Découverte de l'informatique quantique, état de l'art et tests sur machine IBM

Thomas Cluzel, Claude Mazel, David Hill

Rapport de Recherche LIMOS – RR-2020-0304

ISIMA/LIMOS UMR CNRS 6158
Campus des Cézeaux . 1 rue de la Chébarde . TSA 60125
F-63178 Aubière CEDEX
FRANCE

Roles:

Thomas Cluzel : Rédaction, Conceptualisation, Méthodologie, Développement logiciel et analyse des résultats

David Hill : Révision et édition, Conceptualisation, Méthodologie, Validation, analyse des résultats, Supervision et administration de projet.

Claude Mazel : Révision et édition, Conceptualisation, Méthodologie, Validation, analyse des résultats, Supervision et administration de projet.

Résumé

Le but de ce rapport de recherche était de dresser un état de l'art de l'avancement en **informatique quantique dans le cadre d'un projet du Master de recherche international de l'école d'ingénieur ISIMA**. Pour cela, nous avons d'abord étudié les principes de la physique quantique nécessaires à la compréhension de l'informatique quantique, à savoir, la superposition quantique, l'intrication quantique et la réduction du paquet d'onde. Ces principes permettent de définir le concept de **qubit** qui est le support de l'information quantique. Le qubit est limité par le phénomène de **décohérence** qui empêche les qubits de rester longtemps dans un état superposé. De plus, il est impossible de copier l'état d'un qubit à cause du théorème de non-clonage. Les qubits peuvent être manipulés avec des **portes quantiques** qui forment alors des **circuits quantiques**.

Au cours de ce projet, nous avons analysé l'**algorithme de Grover** qui est un algorithme quantique de recherche. Puis, nous l'avons implémenté à l'aide de la bibliothèque **Python Qiskit** développée par IBM. Enfin, nous avons exécuté cet algorithme sur un véritable **ordinateur quantique** mis à disposition via la plateforme **IBM Q Experience** et nous avons constaté des problèmes de reproductibilité.

A ce jour, les ordinateurs quantiques ne sont capables de manipuler qu'un nombre restreint de qubits et de leurs appliquer un nombre restreint de portes quantiques mais cela a été suffisant pour que Google affirme avoir atteint la **suprématie quantique** en octobre 2019.

Mots-clés : informatique quantique, qubit, décohérence, portes quantiques, circuits quantiques, algorithme de Grover, Python, Qiskit, ordinateur quantique, IBM Q Experience, suprématie quantique.

Abstract

The goal of this research report was to draw the state of the art of the advancement in **quantum computing as part of a project of the International Master Research of the ISIMA engineering school**. To do so, we first studied the principles of quantum physics required to understand quantum computing, namely, quantum superposition, quantum entanglement and wave-function collapse. These principles define a **qubit** which represents the quantum information. Qubits are, however, limited by **decoherence**, which prevents them to stay in a superposed state for a long period of time, and the no-cloning theorem, which prevents the copy of the superposed state of a qubit. We can interact with qubits thanks to **quantum gates** which composed **quantum circuits**.

Throughout this project, we analysed **Grover's algorithm** which is a quantum search algorithm. Then, we implemented it with **Qiskit**, a **Python**'s library developed by IBM. Eventually, we executed it on a real **quantum computer** available on **IBM Q Experience** and we found reproducibility issues.

Today, quantum computers cannot handle many qubits nor apply many quantum gates, but it was enough for Google to claim to have reached **quantum supremacy** in October 2019.

Keywords: quantum computing, qubit, decoherence, quantum gates, quantum circuits, Grover's algorithm, Qiskit, Python, quantum computer, IBM Q Experience, quantum supremacy.

Table des matières

Table des figures et illustrations

Introduction

En 1843, Ada Lovelace publie le premier programme informatique de l'histoire. Ce dernier était destiné à être exécuté par la machine analytique de Charles Babbage. Cependant, cette machine n'existait pas encore à cette époque. Ada Lovelace mourut avant d'avoir pu tester son programme.

L'informatique quantique en est aujourd'hui au même point qu'Ada Lovelace et Charles Babbage. Les concepts de l'informatique quantique ont été énoncés il y a plusieurs décennies et des algorithmes quantiques basés sur ces principes ont également été conçus au cours de cette période. Toutefois, nous ne disposons pas encore de d'ordinateurs quantiques permettant d'exploiter ces algorithmes. Les ordinateurs quantiques actuels sont limités en termes de temps de calculs et de mémoire mais se développent d'année en année.

Au lendemain de l'annonce par Google de la suprématie quantique*[1] [Arute *et al.* 2019], ce rapport dresse un état de l'art de l'avancement en informatique quantique.

Nous décrirons d'abord les concepts de physique quantique utiles en informatique quantique. Puis nous aborderons les algorithmes quantiques et analyserons l'algorithme de Grover. Enfin nous aborderons les technologies déjà existantes et utiliserons un véritable ordinateur quantique*.

[1] Tous les termes suivis d'une « * » sont définis dans le lexique.

Etat de l'art

L'idée d'utiliser les principes de la physique quantique en informatique remonte à 1980, quand Paul Benioff énonce l'idée d'une machine de Turing quantique [Benioff 1980]. 5 ans plus tard, David Deutsch montre que toute fonction calculable par un ordinateur classique peut être implémentée par un circuit quantique* [Deutsch Penrose 1985].

Il faut ensuite attendre 1992 pour voir naître le premier algorithme quantique : l'algorithme de Deutsch-Jozsa [Deutsch Jozsa 1992]. Le but de cet algorithme est de déterminer si une fonction booléenne $f : \{0,1\}^n \rightarrow \{0,1\}$ est constante ou équilibrée, c'est-à-dire si elle renvoie toujours 0 ou 1 ou si elle renvoie 0 pour la moitié des entrées et 1 pour l'autre moitié. Cet algorithme a certes peu d'utilité en pratique mais a le mérite d'être un pionnier dans son domaine. De plus, il est déterministe, ce qui est rare pour un algorithme quantique.

D'autres algorithmes suivent celui de Deutsch-Jozsa. Notamment l'algorithme de Shor en 1994 [Shor 1994] qui permet de factoriser un nombre en un produit de facteurs premiers en temps polynomial ainsi que l'algorithme de Grover en 1996 [Grover 1996] qui permet de rechercher un élément satisfaisant une propriété dans un ensemble quelconque avec une complexité de $O(\sqrt{n})$.

La théorie de l'informatique quantique se développe peu à peu mais jusque-là rien de concret n'avait été expérimenté. En 1998, le premier ordinateur quantique de l'Histoire exécute l'algorithme de Deutsch-Jozsa [Jones Mosca 1998]. Il s'agit d'un ordinateur quantique de 2 qubits utilisant la résonance magnétique nucléaire. A partir de ce moment l'informatique quantique cesse d'être une théorie et devient une réalité.

Fort de ce succès, les ordinateurs quantiques se sont, à leur tour, développés. C'est ainsi qu'en 2001, IBM annonce avoir factorisé le nombre 15 en 3×5 avec un ordinateur quantique doté de 7 qubits [Vandersypen *et al.* 2001]. L'utilité des algorithmes quantiques est alors mise en évidence. Le seul obstacle qui se dresse devant eux est la mise à l'échelle verticale. Le nombre de qubits disponibles et le temps de calcul possible sont les deux principaux paramètres à améliorer sur les ordinateurs quantiques.

Ainsi, depuis les années 1980, de nouveaux algorithmes quantiques sont découverts pour de nombreuses applications. Pour exécuter ses algorithmes, les ordinateurs quantiques deviennent de plus en plus puissants et abordables. En janvier 2019, au CES de Las Vegas, IBM présente le « IBM Q System One » : le premier ordinateur quantique commercial. Il possède 20 qubits [1].

En octobre 2019, Google annonce avoir atteint la suprématie quantique [Preskill 2012], c'est-à-dire avoir construit un ordinateur quantique qui surpasse n'importe quel ordinateur classique [Arute *et al.* 2019]. Cet ordinateur, nommé Sycamore, possède 53 qubits. Il a résolu le problème d'échantillonnage d'un circuit quantique en 200 secondes. Selon Google, il aurait fallu 10 000 ans à Summit, le plus puissant supercalculateur du monde, pour résoudre ce problème. Selon IBM, concepteur de Summit, le même résultat peut être obtenu en 2 jours et demi en utilisant toute la mémoire du supercalculateur [2]. A noter qu'IBM possède également un ordinateur quantique de 53 qubits [3].

En marge des recherches sur les algorithmes quantiques et la course à la suprématie quantique, ont émergé les outils de développement pour l'informatique quantique. Le premier langage de scripts quantique, le QCL (Quantum Computation Language), est apparu en 1998 [Ömer 1998]. Il n'était alors que purement théorique et n'avait pour but que de proposer un formalisme pour la conception de programmes informatiques quantiques. D'autres types d'outils ont par la suite été créés comme des outils de programmation graphiques permettant de faire des glisser-déposer* de portes quantiques sur des partitions* (Quirk par exemple), des bibliothèques pour des langages de programmation classiques comme Qiskit ou Cirq. Il existe aussi des simulateurs quantiques en C comme QuEST, en C++ comme Quantum++, en Lisp comme QVM, en Haskell comme Quipper, en Java comme jQuil, en Javascript comme Quirk, en OCaml comme QOCS, en Python* comme QuPy. Il y a également des langages conçus pour servir d'intermédiaires entre les langages de hauts niveaux et les ordinateurs quantiques comme OpenQASM [Cross *et al.* 2017] et OpenFermion. Enfin, il existe même des compilateurs pour des langages informatiques quantiques comme le Q# [Prieur 2019].

Parmi les acteurs majeurs de l'informatique quantique aujourd'hui, il y a des grandes entreprises depuis longtemps ancrées dans le secteur de l'informatique comme IBM, Microsoft, Google, Atos ou Intel, des start-ups comme Rigetti Computing ainsi que des centres de recherches. Ces différents acteurs se positionnent sur les différents secteurs de l'informatique quantique. Par exemple Microsoft édite le langage Q#. Google a conçu Sycamore. D-Wave est spécialisé dans le recuit quantique, une application spécifique de l'informatique quantique. Les chercheurs comme Peter Shor découvrent des algorithmes quantiques…

I. Concepts de physique quantique pour l'informatique quantique

A. Principes physiques indispensables

La physique classique permet d'expliquer des phénomènes à l'échelle macroscopique. Cependant, les théories et modèles de la physique classique ne fonctionnent plus à l'échelle microscopique. Pour décrire le comportement des particules, il est nécessaire d'utiliser les théories quantiques. Le domaine de la physique quantique le plus utile pour cette tâche est la mécanique quantique. De nombreuses lois existent en physique quantique comme la quantification (un atome ne peut être excité qu'à des niveaux précis d'énergies), la dualité onde-particule (le comportement d'une particule peut être décrit comme une onde ou comme un corps) ou encore le principe d'indétermination de Heisenberg ou principe d'incertitude (il existe une limite fondamentale à la précision avec laquelle on peut mesurer plusieurs propriétés d'une particule simultanément). Toutefois, toutes ces lois ne sont pas nécessaires à l'appréhension de l'informatique quantique.

