eficiencia de cálculo. En resumen, a partir de cierto número de hebras, la diferencia entre el tiempo ganado por la paralelización y el perdido por la gestión de las hebras y los cálculos va reduciéndose.

Con el objetivo final de comparar el nodo del servicio HPC con la GPU, se tomaran 96 hebras en el nodo. Aunque se haya visto que los valores óptimos están muy por debajo, se sigue observando una ligera mejora en los tiempos y, para hacer una comparación justa, se usará el nodo a su máxima capacidad. En otro caso, en el que no se busque comparar, sino hacer los cálculos de la simulación usando eficientemente los recursos, lo correcto es aplicar los valores optimizados.

5.1.3 Optimización del parámetro de fusión de puertas

A continuación, se realiza un análisis respecto al parámetro que controla el número de puertas fusionadas. En la Figura 16 se grafica la comparación con 20 qubits, para la CPU y la GPU, del tiempo para cada caso. Para este tamaño de sistema,

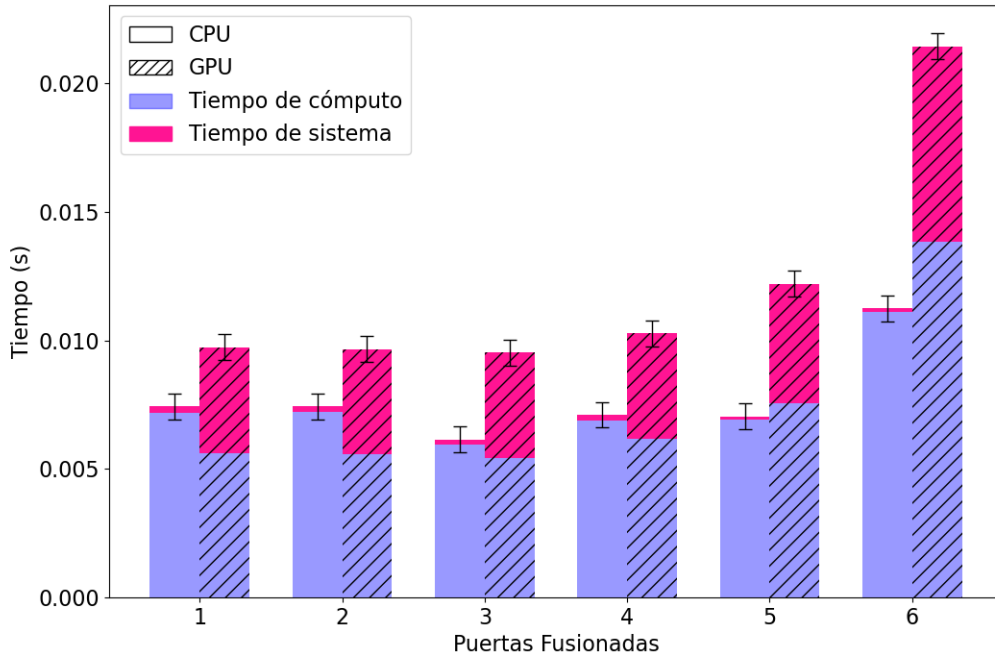


Figura 16: Comparación de los tiempos para $n = 20$ qubits en CPU y GPU, variando el número de puertas fusionadas. Las barras de error corresponden a la suma de tiempo total. La profundidad de los circuitos es $m = 0.7n$.

la GPU muestra mayores tiempos para actividades del sistema, no relacionadas con el cómputo de la evolución de la función de onda. Esto es debido a que la información debe recorrer una gran distancia para ir desde la CPU que recibe el trabajo hasta los núcleos de la GPU.

El valor óptimo para las puertas fusionadas es 3 en CPU y GPU, aunque la diferencia con 1 y 2 puertas fusionadas es despreciable en la GPU. Con estos valores se consigue que la matriz resultante de la fusión de las puertas no exceda el tamaño de la memoria caché, evitando un exceso de entrada y salida de datos a otras memorias menos cercanas. A partir de 5 y 6 puertas, en CPU y GPU respectivamente, los tiempos aumentan significativamente debido al exceso de tamaño de las matrices resultantes y a la reducción de la paralelización disponible, que se ha invertido en la fusión. En otras palabras, aunque se fusionen las matrices en grupos mayores no se compensa la pérdida de paralelización en este proceso con la necesaria para el cómputo de las nuevas matrices.

También se ha buscado el valor óptimo para un sistema de 30 qubits. En la Figura 17 se vuelve a realizar la comparación anterior. La GPU sigue mostrando mayores tiempos para actividades del sistema, como es de esperar. Respecto a los

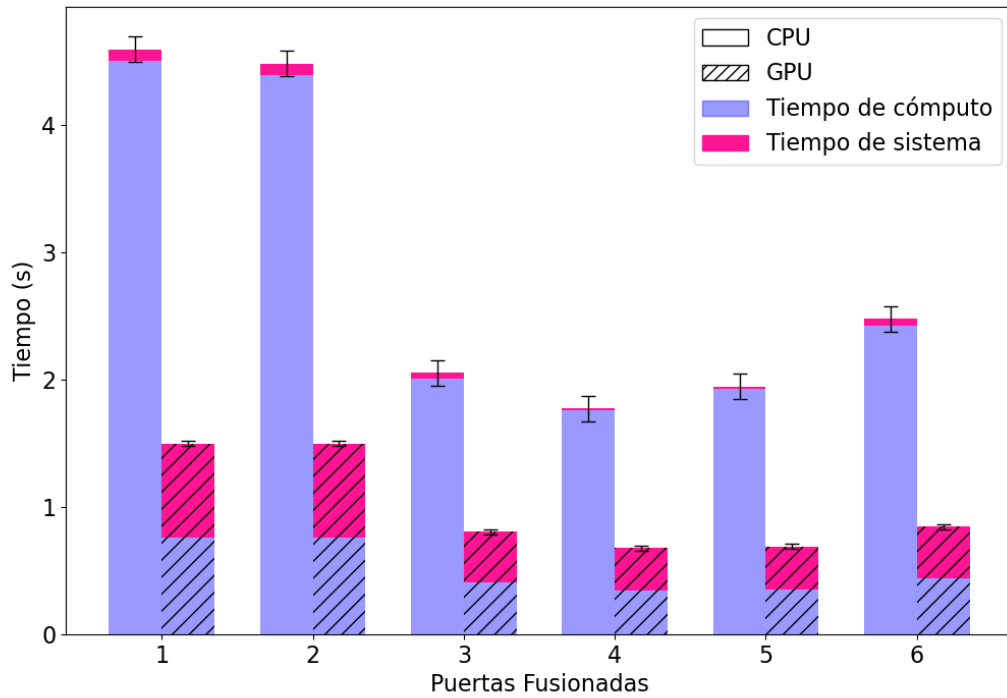


Figura 17: Comparación de los tiempos para $n = 30$ qubits en CPU y GPU, variando el número de puertas fusionadas. Las barras de error corresponden a la suma de tiempo total. La profundidad de los circuitos es $m = 0.7n$.

valores óptimos, han cambiado, siendo 4 en CPU y GPU, aunque la diferencia con 5 puertas fusionadas es muy pequeña para la GPU. Ahora, los valores de 1 y 2 para el parámetro de fusión empeoran con creces los tiempos. La paralelización que se

gana al fusionar pocas puertas no compensa a la paralelización que se necesita para operar todas las puertas por separado o en fusiones de dos.

Los resultados obtenidos concuerdan con las recomendaciones dadas por los desarrolladores de qsim en [39]. Una última diferencia a destacar es que para el sistema de 20 qubits la GPU es más lenta que la CPU, mientras que para 30 qubits se da lo contrario. Por lo tanto, existe un punto crítico entre esos dos tamaños de sistema en el que la GPU supera a la CPU. Para realizar esa comparación, se va a tomar el parámetro de puertas fusionadas con valor de 3 para 20, 21 y 22 qubits, mientras que se toman 4 puertas fusionadas de 23 qubits en adelante.

5.1.4 Comparación CPU vs GPU

En las secciones 5.1.2 y 5.1.3 se han escogido los parámetros adecuados para hacer una comparación justa entre el nodo y la GPU (ver Figura 18). En el intervalo de tamaños de sistema estudiado, la GPU es más lenta hasta los 22 qubits; a partir de ahí supera a las CPUs del nodo. Vemos que su comportamiento en un inicio no es exponencial, pero lo acaba siendo. Se cumple que para tamaños grandes del sistema en el algoritmo de Schrödinger el tiempo está acotado según $O(g2^n)$. El hecho de que en tamaños más pequeños no sea cierto se debe, al igual que en las secciones anteriores, al aumento de tiempo invertido en otras tareas.

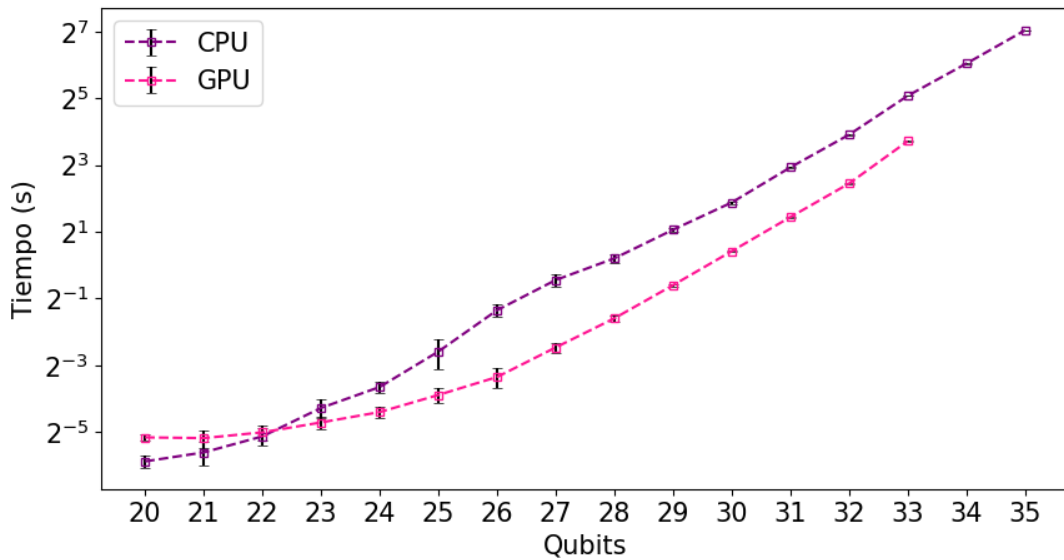


Figura 18: Tiempo de simulación para la GPU y el nodo en función del tamaño del sistema con 40 ciclos de profundidad, en escala logarítmica.

En el caso del nodo, es más complicado establecer cuando deja de tener un comportamiento exponencial debido a las fluctuaciones en los valores medidos. Aún así, la tendencia parece dejar de ser exponencial por debajo de los 25 o 26 qubits, al igual que la GPU.

5.1.5 Efecto del entrelazamiento cuántico

El entrelazamiento afecta al tiempo de simulación negativamente. Se ha representado en la Figura 19 la dependencia del tiempo con el número de ciclos. Los valores siguen una forma de escalera debido a la estructura del circuito; recordamos que se estructura en secuencias de ocho ciclos. Los intervalos de ciclos en los que solo hay una capa de puertas 1-qubit, por ejemplo el intervalo 19-22, se mantienen estables en el tiempo. Sin embargo, en los intervalos en los que los ciclos cuentan con dos capas, siendo una de ellas de puertas 2-qubits, el tiempo escala linealmente, como en el intervalo 24-26. Se verifica la cota para el tiempo de $O(g2^n)$, donde g es el número de puertas de 2 qubits.