Trois principes de la physique quantique sont fondamentaux en informatique quantique :

- La superposition quantique.
- L'intrication quantique.
- La réduction du paquet d'onde.

1. Superposition quantique

En physique classique, une particule se trouve dans un état donné à un instant donné. La superposition quantique* énonce, au contraire, qu'une particule peut se trouver dans une superposition d'états.

Soit $n \in \mathbb{N}$, $\{|e_1\rangle, \dots, |e_n\rangle\}$ l'ensemble des états propres* dans lesquels peut se trouver une particule et $\{a_1, \dots, a_n\} \in \mathbb{C}^n$, alors l'état superposé de la particule est :

$$|\psi\rangle = \sum_{i=0}^{n} a_i |e_i\rangle$$

Un état quantique est noté $|e\rangle$ et prononcé « ket e » qui est une notation simplifiée d'une matrice colonne.

$$|e\rangle = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix}$$

Ainsi, un état superposé n'est autre qu'une combinaison linaire de vecteurs.

La notation $\langle l|$ existe également, se prononce « bra l » et est une notation simplifiée d'une matrice ligne.

$$\langle l| = [l_1\ l_2\ \dots\ l_m]$$

De plus, la notation $\langle l|e\rangle$ désigne le produit scalaire des vecteurs l et e. Ces notations ont été introduite par Paul Dirac en 1939 pour simplifier l'écriture des équations de mécaniques quantiques [Dirac 1939].

Pour mieux appréhender le concept de superposition quantique, il est possible de faire une analogie avec la logique floue en informatique ou la théorie des ensembles flous en mathématique [Zadeh 1965]. Une tâche récurrente en science de la donnée est la classification. Supervisée ou non-supervisée, la classification consiste à déterminer dans quelle classe de données une donnée doit être classée. Souvent, les algorithmes de classification produisent un vecteur en sortie. Ce vecteur comporte des 0 et un 1. L'indice auquel se trouve ce 1 correspond à la classe à laquelle la donnée appartient selon l'algorithme.

$$classification(donnée) = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}, \forall i \in \{1, \dots, n\}, a_i \in \{0, 1\}$$

Par exemple : $\begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ et la donnée est classée dans la quatrième classe.

La logique floue, quant à elle, permet de classifier une donnée non pas dans une classe mais sur plusieurs classes. Au lieu de donner des 0 et un 1, le vecteur de sortie est constitué de réels entre 0 et 1 dont la somme vaut 1. Cette sortie peut être interprétée comme une probabilité d'appartenance à une classe ou comme une proportion d'appartenance à une classe. En théorie des ensembles, cette analogie reste la même mais les classes sont appelées des ensembles.

$$classification\_floue(donnée) = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}, \forall i \in \{1, \dots, n\}, a_i \in [0, 1] \; ; \; \sum_{i=0}^{n} a_i = 1$$

Toutefois, cette analogie n'est pas parfaite. En effet, les coefficients des états propres d'un état de superposition quantique sont des nombres complexes et non pas des réels. C'est le carré de leur module qui doit être entre 0 et 1.

$$|\psi\rangle = \sum_{i=0}^{n} a_i |e_i\rangle \, , \forall i \in \{1, \dots, n\}, |a_i|^2 \in [0, 1] \; ; \; \sum_{i=0}^{n} |a_i|^2 = 1$$

Maintenant que nous avons découvert le principe de superposition quantique, intéressons-nous à l'intrication quantique.

2. Intrication quantique

Dire que deux particules sont intriquées, c'est dire que deux particules sont liées. Par conséquent, si l'on connait l'état de la première particule, on peut déduire l'état de la seconde particule.

L'intrication quantique* [Fickler *et al.* 2013] est aussi appelée enchevêtrement quantique.

Nous avons décrit ce qu'il y a à savoir sur l'intrication quantique pour faire de l'informatique quantique. Voyons maintenant le dernier concept indispensable : la réduction du paquet d'onde.

3. Réduction du paquet d'onde

Dans certaines sciences, comme l'électronique, effectuer une mesure d'un système influe sur ce dernier. Il est possible de dire que : « J'observe donc je perturbe. ». En effet, lorsque l'on effectue une mesure sur un circuit électronique, il faut prendre en compte le fait que l'appareil de mesure est également un élément du circuit. Il peut disposer de sa propre résistance par exemple. Ainsi, effectuer une mesure revient à interagir avec le système.

En physique quantique, un système quantique est un ensemble de particules dans un état de superposition quantique. Certaines particules du système peuvent être intriquées. Un tel système est représenté par une fonction d'onde quantique.

Le principe de réduction du paquet d'onde* [Heisenberg 1927] énonce que mesurer l'état d'un système quantique transforme son état en l'état mesuré. La mesure réduit la fonction d'onde à un de ses états propres.

Il est important de remarquer les conséquences suivantes :

- Mesurer l'état d'un système quantique détruit son état superposé.
- Il est impossible de restaurer l'état superposé qu'un système avait avant une mesure. Il est, par conséquent, impossible d'effectuer plusieurs mesures consécutives de l'état d'un système superposé.
- Il est impossible de connaître les valeurs des coefficients complexes des états propres de l'état superposé d'un système.

Ainsi, en physique quantique : « Je mesure donc je réduis. ». La mesure peut être vue comme une réalisation d'une expérience aléatoire dont les probabilités sont les carrés des modules des coefficients. Soit $|\psi\rangle$ l'état superposé d'un système quantique.

$$|\psi\rangle = \sum_{i=0}^{n} a_i |e_i\rangle$$

$$\forall i \in \{1, \dots, n\}, P(mesure(|\psi\rangle) = |e_i\rangle) = |a_i|^2$$

Maintenant que nous avons détaillé les concepts de physique quantique, étudions les concepts d'informatique quantique reposant sur ces derniers.

B. Qubits et informatique probabiliste

1. Bit

En informatique classique, les informations sont représentées par des bits* (« Binary Digits »). Un bit peut prendre les valeurs 0 ou 1. Il peut donc être assimilé à une variable booléenne*. Un bit peut être réalisé matériellement par différents phénomènes physiques, par exemple l'électricité ou la lumière. Dans le cas de l'électricité, la différence de tension entre deux courants électriques peut générer les valeurs 0 et 1.

Plusieurs bits peuvent être regroupés. Un groupe de 8 bits s'appelle un octet*. Un octet peut prendre 2^8 valeurs différentes car chaque bit qui le compose peut prendre 2 valeurs différentes. De même, n bits peuvent prendre 2^n valeurs différentes.

Il est important de noter que les bits servent à représenter une information. Une donnée est codée sur un certain nombre de bits et la valeur de ces bits représente la donnée. Par exemple les huit bits « 0100 0001 » peuvent représenter :

- Le nombre 65 s'ils représente un nombre entier.
- La lettre 'A' majuscule s'ils représentent un caractère de la table ASCII.
- Un niveau de gris de 25,5% de luminosité.
- N'importe quelle autre donnée.

Après ce rappel sur la représentation de l'information en informatique classique, voyons comment l'information est représentée en informatique quantique.

2. Qubit

En informatique quantique, le support de l'information quantique est le qubit, aussi noté qbit ou q-bit, pour « Quantum bit ». Contrairement au bit, le qubit n'a pas de valeur mais un état. S'agissant d'un objet quantique, il peut se trouver dans un état superposé. Le bit prend ses valeurs dans $\{0, 1\}$, le qubit a un état qui est une superposition des états propres $|0\rangle$ et $|1\rangle$ prononcé « ket zéro » et « ket un ».

Il est dès lors possible de clarifier un abus de langage : un qubit ne vaut pas 0 et 1 en même temps. Un qubit se trouve dans un état de superposition de $|0\rangle$ et $|1\rangle$.

$$|qubit\rangle = a|0\rangle + b|1\rangle, a \in \mathbb{C}, b \in \mathbb{C}$$

Pour réaliser un qubit matériellement, il est possible d'utiliser différents phénomènes physiques comme les nanojonctions Josephson [Vion 2003] qui se basent sur des états de spin ou encore les trous d'électrons [Prechtel *et al.* 2016].

Par rapport à ce qui est décrit dans la partie A :

- Le qubit est assimilé à un état quantique $|\psi\rangle$ qui est la superposition des états propres $|0\rangle$ et $|1\rangle$ avec les coefficients complexes a et b.
- Il est possible d'intriquer des qubits et ainsi déterminer l'état d'un qubit connaissant l'état d'un autre.
- Enfin, mesurer l'état d'un qubit réduit son état à un de ses états propres ($|0\rangle$ ou $|1\rangle$) à cause du phénomène du réduction du paquet d'onde. De plus, il est impossible de connaître les valeurs des coefficients a et b.

La mesure de l'état d'un qubit produit un bit :

État mesuré du qubit	**Valeur du bit produit**
$\|\mathbf{0}\rangle$	0
$\|\mathbf{1}\rangle$	1

Tableau 1 : Correspondance entre état mesuré d'un qubit et valeur du bit produit

L'état mesuré d'un qubit dépend des coefficients complexes a et b qui définissent son état. La probabilité de mesurer l'état $|0\rangle$ et d'obtenir la valeur 0 est $|a|^2$ et la probabilité de mesurer l'état $|1\rangle$ et d'obtenir 1 est $|b|^2$. a et b respectent $|a|^2 + |b|^2 = 1$.

Pour représenter l'état d'un bit, il est possible de faire appel à un système à deux états comme un bouton. Nous pouvons considérer que l'état 1 correspond au bouton enfoncé et que l'état 0 correspond au bouton relâché. Pour un qubit, une représentation souvent utilisée est la sphère de Bloch [Bloch 1946].