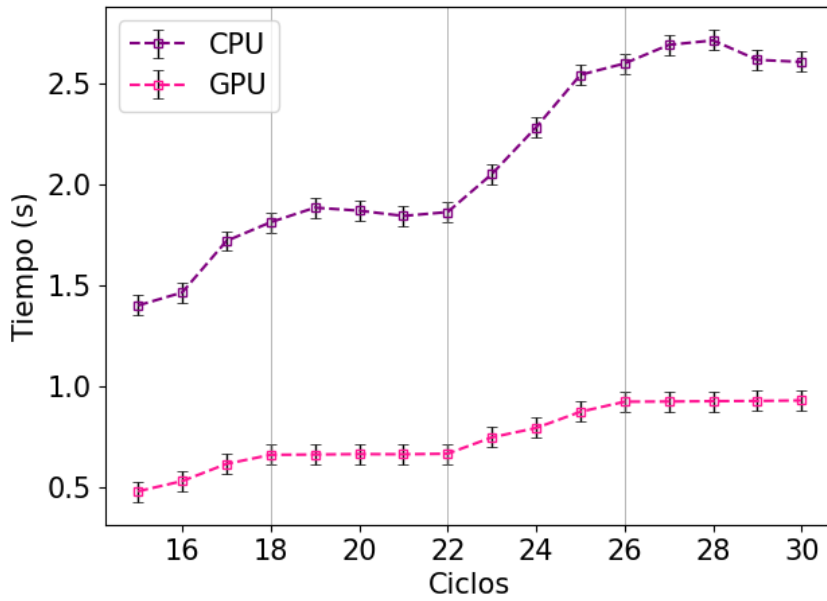


Figura 19: Tiempo de simulación para la GPU y CPU en función del número de ciclos, para un tamaño de 30 qubits. Los cortes verticales señalizan el paso de un ciclo de puertas 1-qubit a puertas 2-qubits y viceversa.

Las puertas 1-qubit requieren manipular el estado de un único qubit, por lo que apenas toman tiempo. Las puertas 2-qubits como las empleadas generan entrelazamiento entre los estados de los qubits sobre los que actúa, requiriendo un mayor número de cálculos, lo que las hace más complejas. La gran capacidad de paralelización de la GPU se traduce en un crecimiento más lento del tiempo con la profundidad.

5.2 Algoritmo Schrödinger-Feynman. Comparación con Schrödinger

El segundo método implementado es el algoritmo de Schrödinger-Feynman, para circuitos aleatorios de 40 ciclos. En la Figura 20 se ha comparado con los resultados obtenidos para el algoritmo de Schrödinger en CPU y GPU. El comportamiento es, como se esperaba, exponencial. La pendiente (trabajando en escala logarítmica) es

menor que en los otros dos casos, cerca de ser la mitad (ver Tabla 1). Si se contrasta, deshaciendo el logaritmo, con los órdenes de crecimiento teóricos, se aprecia como las tendencias de los términos dependientes de n son cercanas a las obtenidas.

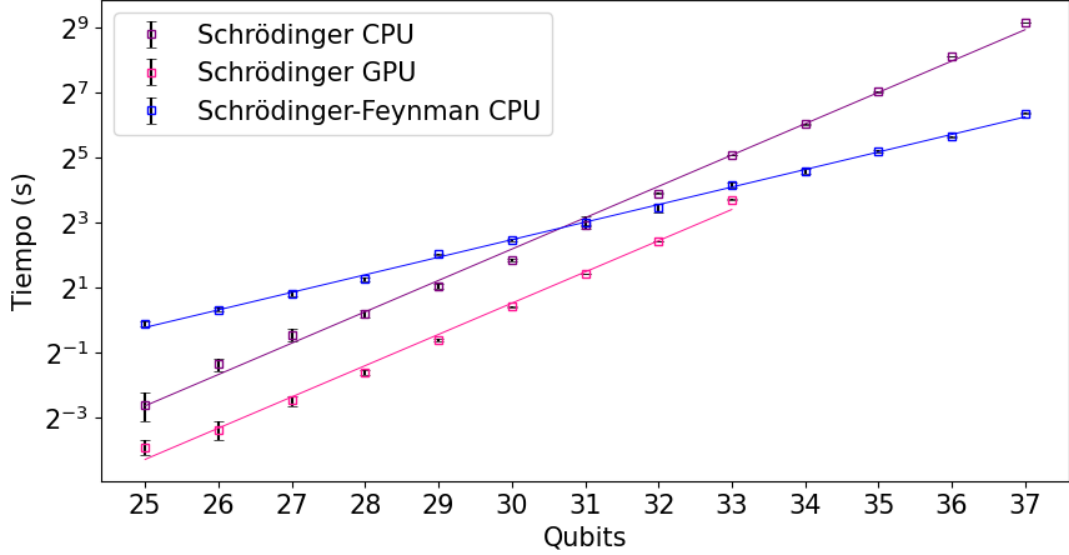


Figura 20: Comparación de los tiempos de simulación del algoritmo de Schrödinger (en CPU y GPU) con el algoritmo de Schrödinger-Feynman (en CPU), en un circuito de 40 ciclos de profundidad. En escala logarítmica.

	Pendiente de ajuste (p)	$t = k \times 2^{pn}$	O teórico
Schrödinger CPU	$(0.96 \pm 0.02)n$	$k \times 2^{0.96n}$	$O(g2^n)$
Schrödinger GPU	$(0.96 \pm 0.03)n$	$k \times 2^{0.96n}$	$O(g2^n)$
Schrödinger-Feynman CPU	$(0.54 \pm 0.01)n$	$k \times 2^{0.54n}$	$O(2 \times 2^{n/2} 4^{m/4})$

Tabla 1: Resultados de los ajustes de la Figura 20. El término k es una constante arbitraria.

El algoritmo SF presenta tiempos más lentos para tamaños pequeños, siendo la peor de las opciones en esos casos. Sin embargo, debido a la tendencia observada, conforme aumenta el tamaño del sistema supera al algoritmo de Schrödinger tanto en GPU como CPU. A partir de 32 qubits, es la mejor opción. Este método muestra una mejora significativa, tanto en tiempo como en memoria, permitiendo aprovechar mejor los recursos. Por ejemplo, supone un mayor alcance para la optimización a partir de hebras si los circuitos no son demasiado profundos. Mientras que con Schrödinger se ha visto (sección 5.1.2) que con 26 qubits el valor óptimo son 32 hebras, con el método de SF con una o dos hebras es suficiente. Debido a que SF divide el circuito en dos partes, cada parte tiene 13 qubits cuyo vector de estado evoluciona siguiendo el método de Schrödinger, para el cual se ha visto que por debajo de 20 qubits el valor óptimo va de dos a una hebra.

Para hacer una comparación más general de estos dos métodos en CPU, se ha representado en las Figuras 21 y 22 un continuo del tiempo en función de los qubits n y el número de ciclos de puertas 2-qubits m , inspiradas en las presentadas

en estas notas [40]. Además, se ha indicado la memoria necesaria para cada tamaño de sistema, teniendo en cuenta que la profundidad del circuito no influye significativamente en la memoria requerida. Se han construido según los órdenes para el crecimiento del tiempo siguientes:

$$\text{Schrödinger} : t \sim Mn2^n \frac{k}{N_{S,cores}} \quad (5.1)$$

$$\text{Schrödinger} - \text{Feynman} : t \sim M \frac{n}{4} 2^{n/2+1} 4^{M/2} \frac{k'}{N_{SF,cores}} \quad (5.2)$$

donde $N_{S,cores}$ y $N_{SF,cores}$ son el número de cores, equivalente al número de hebras, usados en la simulación. Como lo interesante de esta comparación es tener una visión más amplia del comportamiento y alcance de estos algoritmos y, además, se desconoce el número óptimo de hebras para todos los tamaños de sistema, se ha tomado $N_{S,cores} = 64$ y $N_{SF,cores} = 2$. El término M corresponde con el número de ciclos que cuentan con una capa de puertas 2-qubits. Los términos k y k' son constantes tomadas para que la representación coincida aproximadamente con los resultados obtenidos. Al tomar en el eje y únicamente los ciclos de puertas sobre dos qubits se están omitiendo los ciclos de puertas 1-qubits, visto que estas puertas a penas aportan tiempo a la simulación.

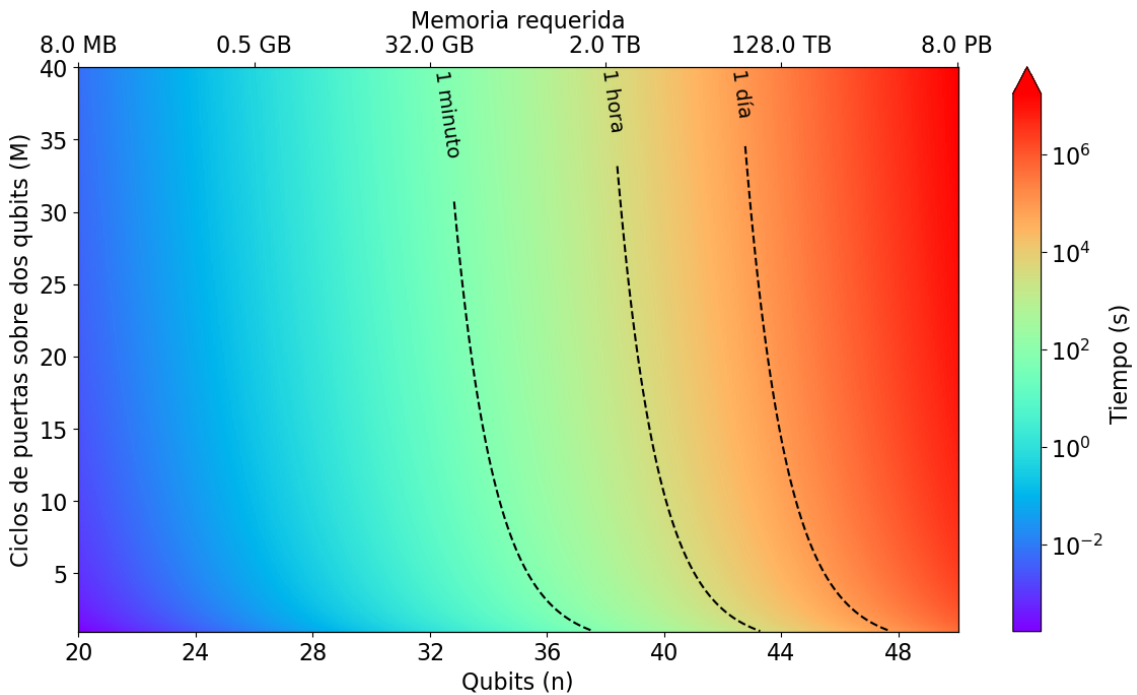


Figura 21: Algoritmo de Schrödinger. El gradiente de color corresponde al tiempo, que depende del número de qubits y del número de ciclos con puertas 2-qubits, según la ecuación 5.1. En el eje horizontal superior se indica la memoria requerida.