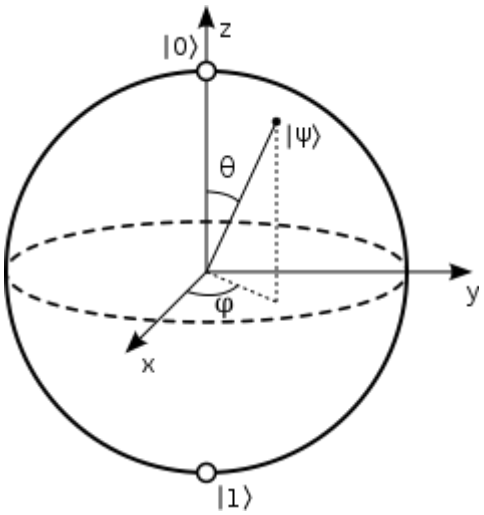


Figure 1 : Sphère de Bloch [4]

Cette sphère donne une représentation géométrique de l'état $|\psi\rangle$ d'un qubit.

$$|\psi\rangle = a|0\rangle + b|1\rangle, a \in \mathbb{C}, b \in \mathbb{C}, |a|^2 + |b|^2 = 1$$

Avec cette représentation, on peut supposer, sans perte de généralité, que a est un réel positif. L'état $|\psi\rangle$ peut donc être réécrit comme suit :

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

De cette manière, un point de cette sphère unité est défini par :

$$\begin{cases} x = \sin\theta \times \cos\phi \\ y = \sin\theta \times \sin\phi \\ z = \cos\theta \end{cases}$$

Un point de la sphère de Bloch représente un état possible d'un qubit. Les états propres sont représentés par $|0\rangle = (0, 0, 1)$ et $|1\rangle = (0, 0, -1)$.

Comme les bits, les qubits peuvent être regrouper en paquets. Un paquet de n bits peut prendre 2^n valeurs. Un paquet de n qubits dispose de 2^n états propres et 2^n coefficients complexes. Par exemple, deux qubits se représentent ainsi :

$$|2\ qubits\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle, a \in \mathbb{C}, b \in \mathbb{C}, c \in \mathbb{C}, d \in \mathbb{C}$$

$$|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$$

De manière générale, tout état d'un registre quantique* de n qubits peut être représenté par :

$$|\text{n } qubits\rangle = \sum_{k=0}^{2^n-1} a_k|k\rangle \ ; \ \forall k \in [\![0, 2^n - 1]\!], a_k \in \mathbb{C} \ ; \ \sum_{k=0}^{2^n-1} |a_k|^2 = 1$$

Une superposition équiprobable d'un registre de n qubits peut être représentée par :

$$|n \ qubits\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle$$

Au sein d'un groupe de qubits, il est possible d'intriquer certains qubits. Connaissant la valeur mesurée d'un qubit, nous pouvons déduire la valeur qui sera mesurée pour le qubit avec lequel il est intriqué. Par exemple, si nous intriquons deux qubits, un état superposé possible de ce couple de qubit est :

$$|2 \ qubits \ intriqués\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Dans cet état, mesurer la valeur d'un des deux qubits nous permet de déduire la valeur que nous mesurerions pour l'autre. Si le premier qubit produit la valeur 0, alors le second produira aussi la valeur 0 car la probabilité que ce couple de qubit soit dans l'état $|01\rangle$ est nulle (le coefficient de cet état est nul donc le carré de son module aussi). Respectivement, si nous obtenons la valeur 1 lors de la mesure du premier qubit, le second vaudra nécessairement 1 car la probabilité que le couple de qubit soit dans l'état $|10\rangle$ est nulle.

3. Limites du qubit par rapport au bit

L'utilisation des phénomènes quantiques sur les qubits ouvre de nouvelles possibilités pour manipuler des données. Cependant, tout comme le bit, le qubit a ses limites. Une première limite du qubit a déjà était présentée, il s'agit du phénomène de réduction du paquet d'onde. Il implique que nous n'avons jamais accès aux coefficients des états propres et que nous ne pouvons mesurer que ces derniers. Cela détruit d'ailleurs l'état de superposition du qubit, empêchant d'effectuer plusieurs mesures successives. A cause de ce phénomène, l'informatique quantique est probabiliste. Nous allons maintenant présenter deux autres limites du qubit : la décohérence* quantique et le théorème de non-clonage.

La décohérence quantique [Zeh 1970] est une théorie visant à expliquer, notamment, la superposition quantique. De plus, le terme « décohérence » désigne la perte des propriétés quantiques d'un système, c'est-à-dire la réduction de son état superposé. D'après cette théorie, un système dans un état de superposition ne peut exister qu'à une échelle microscopique. A plus grande échelle, l'environnement externe interagit avec le système et pour pouvoir interagir, le système doit déterminer son état. La décohérence survient. Le système voit son état réduit à un de ses états propres. On peut considérer que les interactions avec l'environnement agissent comme une observation qui déclenche le phénomène de réduction du paquet d'onde. L'inverse fonctionne aussi, on peut considérer la mesure de l'état d'un système superposé comme une décohérence déclenchée volontairement au bon moment. La limite du qubit liée à la décohérence est justement que cette dernière peut survenir au mauvais moment s'il y a trop d'interactions avec l'environnement externe. A cause de cela, un qubit peut être mesuré par

inadvertance avant la fin du traitement de la donnée qu'il représente. Une analogie possible avec l'informatique classique serait les erreurs douces. Ces erreurs sont provoquées par des particules cosmiques qui traversent les barrettes de mémoire vive des ordinateurs et changent la valeur des bits stockés en mémoire. Ces erreurs sont redoutées dans le domaine du calcul haute performance.

Le théorème de non-clonage a, quant à lui, été démontré [Park 1970]. Il stipule qu'il est impossible de copier l'état superposé d'un système quantique. Cela a pour conséquence qu'une fois dans un état superposé, il est impossible de copier l'état d'un qubit, c'est-à-dire les valeurs des coefficients de ses états propres, vers un autre qubit. En d'autres termes, il n'existe pas d'opérateur d'affectation en informatique quantique.

Maintenant que le concept de qubit et certaines de ses limites intrinsèques ont été expliqués, découvrons certains défis futurs de la physique pour permettre l'évolution de l'informatique quantique.

C. Défis physiques pour l'informatique quantique

Comme énoncé dans l'introduction, en octobre 2019, Google a annoncé avoir atteint la suprématie quantique [Arute *et al.* 2019]. Le terme de « suprématie quantique » a été introduit en 2012 par John Preskill comme suit : « *We therefore hope to hasten the onset of the era of quantum supremacy, when we will be able to perform tasks with controlled quantum systems going beyond what can be achieved with ordinary digital computers.* » [Preskill 2012]. En d'autres termes, la suprématie quantique est atteinte quand un ordinateur quantique est plus performant que n'importe quel ordinateur classique. On remarque alors que la suprématie quantique n'est pas une limite fixe. En effet, elle dépend directement de la puissance de calcul des ordinateurs classiques. Chaque année, l'ordinateur classique le plus puissant de la planète devient plus performant. Il s'agit actuellement de Summit, un supercalculateur conçu par IBM. C'est pourquoi, lorsque Google annonce avoir atteint la suprématie quantique, les performances de leurs processeur quantique Sycamore sont comparées aux performances théoriques de Summit sur un même problème. Le problème utilisé pour la comparaison des performances est celui de l'échantillonnage aléatoire de circuits quantiques (voir partie II), c'est-à-dire donner des sorties possibles (échantillons) d'un circuit quantique quelconque. Il a fallu à Sycamore 200 secondes pour mesurer 1 000 000 échantillons d'un circuit de 53 qubits. Google avait alors annoncé que Summit aurait eu besoin de 10 000 ans pour simuler ce résultat. IBM a, par la suite, annoncé qu'en utilisant l'intégralité de la mémoire de Summit, soit 250 Pio*, ce dernier pourrait arriver au même résultat en 2 jours et demi [Pednault *et al.* 2019]. Notons que ce résultat n'est que théorique car la simulation n'a pas été réalisée. Mais cela ne suffit pas, le processeur quantique reste plus rapide. Il faut 64 Pio de mémoire pour stocker les coefficients complexes des états propres de 53 qubits avec des nombres flottant en simple précision [Pednault *et al.* 2019] (4 octets par nombre flottant, 2 nombres flottants par nombre complexe, 2^{53} nombres complexes pour représenter l'état du système de 53 qubits, $4 \times 2 \times 2^{53} = 2^{56} = 2^6 \times 2^{50} = 64$ Pio). On remarque également que si Sycamore avait eu 2 qubits de plus, alors Summit n'aurait pas eu assez de mémoire pour simuler son fonctionnement aussi rapidement. Il aurait fallu 256 Pio de mémoire pour représenter l'état d'un système de 55 qubits, ce qui est supérieur à 250 Pio.

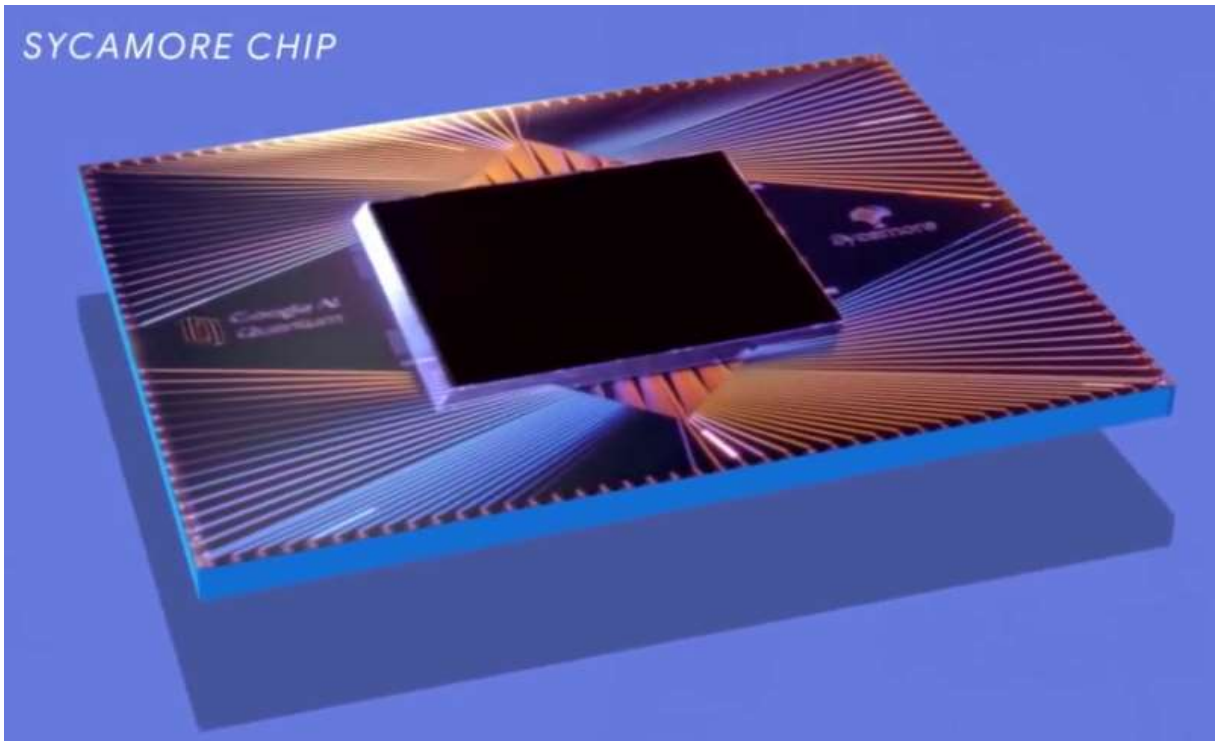


Figure 2 : Processeur quantique Sycamore de Google[2]

Pour réussir à concevoir un tel processeur quantique, Google a relevé certains défis. Nous allons en présenter deux : le nombre de qubits et le temps avant décohérence. A noter qu'aujourd'hui, les termes « processeur quantique » et « ordinateur quantique » sont équivalent.