Mientras que las isócronas son cercanas a una vertical en el caso de Schrödinger, en Schrödinger-Feynman son diagonales con pendiente negativa. En otras

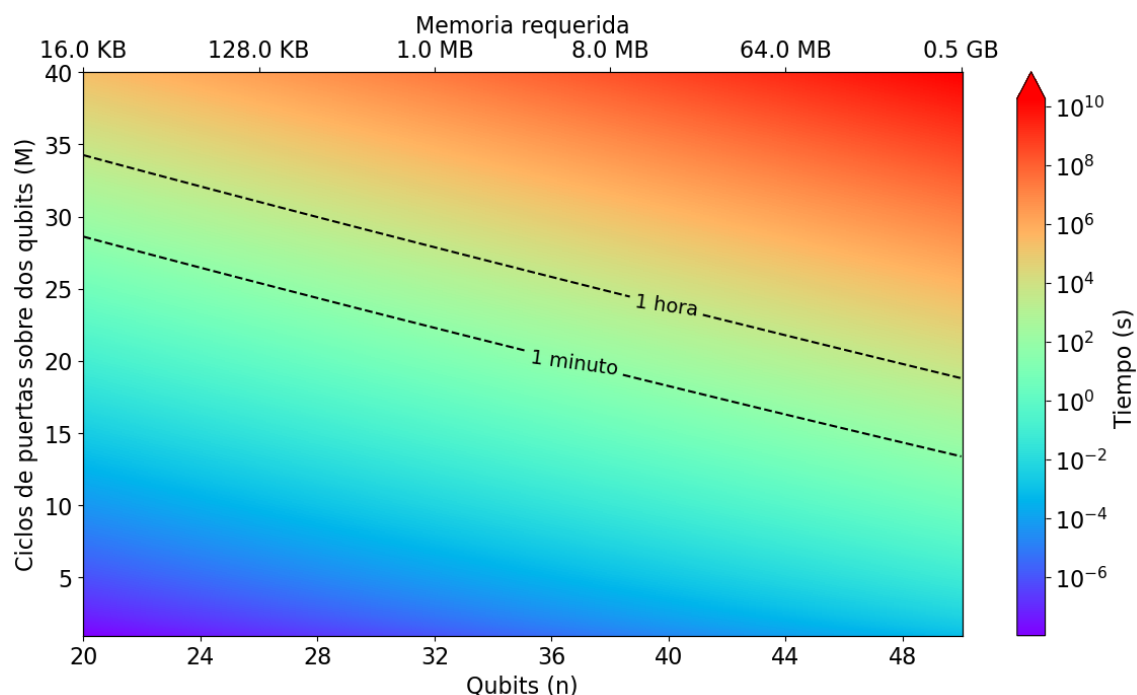


Figura 22: Algoritmo de Schrödinger-Feynman. El gradiente de color corresponde al tiempo, que depende del número de qubits y del número de ciclos con puertas 2-qubits, según la ecuación 5.2. En el eje horizontal superior se indica la memoria requerida.

palabras, en el primer caso se pueden simular circuitos de profundidad muchísimo mayor que en el segundo. El algoritmo SF se ve más afectado por el entrelazamiento de los qubits, ya que se basa en la descomposición de las puertas causantes de este efecto. Sin embargo, permite simular más cantidad de qubits, gracias a la división en subcircuitos. Por ejemplo, un sistema de 48 qubits con 10 capas de puertas 2-qubits toma más de un día de tiempo de simulación mediante Schrödinger, mientras que con SF no llega al minuto. Sumando el factor memoria, ni siquiera es posible simular dicho circuito con Schrödinger, pero sí con SF.

Para concluir con la sección de Resultados y discusión, se recoge en la Tabla 2 un resumen de lo obtenido.

6 Conclusiones

El objetivo principal de este Trabajo de Fin de Grado ha sido realizar simulaciones de alto rendimiento de circuitos cuánticos aleatorios, evolucionando el sistema de qubits mediante el algoritmo de Schrödinger y el algoritmo híbrido de Schrödinger-Feynman. Optimizando los parámetros implicados se han explorado las limitaciones de la simulación clásica de sistemas cuánticos y se ha obtenido una mayor comprensión de su estrecha relación con las arquitecturas del hardware. A lo largo del trabajo, se han comparado diferentes arquitecturas, incluyendo CPU en diferentes configuraciones y GPU, para evaluar su eficiencia.

Este trabajo está motivado por la necesidad de optimizar el rendimiento de

	Ciclos: $m = 0.7d$		Ciclos: $m = 40$	
Qubits (n)	Valor óptimo hebras	Puertas fusionadas	Mejor algoritmo	Mejor arquitectura para el algoritmo
20	2	3	S	CPU
21				3-4
22		4		
23				
...				
25	16			
26	32			
27	32			
...				
30				
31				
32				
...				
33	Límite de memoria de la GPU usando S			
...			SF	CPU
37	Límite de memoria de la CPU usando S)			

Tabla 2: Resumen de resultados. Los valores óptimos de hebras se han recogido según se define en la sección 5.1.2. Los valores de puertas fusionadas se han extendido combinándolos con las recomendaciones de los desarrolladores de qsim. La sigla 'S' corresponde a Schrödinger y 'SF' a Schrödinger-Feynman. Las celdas grises denotan la falta de datos.

simulaciones de sistemas cuánticos, un problema computacionalmente costoso que hace necesaria su implementación en sistemas de computación de alto rendimiento. Estas simulaciones son de gran importancia para el avance de este y otros campos, y la simulación concreta de circuitos cuánticos aleatorios es de especial interés para la demostración de la ventaja cuántica, donde los ordenadores cuánticos superan las capacidades de los clásicos.

El problema práctico abordado ha consistido en la simulación de unos circuitos cuánticos aleatorios que fueron diseñados para evaluar la superioridad cuántica del chip cuántico Sycamore. Están compuestos por numerosos ciclos de puertas 1-qubit y 2-qubit, las últimas presentando efectos de entrelazamiento cuántico entre los qubits sobre los que actúan. Su simulación clásica a gran escala es un desafío vista la gran cantidad de recursos necesarios, consecuencia de la conocida como la maldición de la dimensión. Se han comparado y optimizado los parámetros de fusión de puertas y de número de hebras, además de hacer comparaciones entre GPU y CPU, para el método de Schrödinger. Después, se ha comparado con un método híbrido de los algoritmos de Schrödinger y Feynman, concluyendo cuáles son los tamaños de sistema para los que mejor se aprovechan los recursos. Los resultados obtenidos se enumeran a continuación:

1. Tras las simulaciones realizadas con el algoritmo de Schrödinger, se ha visto que la eficiencia de la paralelización mejora cuando se divide el trabajo en subtareas, incluyendo un coste adicional debido a la gestión de las hebras.

Se ha encontrado que el número óptimo de hebras varía según el tamaño del sistema, siendo de 32 hebras para tamaños de 26 y 27 qubits pero reduciéndose a solo 2 para 20 qubits, ya que se trata de un sistema pequeño que no necesita una gran paralelización.

2. Respecto al parámetro de puertas fusionadas, el valor óptimo se encuentra en 3 para sistemas de 20 qubits y en 4 para sistemas de 30 qubits, tanto en CPU como en GPU. Con estos valores, se asegura que la matriz resultante de la fusión de las puertas no supere la capacidad de la memoria caché, evitando así un exceso de transferencias de datos hacia y desde memorias más lentas y lejanas.
3. Una vez definidos los parámetros adecuados, se ha hecho una comparación directa entre CPU y GPU con el método de Schrödinger. La CPU es más rápida hasta los 22 qubits, pero para mayor tamaño es superada por la GPU, gracias a su alta capacidad de manejar múltiples operaciones en paralelo. En ambos casos se cumple que para tamaños grandes del sistema el tiempo está acotado según $O(g2^n)$. Por otro lado, variando la profundidad del circuito se observa la dependencia lineal de esta cota con g , el número de puertas 2-qubits. Este aumento lineal del tiempo se debe al entrelazamiento que generan dichas puertas sobre los qubits.
4. Por último, se han descrito las diferencias entre el algoritmo de Schrödinger en CPU GPU y el de Schrödinger-Feynman en CPU, comparando los tiempos en el intervalo de 25 a 37 qubits. El algoritmo SF presenta tiempos más lentos con pocos qubits, pero a partir de 32 qubits es la mejor opción. Este método muestra una mejora significativa, tanto en tiempo como en memoria, pasando del requisito de $O(2^n)$ para Schrödinger al de $O(2^{2/2+1})$, permitiendo un uso más eficiente de los recursos. Haciendo una comparación más amplia de las tendencias de estos algoritmos, se ha visto que el algoritmo SF se ve más afectado por el entrelazamiento de los qubits, ya que se basa en la descomposición de las puertas causantes de este efecto. En contraste, tiene la ventaja de simular más cantidad de qubits, gracias a la división en subcircuitos.

Junto a los resultados logrados, se han abierto nuevas vías de trabajo. El comportamiento del método de Schrödinger-Feynman y de la GPU para muchos qubits es notable, y la combinación de ambos en estas condiciones parece prometedora. Además, todo lo estudiado cambia si se trabaja con circuitos cuánticos con ruido, que intentan simular las condiciones no ideales de los ordenadores cuánticos. Se podrían optimizar los recursos para este caso. Las múltiples oportunidades que se desdoblan acentúan la inmensa amplitud de la rama de la computación cuántica, que continúa expandiéndose.

Referencias

- [1] Y. Manin, "Computable and Uncomputable", n.º 128, 1980.
- [2] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines", *Journal of Statistical Physics*, vol. 22, n.º 5, págs. 563-591, mayo de 1980. DOI: [10.1007/BF01011339](https://doi.org/10.1007/BF01011339). dirección: <https://doi.org/10.1007/BF01011339>.
- [3] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer", *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, n.º 1818, págs. 97-117, jul. de 1985. DOI: [10.1098/rspa.1985.0070](https://doi.org/10.1098/rspa.1985.0070).
- [4] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring", en *Proceedings 35th Annual Symposium on Foundations of Computer Science*, nov. de 1994, págs. 124-134. DOI: [10.1109/SFCS.1994.365700](https://doi.org/10.1109/SFCS.1994.365700).
- [5] L. K. Grover, *A fast quantum mechanical algorithm for database search*, nov. de 1996. DOI: [10.48550/arXiv.quant-ph/9605043](https://doi.org/10.48550/arXiv.quant-ph/9605043).
- [6] Y. Aharonov, L. Davidovich y N. Zagury, "Quantum random walks", *Physical Review A*, vol. 48, n.º 2, págs. 1687-1690, ago. de 1993. DOI: [10.1103/PhysRevA.48.1687](https://doi.org/10.1103/PhysRevA.48.1687).
- [7] J. Kempe, "Quantum random walks - an introductory overview", en *Contemporary Physics*, vol. 44, n.º 4, págs. 307-327, jul. de 2003, 1427 citations (Semantic Scholar/arXiv) [2024-06-09] 1427 citations (Semantic Scholar/DOI) [2024-06-09] arXiv:quant-ph/0303081. DOI: [10.1080/00107151031000110776](https://doi.org/10.1080/00107151031000110776).
- [8] *Trabajos de Fin de Grado en Física para el curso 2023/2024*. dirección: <https://fciencias.ugr.es/estudios/titulos-de-grado/2-principal/4287-trabajos-de-fin-de-grado-en-fisica-para-el-curso-2023-2024> (visitado 03-06-2024).
- [9] H. E. Brandt, "Qubit devices and the issue of quantum decoherence", *Progress in Quantum Electronics*, vol. 22, n.º 5, págs. 257-370, sep. de 1999. DOI: [10.1016/S0079-6727\(99\)00003-8](https://doi.org/10.1016/S0079-6727(99)00003-8).
- [10] F. Cavaliere, J. Mattsson y B. Smeets, "The security implications of quantum cryptography and quantum computing", *Network Security*, vol. 2020, n.º 9, págs. 9-15, sep. de 2020. DOI: [10.1016/S1353-4858\(20\)30105-7](https://doi.org/10.1016/S1353-4858(20)30105-7).
- [11] F. Arute, K. Arya, R. Babbush et al., "Quantum supremacy using a programmable superconducting processor", *Nature*, vol. 574, n.º 7779, págs. 505-510, oct. de 2019. DOI: [10.1038/s41586-019-1666-5](https://doi.org/10.1038/s41586-019-1666-5).
- [12] C. Huang, F. Zhang, M. Newman et al., *Classical Simulation of Quantum Supremacy Circuits*, mayo de 2020. DOI: [10.48550/arXiv.2005.06787](https://doi.org/10.48550/arXiv.2005.06787).
- [13] R. Kashino, R. Kobayashi, N. Fujita y T. Boku, "Multi-hetero Acceleration by GPU and FPGA for Astrophysics Simulation on oneAPI Environment", en *International Conference on High Performance Computing in Asia-Pacific Region*, ép. HPCAsia '22, New York, NY, USA: Association for Computing Machinery, 2022, págs. 84-93. DOI: [10.1145/3492805.3492817](https://doi.org/10.1145/3492805.3492817).