Le nombre de qubits manipulable par un processeur quantique est une première limite à ses capacités de calcul. En effet, la taille des problèmes qu'il peut résoudre dépend directement du nombre de qubits qu'il peut manipuler. La difficulté à créer des processeurs quantiques provient de la décohérence. Plus un système quantique, comme un ensemble de qubits, est grand, plus il est difficile de limiter les interactions avec l'environnement extérieur. Actuellement, il n'y a aucun processeur quantique universel ayant plus de 53 qubits.

Le temps avant décohérence correspond, comme son nom l'indique, à la durée moyenne avant que ne survienne la décohérence d'un système quantique. Pour avoir le temps de manipuler les qubits du système et ainsi effectuer des calculs, il est nécessaire que cette durée soit suffisamment longue. Cependant, il est difficile d'empêcher les interactions avec l'environnement extérieur sur une longue durée. Pour réduire ces interactions, les processeurs quantiques sont refroidis à une température proche du zéro absolu. Par exemple, Sycamore fonctionne à une température de 20 mK.

En marge de la course à la création de processeurs quantiques toujours plus puissants, d'autres utilisations de l'informatique quantique se développent. Cela a récemment été le cas pour la téléportation quantique, c'est-à-dire le transfert d'information grâce à l'intrication quantique. Un système quantique émetteur est intriqué avec un système quantique récepteur. La distance n'a pas d'impact sur l'intrication. L'émetteur utilise le lien qu'est l'intrication quantique pour transférer une information au récepteur. Après le transfert, l'information émise n'est plus accessible à l'émetteur à cause du théorème de non-clonage, d'où le nom de

[2] https://commons.wikimedia.org/wiki/File:Google_Sycamore_Chip_002.png

téléportation quantique. Mais c'est également grâce au théorème de non-clonage qu'une telle communication est impossible à espionner. Si une autre personne observe la communication, alors la décohérence survient et empêche le transfert d'informations. Le premier transfert d'information entre deux processeurs quantiques, non reliés entre eux, a été réalisé puis publié en décembre 2019 [Llewellyn *et al.* 2019]. Le défi est maintenant d'augmenter la distance entre émetteur et récepteur.

Nous venons de présenter les défis physiques à relever pour fabriquer des processeurs quantiques plus puissants alors voyons maintenant comment utiliser ces processeurs pour effectuer des calculs et exécuter des algorithmes.

II. Algorithmes quantiques

A. Portes et circuits quantiques

1. Portes quantiques

En informatique classique, les bits sont manipulés par des portes logiques. Ces portes permettent d'appliquer des opérations de logique booléennes sur les bits telles que « ET », « OU », « OU EXCLUSIF », « NON » ... Elles prennent des bits en entrée et produisent un autre bit en sortie (les bits en entrée ne disparaissent pas). Chaque porte logique est définie par sa table de vérité qui donne la valeur de sortie en fonction des valeurs des entrées.

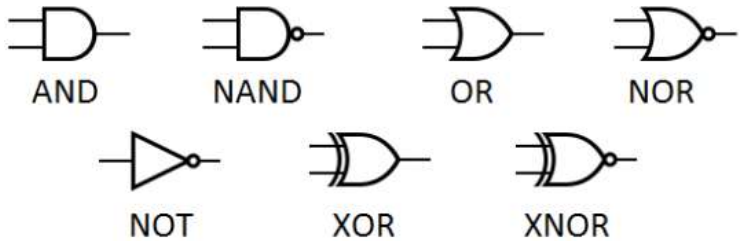


Figure 3 : Portes logiques, notations internationales [5]

X	Y	Sortie
0	0	0
0	1	0
1	0	0
1	1	1

Tableau 2 : Table de vérité de la porte « ET »

En informatique quantique, les qubits sont manipulés par des portes quantiques [Barenco *et al.* 1995]. Ces portes permettent d'appliquer des opérations sur les qubits et ainsi modifier l'état superposé d'un système quantique sans provoquer la décohérence.

Rappelons ici qu'un système de n qubits peut être représenté par un vecteur colonne de 2^n nombres complexes.

$$|n\ qubits\rangle = \sum_{k=0}^{2^n-1} a_k |k\rangle$$

Contrairement aux portes logiques qui sont définies par leur table de vérité, chaque porte quantique* est définie par une matrice unitaire d'ordre 2^n avec n le nombre de qubits manipulés par la porte quantique. Une matrice unitaire est une matrice carrée U à coefficients complexes telle que $UU^\dagger = U^\dagger U = I$ avec $U^\dagger$ la matrice adjointe (transposée de la matrice conjuguée) de U. Pour appliquer une porte quantique à un système de qubits, il faut multiplier la matrice de cette porte avec le vecteur des coefficients complexes du système.

$$|n\ qubits'\rangle = U|n\ qubits\rangle$$

$$|n\ qubits'\rangle = \sum_{k=0}^{2^n-1} a'_k |k\rangle$$

On remarque alors qu'appliquer une porte quantique à un système de qubits modifie son état. Une porte quantique ne crée pas sa sortie mais modifie son entrée. On pouvait voir une porte logique comme une fonction qui calcule et retourne un résultat et on peut voir une porte quantique comme une procédure qui effectue un traitement et modifie ses entrées. D'autre part, les portes quantiques sont réversibles. L'état qu'avait le système avant sa modification peut être retrouvé en appliquant la porte quantique représentée par la matrice adjointe $U^\dagger$ ($U^{-1} = U^\dagger$).

Quelques portes quantiques :

- La porte de Hadamard permet de transformer l'état d'un qubit pour passer d'un état propre à un état superposé équiprobablement. Par exemple, la porte de Hadamard transforme un qubit dans l'état $|0\rangle$ en qubits dans l'état $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$.
- La porte Pauli-X correspond à la porte logique « NON ». Elle transforme l'état d'un qubit dans l'état $|0\rangle$ en l'état $|1\rangle$ et vice versa.
- La porte SWAP agit sur deux qubits et permet d'échanger leurs états. Ainsi, un système de 2 qubits dans l'état $|01\rangle$ passe dans l'état $|10\rangle$ et réciproquement.
- La porte CNOT est une version conditionnelle de la porte Pauli-X. Elle inverse l'état d'un qubit si un premier qubit est dans l'état $|1\rangle$. Cette porte permet d'intriquer des qubits car elle lie l'état d'un qubit à celui d'un autre. Plus généralement, toute porte quantique U peut être contrôlée (rendue conditionnelle) par un qubit.
- La porte de Toffoli [Toffoli 1980] est semblable à la porte CNOT mais agit sur trois qubits et inverse l'état du dernier si les deux premiers sont dans l'état $|1\rangle$.

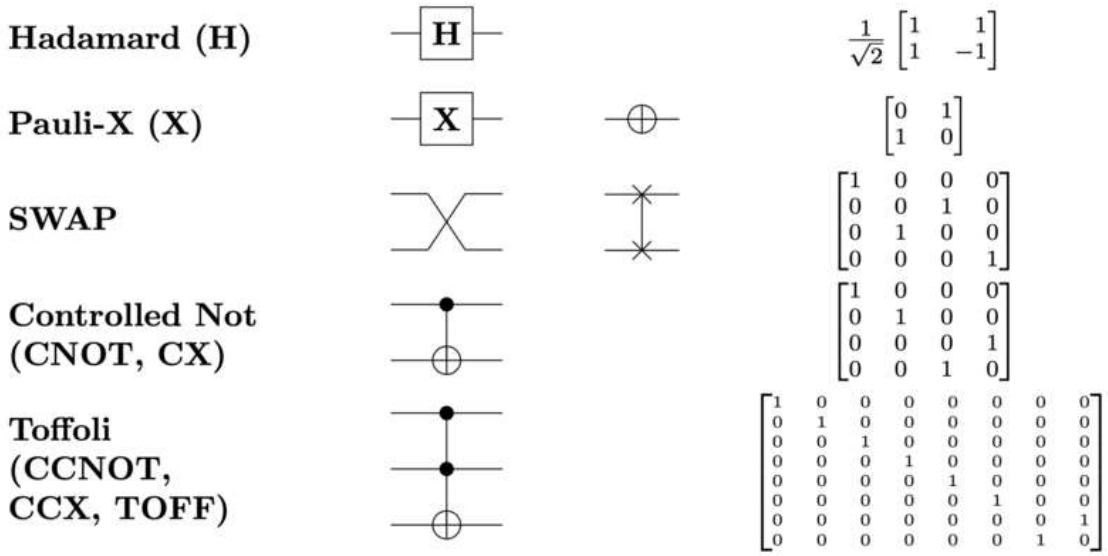


Figure 4 : Schémas et matrices de quelques portes quantiques

Maintenant que nous avons pris connaissance de quelques portes quantiques, voyons comment les utiliser pour créer des circuits quantiques.

2. Circuits quantiques

Pour pouvoir réaliser des calculs et implémenter des algorithmes, en informatique classique, il faut relier entre elles des portes logiques afin de créer des circuits logiques. Un circuit logique représente une fonction booléenne qui prend des bits en entrée et produit d'autres bits en sortie. En informatique quantique, les qubits ne se déplacent pas pour traverser des portes. Ce sont les portes quantiques qui sont appliquées aux systèmes de qubits. Une porte quantique ne produisant aucune sortie mais modifiant ses entrées, un circuit quantique a toujours le même nombre de sorties que d'entrées. Pour représenter un circuit quantique, il est possible d'utiliser une « partition », c'est-à-dire une représentation des portes quantiques appliquées aux qubits d'un système au cours du temps.

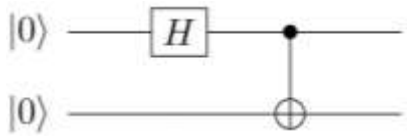


Figure 5 : Circuit d'intrication de deux qubits

Par exemple, cette partition décrit un circuit quantique agissant sur un registre de deux qubits (système de deux qubits) placés initialement dans l'état $|00\rangle$. Chaque qubit est dans l'état $|0\rangle$. Le circuit commence par appliquer la porte de Hadamard au premier qubit, ce qui transforme son état en un état superposé : $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. L'état du registre est alors $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$ car le second qubit est toujours dans l'état $|0\rangle$. Enfin, la porte CNOT est appliqué au registre en utilisant le premier qubit comme qubit de contrôle et le second qubit comme qubit susceptible de voir son état être changé. Comme décrit dans la partie précédente, la porte CNOT inverse l'état du second qubit si le premier est dans l'état $|1\rangle$. Ainsi, l'état du registre de deux qubits devient alors $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. Cet état est celui présenté dans la partie I.B.2 et correspond à un état de deux qubits intriqués.

Sur la partition, les qubits semblent indépendants. Ce n'est pas le cas. Il ne faut pas oublier qu'ils font partie d'un seul registre et doivent être manipulés en tant que tels. Par conséquent, la porte de Hadamard ne peut pas être appliquée au registre de deux qubits car elle ne prend en paramètre qu'un seul qubit. Pour pallier cela, le produit de Kronecker [6] est utilisé. Ici, la porte appliquée au registre est $H \otimes I_2$ ou I_2 est la matrice identité d'ordre 2 et $\otimes$ représente le produit de Kronecker. Ainsi, c'est bien une matrice d'ordre 4 qui est appliquée et qui met en superposition le premier qubit du registre. La porte CNOT, quant à elle, prend déjà deux qubits en entrée, elle n'a pas besoin d'être modifiée.

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}, I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, H \otimes I_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} I_2 & \frac{1}{\sqrt{2}} I_2 \\ \frac{1}{\sqrt{2}} I_2 & -\frac{1}{\sqrt{2}} I_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

De plus, puisque le circuit est une suite de portes quantiques, il est également possible de le représenter comme une matrice qui est le produit des matrices des portes utilisées par le circuit. Ici, ce circuit correspond à la matrice : $CNOT \times H \otimes I_2$. Ainsi, un circuit quantique de portes quantiques peut être représenté par une matrice. Cela en fait un bloc réutilisable : une « procédure » qui modifie ses entrées. Rappelons qu'il ne peut pas y avoir de fonctions quantiques mais seulement des procédures à cause du théorème de non-clonage qui empêche d'affecter un potentiel résultat à des qubits.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, CNOT \times H \otimes I_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

De manière générale, dans un circuit quantique, le produit de Kronecker est utilisé pour appliquer des portes quantiques en parallèles sur différents qubits. Dans le circuit précédent, on peut considérer que nous appliquons H au premier qubit et I_2 au second qubit en même temps. Remarquons que le produit de Kronecker n'est pas commutatif et que l'ordre des qubits à manipuler détermine l'ordre des matrices dans le produit de Kronecker. Cet ordre est le même que celui des qubits lors de leur initialisation, par exemple avec un registre de deux qubits initialisé ainsi $|1\rangle \otimes |0\rangle = |10\rangle$, si nous voulons appliquer la porte de Hadamard au qubit initialisé à $|0\rangle$ sans appliquer de porte à l'autre, la porte à appliquer au registre est $I_2 \otimes H$. Il est possible d'avoir besoin de mettre un registre de n qubits en superposition quantique. Pour cela, il est possible d'appliquer au registre la porte de Hadamard H_n aussi notée $H^{\otimes n}$.

$$\begin{cases} H_1 = H \\ H_n = H \otimes H_{n-1} \end{cases}$$

Le produit matriciel, quant à lui, est utilisé pour appliquer des portes quantiques séquentiellement sur les qubits du registre. Nous remarquons une nouvelle fois que ce produit n'est pas commutatif et que l'ordre des portes détermine l'ordre des matrices dans le produit. Ajouter une porte à la fin de la partition revient à ajouter une matrice à gauche du produit.

Un autre exemple de circuit quantique manipulant un registre quantique de 5 qubits est donné en annexe 1. Ce circuit sert à additionner deux qubits. La matrice qui représente ce circuit est donnée en annexe 2. Elle a été calculée et utilisée par un programme C++ développé lors de ce projet (voir https://github.com/ThomasCluzel/zz3-quantum-computing/tree/master/zz3-projet-simulateur-quantique) avec la bibliothèque uBLAS de Boost.

Certaines procédures quantiques récurrentes ont même été nommées, c'est le cas, par exemple, de la transformée de Fourier quantique (QFT) [Coppersmith 1994] qui peut être appliquée à n qubits. On peut remarquer que comme de nombreux circuit, elle commence par appliquer la porte de Hadamard aux qubits pour obtenir un état superposé avant de les manipuler avec des portes de changement de phase contrôlées $CR_k = \begin{bmatrix} I_2 & 0 \\ 0 & R_k \end{bmatrix}, R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2i\pi}{2^k}} \end{bmatrix}$.

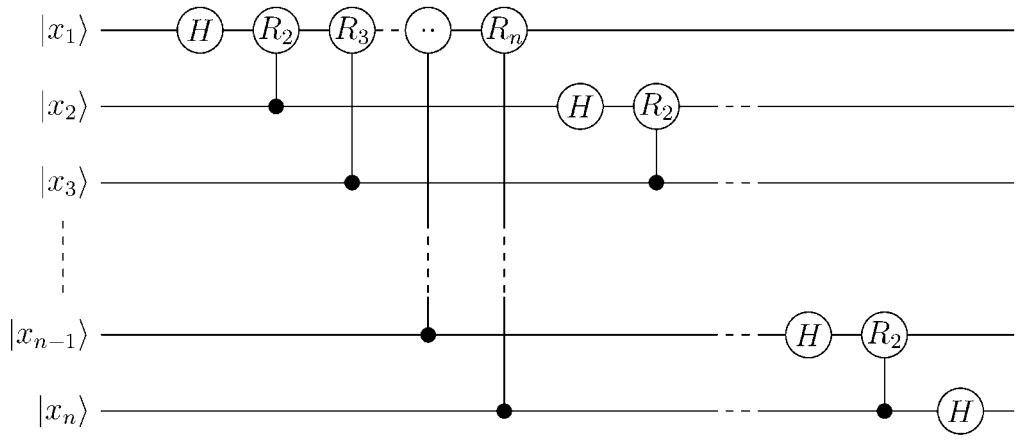


Figure 6 : Transformée de Fourier quantique [7]

En plus des portes, il y a un dernier élément que l'on peut rencontrer dans un circuit quantique : la mesure. La mesure est l'opération qui réduit l'état superposé d'un qubit à un de ses états propres $|0\rangle$ ou $|1\rangle$ par le principe de réduction du paquet d'ondes. Mesurer permet d'extraire une valeur binaire 0 ou 1 d'un qubit. Contrairement aux portes quantiques, la mesure est irréversible.

Figure 7 : Symbole de la mesure d'un qubit

Nous venons de présenter ce que sont les circuits quantiques, prenons du recul pour nous intéresser aux algorithmes quantiques.

B. Analyse détaillée de l'algorithme de Grover

Dans cette partie, nous allons analyser un algorithme quantique : l'algorithme de Grover [Grover 1996]. Il s'agit d'un algorithme de recherche dans un ensemble non-ordonné. Il permet de trouver un élément x d'un ensemble E vérifiant une propriété P. Pour rechercher un élément dans un ensemble quelconque de taille N, il faut, avec un ordinateur classique, tester un par un les N éléments de l'ensemble. L'algorithme de Grover, quant à lui, utilise les propriétés de l'informatique quantique pour accélérer cette recherche. Cependant, comme beaucoup d'algorithmes quantiques, l'algorithme de Grover est probabiliste. Par conséquent, il ne fournit pas toujours le bon résultat.

1. Présentation de l'algorithme

Commençons par décrire le principe de cet algorithme. Le but de l'algorithme de Grover est de placer un registre de qubits représentant l'ensemble de recherche dans un état superposé puis d'augmenter la probabilité de mesurer l'élément recherché. Enfin, une fois cette probabilité suffisamment élevée, mesurer le registre quantique permet de récupérer la valeur de l'élément recherché, avec une probabilité suffisamment élevée.

L'algorithme de Grover sur un registre de n qubits est le suivant :

1. Initialiser le registre dans l'état $|0\rangle^{\otimes n}$.
2. Appliquer une porte de Hadamard à chaque qubit du registre pour le placer dans l'état superposé suivant :

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$$

3. Répéter $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ fois :
   i. Appliquer l'oracle U_f pour modifier la phase du ou des éléments recherchés.
   $$|\psi\rangle = U_f|\psi\rangle$$
   ii. Appliquer la réflexion U_s pour augmenter la probabilité de mesurer le ou les éléments recherchés.
   $$|\psi\rangle = U_s|\psi\rangle$$
   $$U_s = H^{\otimes n}(2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I_n)H^{\otimes n}$$
4. Mesurer le registre.

Comme beaucoup d'algorithmes quantiques, l'algorithme de Grover commence par appliquer une transformée de Hadamard sur le registre de qubits. On remarque qu'après l'étape 2, toutes les valeurs possibles du registre ont la même probabilité d'être mesurées, à savoir $\frac{1}{2^n}$.