- [14] C. Augonnet, D. Goudin, A. Pujols y M. Sesques, "Accelerating a Massively Parallel Numerical Simulation in Electromagnetism Using a Cluster of GPUs", en *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski y J. Waśniewski, eds., Berlin, Heidelberg: Springer, 2014, págs. 593-602. DOI: [10.1007/978-3-642-55224-3_55](https://doi.org/10.1007/978-3-642-55224-3_55).
- [15] R. Ben-Shalom, A. Ladd, N. S. Artherya et al., "NeuroGPU: Accelerating multi-compartment, biophysically detailed neuron simulations on GPUs", *Journal of Neuroscience Methods*, vol. 366, pág. 109400, ene. de 2022. DOI: [10.1016/j.jneumeth.2021.109400](https://doi.org/10.1016/j.jneumeth.2021.109400).
- [16] M. Januszewski, A. Ptok, D. Crivelli y B. Gardas, "GPU-based acceleration of free energy calculations in solid state physics", *Computer Physics Communications*, vol. 192, págs. 220-227, jul. de 2015. DOI: [10.1016/j.cpc.2015.02.012](https://doi.org/10.1016/j.cpc.2015.02.012).
- [17] D. Ramon, F. Steinmetz, D. Jolivet, M. Compiègne y R. Frouin, "Modeling polarized radiative transfer in the ocean-atmosphere system with the GPU-accelerated SMART-G Monte Carlo code", *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 222-223, págs. 89-107, ene. de 2019. DOI: [10.1016/j.jqsrt.2018.10.017](https://doi.org/10.1016/j.jqsrt.2018.10.017).
- [18] M. A. Nielsen e I. L. Chuang, *Quantum computation and quantum information*, 10th anniversary ed. Cambridge ; New York: Cambridge University Press, 2010.
- [19] R. Jozsa y N. Linden, "On the role of entanglement in quantum computational speed-up", *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 459, n.º 2036, págs. 2011-2032, ago. de 2003, 660 citations (Semantic Scholar/arXiv) [2024-06-04] 660 citations (Semantic Scholar/DOI) [2024-06-04] arXiv:quant-ph/0201143. DOI: [10.1098/rspa.2002.1097](https://doi.org/10.1098/rspa.2002.1097).
- [20] M. Van den Nest, "Universal Quantum Computation with Little Entanglement", *Physical Review Letters*, vol. 110, n.º 6, pág. 060504, feb. de 2013. DOI: [10.1103/PhysRevLett.110.060504](https://doi.org/10.1103/PhysRevLett.110.060504).
- [21] S. Markidis, "Enabling Quantum Computer Simulations on AMD GPUs: a HIP Backend for Google's qsim", en *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, Denver CO USA: ACM, nov. de 2023, págs. 1478-1486. DOI: [10.1145/3624062.3624223](https://doi.org/10.1145/3624062.3624223).
- [22] T. Häner y D. S. Steiger, "0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit", en *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, nov. de 2017, págs. 1-10. DOI: [10.1145/3126908.3126947](https://doi.org/10.1145/3126908.3126947).
- [23] M. Smelyanskiy, N. P. D. Sawaya y A. Aspuru-Guzik, *qHiPSTER: The Quantum High Performance Software Testing Environment*, mayo de 2016. DOI: [10.48550/arXiv.1601.07195](https://doi.org/10.48550/arXiv.1601.07195).
- [24] T. H. Cormen, ed., *Introduction to algorithms*, 3rd ed. Cambridge, Mass: MIT Press, 2009.
- [25] 1. *Introducción al servicio HPC – PROTEUS*, ene. de 2024. dirección: <https://proteus.ugr.es/docs/manual-uso/1-high-performance-computing/> (visitado 03-06-2024).
- [26] 4th Gen AMD EPYC™ Processor Architecture. dirección: <https://www.amd.com/es/products/processors/server/epyc/4th-generation-architecture.html> (visitado 05-06-2024).