Après l'initialisation, il faut appliquer l'oracle U_f. Il s'agit d'une procédure quantique qui dépend de l'élément recherché. Par conséquent, pour chaque ensemble E et chaque propriété P, il faut concevoir une transformation U_f spécifique. Cette transformation doit inverser la phase des états propres recherchés, c'est-à-dire changer le signe des coefficients complexes facteurs de ces états. Pour cela la porte Pauli-Z, ou des circuits plus évolués utilisant la porte Pauli-Z, sont utilisés. De manière générale, U_f peut être représenté par une matrice diagonale dont les coefficients valent -1 pour les états recherchés et 1 pour les autres. Par exemple, avec $n = 3$ qubits, si l'état propre recherché est $|101\rangle$, alors U_f est comme suit :

$$U_f = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & Z & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \; avec \; I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \; et \; Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Il existe une variante de cette transformation U_f qui utilise un qubit auxiliaire, c'est-à-dire un qubit ne servant qu'à cette transformation et n'ayant aucune autre utilité dans l'algorithme. Cette variante consiste à remplacer les changements de phases par des inversions de l'état du qubit auxiliaire. Ainsi, au lieu de concevoir un circuit avec des portes Pauli-Z, des portes CNOT sont utilisées.

Expliquons maintenant la seconde transformation de cet algorithme : U_s. U_s sert à amplifier la probabilité de mesurer l'état recherché. La précédente transformation U_f a seulement changé la phase des éléments recherchés, ce qui n'a modifié la probabilité d'aucun

état propre, car le carré du module de l'opposée est égale au carré du module ($\forall a \in \mathbb{C}, |-a|^2 = |a|^2$). Le but de la transformation U_s est d'augmenter le module du coefficient des éléments dont la phase a été changée. Pour cela, cette transformation peut être séparée en trois parties comme suit :

$$U_s = H^{\otimes n}(2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I_n)H^{\otimes n}$$

$$2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I_n = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

La transformation U_s applique d'abord une porte de Hadamard à tous les qubits du registre, puis une matrice diagonale, dont le premier coefficient vaut 1 et les autres valent -1, et applique de nouveau une porte de Hadamard à tous les qubits du registre. L'application successive des transformations U_f puis U_s sur le registre quantique permet d'augmenter la probabilité de mesurer l'élément recherché. C'est pourquoi répéter cette opération plusieurs fois augmente encore plus cette probabilité. Le nombre optimal de répétitions est $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ [Boyer *et al.* 1996]. Ainsi, la complexité de l'algorithme de Grover pour n qubits est $O(\sqrt{2^n})$ et bien qu'il s'agisse d'une accélération quadratique par rapport à l'algorithme classique, on remarque que cette complexité reste exponentielle.

2. Circuit quantique de l'algorithme de Grover

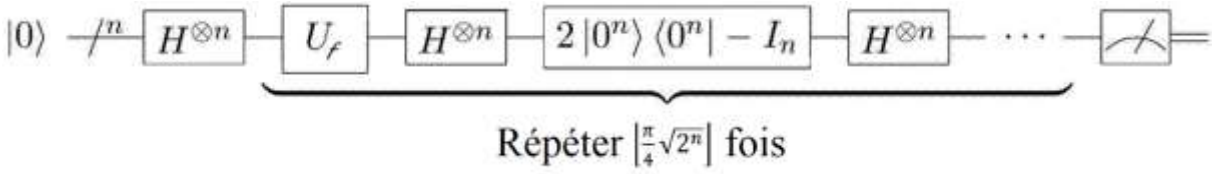


Figure 8 : Circuit quantique de l'algorithme de Grover

Le circuit quantique réalisant l'algorithme de Grover est présenté Figure 8. Nous pouvons observer les différentes étapes de l'algorithme. Il commence par appliquer la porte de Hadamard à tous les qubits du registre. Puis, il répète les transformations U_f et U_s. Enfin, le registre est mesuré pour obtenir la sortie de l'algorithme sur n bits classiques.

$$résultat = mesure\left(\left(U_s U_f\right)^{\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor} H^{\otimes n}|0\rangle^{\otimes n}\right)$$

Ce circuit décrit bien l'algorithme de Grover mais ne représente pas une implémentation de ce dernier pour plusieurs raisons. La première est que le nombre de qubits n est inconnu. Donc, la taille de l'ensemble de recherche n'est pas connue. La seconde est que l'oracle U_f n'est pas implémenté. S'agissant d'une représentation générale, il est normal que nous ne sachions pas encore quel élément rechercher. Enfin, une troisième raison est que la porte $2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I_n$ n'est pas une porte quantique connue.

Supposons que nous cherchions dans un registre de 3 qubits l'élément $|101\rangle$. Le premier problème est déjà résolu. Il faut maintenant concevoir la porte U_f. Pour cela, nous disposons de 2 options, utiliser des portes Pauli-Z ou des portes CNOT et un qubit auxiliaire. Les circuits de ces deux options sont représentés Figure 9.

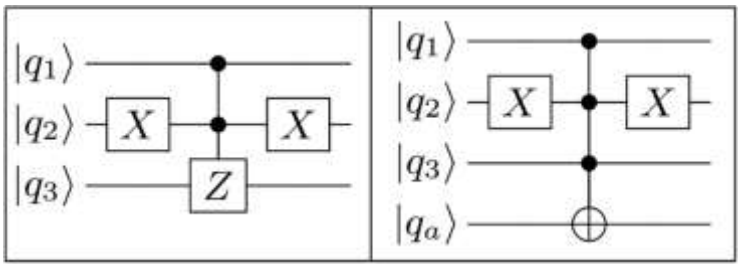


Figure 9 : Exemple d'oracle U_f pour la recherche de l'état $|101\rangle$ sur 3 qubits avec une porte CCZ (à gauche) et avec une porte CCCNOT et un qubit auxiliaire (à droite) [Figgatt et al. 2017]

En choisissant la seconde option, il faut que le qubit auxiliaire soit initialement dans l'état $|1\rangle$ avant de lui appliquer la porte de Hadamard. Pour cela, il faut lui appliquer la porte Pauli-X avant la porte de Hadamard.

Un nouveau problème survient alors. Il faut pouvoir implémenter les portes CCZ et CCCNOT. Une façon d'implémenter des portes contrôlées avec un nombre quelconque de qubits de contrôle consiste à utiliser des portes de Toffoli et une porte quelconque contrôlée. Un exemple d'utilisation de cette méthode est donné Figure 10 pour 5 qubits de contrôles.

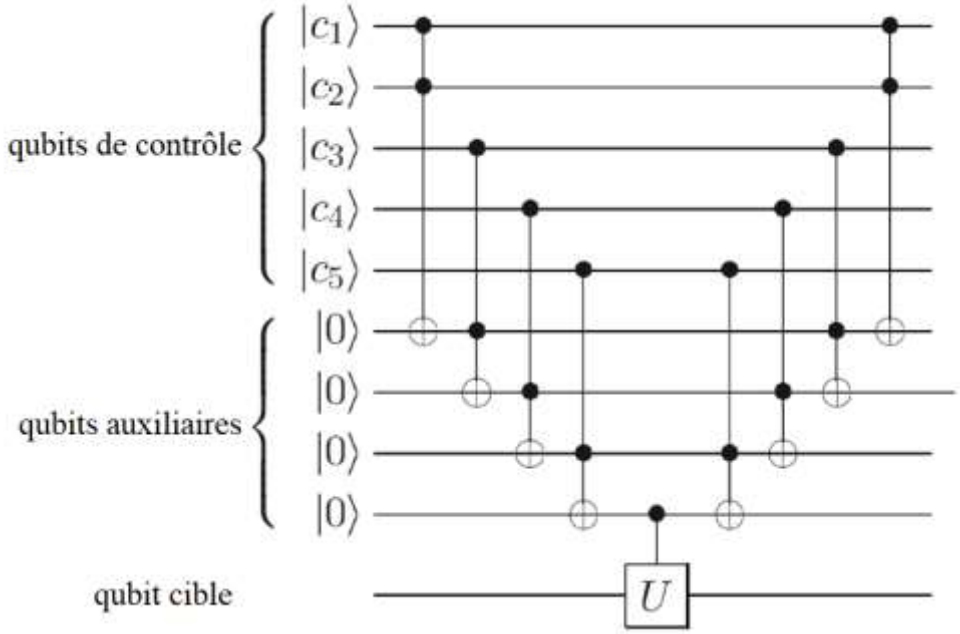


Figure 10 : Implémentation de la porte C^nU avec n=5 [Nielsen Chuang 2000]

Avec cette méthode, nous sommes en mesure d'implémenter n'importe quelle porte $C^n(X)$. Cette solution fonctionne quel que soit le nombre de qubits de contrôle. En ce qui concerne notre exemple, nous pouvons d'ores et déjà implémenter la porte CCCNOT et implémenter la deuxième version de l'oracle U_f. Concernant la première version de l'oracle U_f utilisant une porte Pauli-Z, nous devons encore implémenter la porte CZ. Il se trouve que :

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} = (I \otimes H)(CNOT)(I \otimes H)$$

Nous pouvons ainsi implémenter les deux versions de l'oracle U_f quel que soit le nombre de qubits. Toutefois, dans notre exemple, pour rechercher l'état $|101\rangle$, des optimisations sont possibles. Nous pouvons par exemple implémenter CCZ comme ceci : $CCZ = (I \otimes I \otimes H)(Toffoli)(I \otimes I \otimes H)$.

De manière générale, l'oracle U_f utilisant une porte Pauli-Z à l'avantage, par rapport à celui utilisant une porte CNOT, de nécessiter un qubit auxiliaire de moins.

Il faut encore implémenter la porte représentée par la matrice $2|0\rangle^{\otimes 3}\langle 0|^{\otimes 3} - I_3$. Cependant, il est plus facile d'implémenter une porte représentée par l'opposée de cette matrice. Elle permet d'obtenir le même résultat et peut être implémentée par $X^{\otimes 3}(CCZ)X^{\otimes 3} = I_3 - 2|0\rangle^{\otimes 3}\langle 0|^{\otimes 3}$.

3. Illustration et utilisation de l'algorithme de Grover

Nous disposons maintenant d'un circuit quantique implémentant l'algorithme de Grover pour rechercher l'état $|101\rangle$ représenté par 3 qubits. La première itération de l'algorithme est illustrée Figure 11. Par ailleurs, cette illustration rappelle que l'algorithme de Grover est probabiliste. Lorsque l'on répète jusqu'à $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ itérations, la probabilité de mesurer l'élément recherché augmente, et pour exactement $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ itérations, elle est supérieure à $1 - 2^{-n}$, mais rien ne garantit que la mesure produise bien la valeur recherchée. C'est pourquoi l'algorithme de Grover n'est pas déterministe. Pour cela, après la mesure, il est nécessaire de vérifier que la valeur obtenue vérifie bien la propriété P. Si ce n'est pas le cas l'algorithme doit être réexécuté depuis le début. Rappelons qu'il est impossible, après une mesure, de retrouver l'état superposé qui l'a engendrée à cause de la réduction du paquet d'onde.