- [27] *November 2021 | TOP500*. dirección: <https://www.top500.org/lists/top500/2021/11/> (visitado 03-06-2024).
- [28] H. Jeon, "GPU Architecture", en *Handbook of Computer Architecture*, A. Chattopadhyay, ed., Singapore: Springer Nature, 2022, págs. 1-29. DOI: [10.1007/978-981-15-6401-7_66-1](https://doi.org/10.1007/978-981-15-6401-7_66-1).
- [29] S. Tsutsui y P. Collet, eds., *Massively Parallel Evolutionary Computation on GPGPUs* (Natural Computing Series). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.
- [30] *NVIDIA Ampere Architecture*. dirección: <https://resources.nvidia.com/c/ampere-architecture-white-paper?x=sfvhf4>.
- [31] F. Arute, K. Arya, R. Babbush et al., "Supplementary information for "Quantum supremacy using a programmable superconducting processor"", *Nature*, vol. 574, n.º 7779, págs. 505-510, oct. de 2019. DOI: [10.1038/s41586-019-1666-5](https://doi.org/10.1038/s41586-019-1666-5).
- [32] *quantumlib/Cirq*, jun. de 2024. dirección: <https://github.com/quantumlib/Cirq> (visitado 03-06-2024).
- [33] *quantumlib/qsim*, mayo de 2024. dirección: <https://github.com/quantumlib/qsim> (visitado 03-06-2024).
- [34] I. L. Markov, A. Fatima, S. V. Isakov y S. Boixo, *Quantum Supremacy Is Both Closer and Farther than It Appears*, arXiv:1807.10749 [quant-ph], sep. de 2018. dirección: <http://arxiv.org/abs/1807.10749>.
- [35] K. Young, M. Scese y A. Ebneenasir, *Simulating Quantum Computations on Classical Machines: A Survey*, nov. de 2023. dirección: <http://arxiv.org/abs/2311.16505>.
- [36] *Repositorio de códigos usados*. dirección: <https://github.com/joyeuxnoelia/TFG-Grado-F-sica-UGR-Noelia-S-nchez-G-mez>.
- [37] *scalene: Scalene: A high-resolution, low-overhead CPU, GPU, and memory profiler for Python with AI-powered optimization suggestions*. dirección: <https://github.com/plasma-umass/scalene> (visitado 03-06-2024).
- [38] H.-h. Miao e Y. I. Ozthigov, *Distributed computing quantum unitary evolution*, abr. de 2024. DOI: [10.48550/arXiv.2403.06937](https://doi.org/10.48550/arXiv.2403.06937).
- [39] *Choosing hardware for your qsim simulation | Quantum Simulator*. dirección: https://quantumai.google/qsim/choose_hw (visitado 09-06-2024).
- [40] P. Roushan, A. Zalcman y C. Neill, "Lecture 3. The Quantum Supremacy Experiment",
- [41] malishoaib, *Sample Size Estimation for Machine Learning Models Using Hoeffding's Inequality*, sep. de 2017. dirección: <https://malishoaib.wordpress.com/2017/09/08/sample-size-estimation-for-machine-learning-models-using-hoeffdings-inequality/> (visitado 03-06-2024).

A Profiler: Scalene

Los valores de tiempo se han obtenido apartir de Scalene, un profiler de Python en CPU y GPU de alto rendimiento [37]. Scalene realiza el perfilado a nivel de línea

y por función, señalando las funciones y las líneas específicas de código responsables del tiempo de ejecución en el programa, dividiéndolo en tiempo de Python, tiempo nativo y tiempo de sistema:

- **Tiempo Python:** Este tiempo es el que el intérprete de Python dedica a la ejecución de funciones y operaciones puramente en Python, sin tener en cuenta extensiones o librerías en otros lenguajes.
- **Tiempo nativo:** El resto del tiempo que el intérprete pasa ejecutando código se recoge aquí. Por ejemplo, el tiempo empleado en ejecutar librerías como numpy, scipy o qsim.
- **Tiempo de sistema:** Es el tiempo que se invierte en el sistema operativo en hacer llamadas al sistema y operaciones de entrada y salida, como leer o escribir en el disco.

En este trabajo se han dividido en dos tipos de tiempo para facilitar la comprensión: por un lado, se han unido el tiempo de Python y el nativo y se han denominado tiempo de cómputo. Por otro, se ha tomado el tiempo de sistema. En Además de rastrear el uso de la CPU, Scalene también señala las líneas específicas de código responsables del crecimiento de la memoria.

Scalene usa técnicas de muestreo para medir el tiempo de ejecución y el uso de memoria. La frecuencia de muestreo es un parámetro a modificar por el usuario. Es importante usar la frecuencia de muestreo adecuada al código que se desea estudiar, ya que si se da un evento demasiado rápido puede ser ignorado por el profiler. Tampoco puede usarse una frecuencia excesivamente alta, ya que puede aumentar la sobrecarga en la ejecución del trabajo. Para tomar el valor de muestreo se debe tener en cuenta el tamaño de tiempo que toma la línea de código que se desea estudiar. En el caso de 20 qubits, en los que los tiempos son del orden de 10^{-3} segundos, no debe usarse un tiempo de muestreo superior a este valor, visto que Scalene daría valores de tiempo mayores o lo pasaría por alto.

B Incertidumbres. Desigualdad de Hoeffding

Para estimar el tamaño de repeticiones necesarias de cada simulación para que las incertidumbres estimadas sean adecuadas para cada valor se ha empleado la desigualdad de Hoeffding's [41]:

$$Pr \left[\left| \frac{1}{n} \sum_{i=1}^n Z_i - E[Z] \right| \geq \epsilon \right] \leq 2 \exp(-2n\epsilon^2) \quad (\text{B.1})$$

Esto es, la probabilidad de que la diferencia entre la media de muestreo Z y la media real $E(Z)$ sea mayor que algún valor ϵ pequeño. La exponencial del término a la derecha está acotada por arriba por un factor α , que da una confianza del $100(1 - \alpha) \%$ de que la diferencia entre las medias de la muestra y la población no se desvía de ϵ . Con esta cota se puede calcular el tamaño de la muestra mínimo para un ϵ y un α dados:

$$2 \exp(-2n\epsilon^2) \leq \alpha \quad \longrightarrow \quad n \geq -\frac{1}{2\epsilon^2} \ln\left(\frac{\alpha}{2}\right) \quad (\text{B.2})$$

Las las incertidumbres de tiempo de este trabajo se han tomado con, al menos, un 90 % de confianza en todos los casos, y un valor de ϵ acorde con el orden de magnitud de la medida, dentro de lo permitido por los tiempos de simulación.