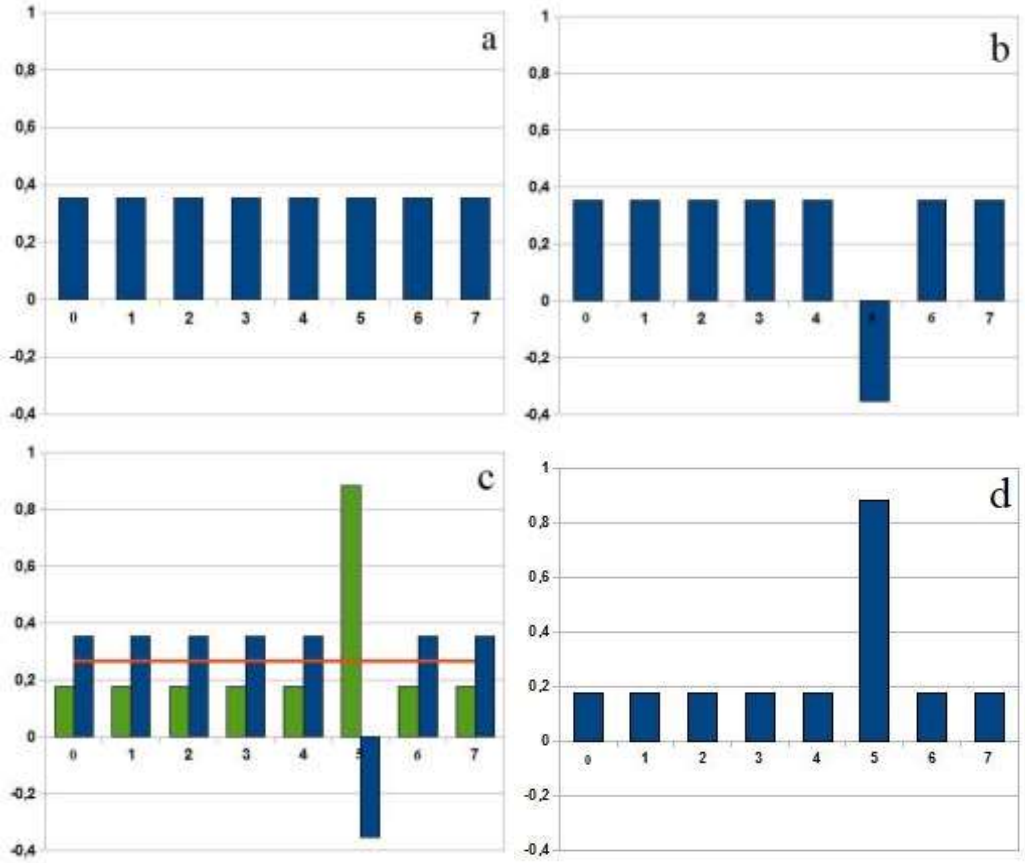


Figure 11 : Illustration de l'état du registre quantique lors des étapes de l'algorithme de Grover : après superposition (a), après application de U_f (b), lors de l'application de U_s (c) et après l'application de U_s (d)

L'algorithme de Grover est un algorithme de recherche. Bien qu'il ait été originellement conçu pour rechercher des entrées dans des bases de données non-triées, il ne s'agit peut-être pas de sa principale utilité. D'abord, parce qu'une base de données quantique ou capable d'interagir avec un ordinateur quantique n'existe pas encore. Ensuite parce qu'il s'agit d'un algorithme de recherche utilisable pour rechercher n'importe quelle valeur. Dans notre exemple, nous avons recherché un nombre entier dans un sous ensemble de $\mathbb{N}$.

Un premier domaine d'application de l'algorithme de Grover pourrait être la résolution de problème de recherche NP-Complets. S'agissant de problème de recherche, l'algorithme de Grover peut être utilisé. De plus, étant donnée une solution potentielle à un problème NP-Complet, il est possible de vérifier cette solution en temps polynomial. Cela permet de vérifier le résultat produit par l'algorithme de Grover sans augmenter la complexité générale du procédé.

Un autre domaine d'application de l'algorithme de Grover est l'inversion de fonctions. Bien qu'il soit impossible d'implémenter des fonctions classiques avec n entrées et m sorties sur un ordinateur quantique, il est possible d'implémenter un circuit avec $n + m$ entrées et $n + m$ sorties [Bennett 1973].

$$(y_1, \dots, y_m) = f(x_1, \dots, x_n)$$

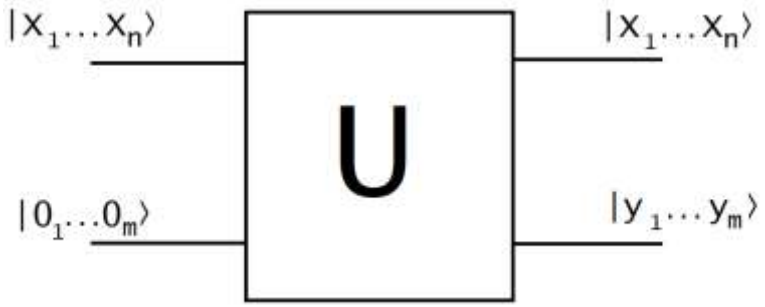


Figure 12 : Utilisation d'une fonction sur un ordinateur quantique

La Figure 12 montre comment utiliser une fonction classique f sur un ordinateur quantique en concevant un circuit quantique U. A partir de ce circuit U, il est possible de concevoir un oracle U_f qui marque les antécédents d'une image par la fonction f. L'algorithme de Grover peut ainsi être utilisé pour rechercher le ou les antécédents $|x_1 \dots x_n\rangle$ de n'importe quelle image $|y_1 \dots y_m\rangle$ par une fonction f. Cette utilisation de l'algorithme de Grover a été expérimentée lors de ce projet. Un exemple recherchant les antécédents de l'addition de deux bits avec une retenue (circuit disponible en annexe 1) est disponible sur le dépôt GitHub* suivant https://github.com/ThomasCluzel/zz3-quantum-computing/tree/master/zz3-projet-ibm-q-experience/05-Grover_function_inversion. N'importe quelle fonction pourrait être inversée car toute fonction calculable par un ordinateur classique peut être implémentée par un circuit quantique [Deutsch Penrose 1985]. Ainsi, il serait par exemple possible de trouver des antécédents d'empreintes issues de fonctions de hachage. Il deviendrait alors capital de garder secrètes les chaînes de caractères utilisées comme sel.

Nous venons de présenter en détail l'algorithme de Grover et d'en donner de possibles utilisations. Listons maintenant quelques autres domaines où l'informatique quantique peut être utilisée.

C. Domaines d'application de l'informatique quantique

Après avoir présenté l'algorithme de Grover, énumérons maintenant quelques domaines auxquels l'informatique quantique peut être appliquée.

Le premier domaine auquel l'informatique quantique peut être appliquée est celui de la cryptanalyse. La cryptanalyse vise à décrypter des messages chiffrés sans en connaitre la clé de déchiffrement. Il existe de nombreux algorithmes de chiffrement qui se répartissent en deux catégories : les algorithmes de chiffrement symétrique et les algorithmes de chiffrement asymétrique. La sécurité des algorithmes de chiffrement asymétrique repose sur un problème mathématique difficile à résoudre pour un ordinateur classique pour assurer la confidentialité de la clé privée. Par exemple, l'algorithme RSA [Rivest *et al.* 1978] repose sur la factorisation de nombres entiers. Soit $N = ab$ avec a et b premiers, il n'existe aucun algorithme polynomial pour trouver a et b connaissant N. Cependant, pour un ordinateur quantique, la factorisation n'est pas aussi difficile. En effet, l'algorithme de Shor [Shor 1994] permet de résoudre ce problème avec une complexité de $O\left(\left(log_2(N)\right)^3\right) = O(n^3)$ avec n le nombre de bits nécessaires pour représenter N. Ainsi, l'algorithme de Shor est un algorithme quantique de complexité polynomiale qui permet de résoudre le problème de factorisation de nombres entiers. Par conséquent, il permet de décrypter les messages chiffrés avec des algorithmes de chiffrement basés sur le problème de factorisation comme RSA.

Un autre domaine proche de la cryptanalyse est celui de la cryptographie. Contrairement à la cryptanalyse, la cryptographie vise à concevoir des algorithmes de chiffrement et des protocoles de communication sécurisés. En plus de pouvoir servir à décrypter des messages, l'informatique quantique peut être utilisée pour sécuriser des communications. Par exemple, le protocole BB84 [Bennett Brassard 1984] permet un échange de clés de chiffrement sécurisé grâce aux propriétés de l'informatique quantique, comme le théorème de non-clonage qui empêche l'écoute du canal de communication.

Après avoir parlé d'échange de clés, nous pouvons passer au domaine de la génération de nombres aléatoires. Il existe 3 grandes catégories de nombres aléatoires en informatique : les nombres pseudo-aléatoires, les nombres quasi-aléatoires et les nombres vraiment aléatoires. Pour générer des nombres pseudo-aléatoires, il faut utiliser des générateurs de nombres pseudo-aléatoires comme Mersenne Twister [Matsumoto Nishimura 1998] ou MRG32k3a [L'Ecuyer *et al.* 2002]. De même, pour générer des nombres quasi-aléatoires, il faut utiliser des générateurs de nombres quasi-aléatoire comme celui de Van Der Corput [Van der Corput 1935]. Les générateurs de nombres pseudo-aléatoires et quasi-aléatoires sont reproductibles. Ce n'est pas le cas des générateurs de nombres vraiment aléatoires qui utilisent des phénomènes physiques pour générer des séquences de nombres. Par exemple, la société suisse ID Quantique commercialise le dispositif « Quantis » [8] sous la forme de cartes filles ou de périphériques USB. Ce générateur de nombres aléatoires matériel génère des nombres vraiment aléatoires en utilisant les propriétés de l'informatique quantique. Toutefois, s'agissant de séquences de nombres vraiment aléatoires, rien ne garantit la qualité statistique de la séquence. C'est pourquoi les séquences générées par ces dispositifs peuvent être biaisées.

Figure 13 : Quantis en tant que périphérique USB (à gauche) et en tant que carte fille (à droite) [8]

Dans le domaine de l'algèbre linéaire, l'algorithme quantique HHL [Harrow *et al.* 2009] permet de résoudre des systèmes d'équations linéaires avec une complexité de $O(\log(n)\kappa^2)$ où n est le nombre de variables et κ est le conditionnement du système d'équations. Cet algorithme peut être utilisé par n'importe quelle application ayant besoin de résoudre des systèmes d'équations linéaires.

Un domaine utilisant l'algèbre linéaire est celui de l'apprentissage machine. L'informatique quantique s'applique aussi à ce dernier. Il existe des versions quantiques d'algorithmes d'apprentissage machine classiques, comme les machines à vecteurs de support quantique [Havlicek *et al.* 2018], les K-moyennes quantiques, la classification hiérarchique quantique ou l'analyse en composantes principales quantique [Wittek 2016].

Un domaine vaste auquel l'informatique quantique peut être appliquée est celui de la simulation. L'informatique quantique peut être utilisée pour simuler des molécules [Aspuru-Guzik *et al.* 2005] ou même des particules [Wecker *et al.* 2015].

Tout comme en informatique classique, les problèmes se répartissent en différentes catégories de complexité en fonction des algorithmes quantiques connus pour les résoudre. De manière analogue aux classes P et NP, il existe deux classes de problèmes importantes en informatique quantique. Tout d'abord, la classe BQP (« Bounded error Quantum Polynomial time ») est l'analogue de la classe P. Elle contient les problèmes qui peuvent être résolus en temps polynomial par un ordinateur quantique avec un taux d'erreur* d'au plus $\frac{1}{3}$. Par exemple, l'algorithme de Shor appartient à la classe BQP. Ensuite, la classe QMA (« Quantum Merlin Arthur ») est l'analogue de la classe NP. Elle contient les problèmes dont une solution peut être vérifiée en temps polynomial par un ordinateur quantique avec une probabilité supérieure à $\frac{2}{3}$. Ces classes sont représentées Figure 14.

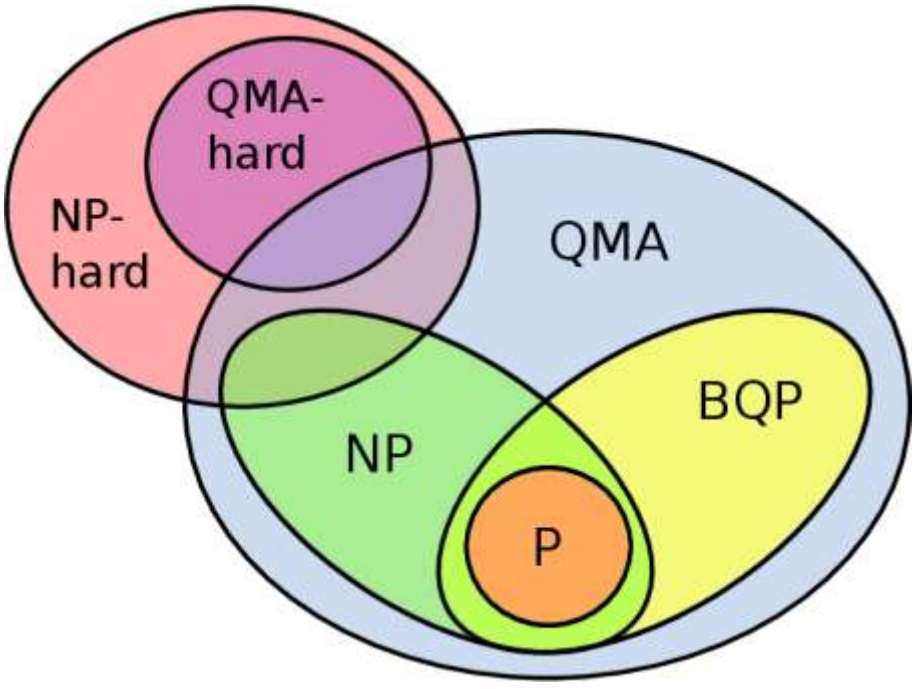


Figure 14 : Représentation des classes de complexité des problèmes [Kassal et al. *2011]*

Nous venons de donner quelques exemples d'algorithmes quantiques dans différents domaines, mais de nombreux autres existent. La liste complète des algorithmes quantiques connus aujourd'hui est disponible sur le zoo des algorithmes quantiques [9]. Cependant, pour pouvoir être utilisés à leur plein potentiel beaucoup de ces algorithmes nécessitent de pouvoir manipuler une grande quantité de qubits et de pouvoir leur appliquer beaucoup de portes quantiques sans provoquer la décohérence. Voyons maintenant quels outils sont aujourd'hui disponibles en informatique quantique.

III. Matériel informatique quantique actuel

A. Acteurs de l'informatique quantique

1. Contraintes des ordinateurs quantiques

En informatique quantique, comme en informatique classique, les domaines du matériel et du logiciel sont distincts. Il y a d'une part, le domaine du logiciel, qui étudie les circuits quantiques de portes quantiques manipulant des qubits comme décrit dans la partie II, et d'autre part, le domaine du matériel, qui vise à construire des ordinateurs quantiques pouvant exécuter ces circuits quantiques.

Aujourd'hui, aucun ordinateur quantique ne fonctionne seul. Un ordinateur quantique est utilisé comme un coprocesseur d'un ordinateur classique. Un ordinateur classique est nécessaire pour interagir avec un ordinateur quantique, lui envoyer un circuit à exécuter et récupérer le résultat. Pour que cela soit possible, lors de la création d'un ordinateur quantique, il faut respecter les critères de DiVincenzo pour le calcul quantique [DiVincenzo 2000] :

1. « *A scalable physical system with well characterized qubits* » : le support physique utilisé pour implémenter les qubits de l'ordinateur quantique doit avoir des propriétés physiques correspondant à celles du modèle du qubit décrit dans la partie I. De plus, ce support doit pouvoir être mis à l'échelle, ce qui veut dire qu'il ne doit pas y avoir de limite physique au nombre de qubits de ce type manipulables par l'ordinateur quantique.
2. « *The ability to initialize the state of the qubits to a simple fiducial state, such as* $|000 \ldots\rangle$ » : les qubits doivent pouvoir être initialisés dans un état avant le début du programme, par exemple, on doit pouvoir initialiser tous les qubits dans l'état $|0\rangle$.
3. « *Long relevant decoherence times, much longer than the gate operation time* » : le temps de cohérence de l'ordinateur doit être suffisamment long pour pouvoir appliquer toutes les portes quantiques des circuits demandés sans que la décohérence ne survienne avant la mesure.
4. « *A "universal" set of quantum gates* » : l'ordinateur quantique doit pouvoir appliquer n'importe quelle porte quantique aux qubits. Pour cela, il doit disposer d'un jeu de portes quantiques universel.
5. « *A qubit-specific measurement capability* » : l'ordinateur quantique doit pouvoir mesurer les qubits qu'il manipule indépendamment. La mesure d'un qubit ne doit pas imposer la mesure de tous les qubits du système.

A ces cinq critères, David DiVincenzo ajoute deux critères de plus à respecter pour la communication quantique qui ne sont pas détaillés ici.

Ces critères sont les lignes directrices à suivre pour concevoir un ordinateur quantique. Une fois construit, de manière analogue à un processeur classique, il existe d'autres indicateurs permettant d'évaluer ses performances. Il ne s'agit pas de fréquence, de nombre de transistors, de cœurs logiques ou encore de dissipation thermique. Le premier paramètre est le nombre de qubits manipulables par le processeur quantique : plus ce nombre est grand, plus le processeur peut traiter des problèmes de grande taille. Viennent ensuite les connections entre ces qubits, c'est-à-dire la capacité à appliquer des portes quantiques à plus d'un qubit comme la porte CNOT. Plus il y a de connexions, meilleur est le processeur quantique. Les paramètres suivants sont les temps $T1$ et $T2$. $T1$ est le temps de relaxation. C'est le temps qu'un qubit dans l'état

$|1\rangle$ met pour se dégrader jusqu'à atteindre l'état $|0\rangle$. $T2$ quant lui, est le temps de cohérence. Il s'agit du temps entre l'application de la première porte quantique et la décohérence. Les temps $T1$ et $T2$ doivent être les plus long possibles. Enfin, les derniers paramètres à prendre en compte sont les taux d'erreurs. Il y a d'abord le taux d'erreur par porte sur un seul qubit, puis le taux d'erreur par porte sur deux qubits et le taux d'erreur de mesure. Une erreur se produit si après une de ces opérations quantiques le résultat obtenu n'est pas celui qui aurait dû être obtenu. Les taux d'erreurs doivent être les plus bas possibles. Nous pouvons analyser les paramètres du processeur Sycamore de Google (présenté dans la partie III.C). Il possède 53 qubits. Chacun de ces qubits est connecté avec 3,25 autres qubits du processeur en moyenne. Dans son cas, $T1 = 16.04\ \mu s$ et $T2$ est inconnu. Sycamore fait 0,15% d'erreur en moyenne sur les portes appliquées à un seul qubit et 0,36% d'erreur en moyenne sur les portes appliquées à deux qubits. Il fait également 3,1% d'erreur lors de la lecture [10].

Pour améliorer les performances des ordinateurs quantiques, certaines techniques existent. Par exemple, pour repousser la décohérence, les ordinateurs quantiques actuels fonctionnent souvent à une température de l'ordre 10mK. Cependant toutes les techniques connues à ce jour ne sont pas suffisantes et nous n'avons pas encore atteint une ère où les ordinateurs quantiques peuvent être utilisés pour résoudre de réels problèmes. Nous sommes actuellement dans l'ère NISQ (« Noisy Intermediate-Scale Quantum ») où l'ère des calculateurs quantiques bruités de taille intermédiaire. Peu d'algorithmes quantiques pourront être utilisés lors de cette ère à cause du nombre restreint de qubits disponibles et des forts taux d'erreur. Ce ne sera que lors de l'ère LSQ (« Large Scale Quantum »), ou l'ère des calculateurs quantiques tolérants aux défauts, que l'informatique quantique pourra être exploitée pour résoudre des problèmes qui sont aujourd'hui hors d'atteinte pour des ordinateurs classiques [Forteza *et al.* 2020].

Nous venons de décrire ce qui définit un ordinateur quantique. Découvrons maintenant quels acteurs sont impliqués dans le domaine de l'informatique quantique.

2. Offres et recherches actuelles de quelques acteurs majeurs de l'informatique quantique

Il y a de nombreux acteurs dans le domaine de l'informatique quantique œuvrant à différentes tâches. Tout d'abord, les chercheurs dans les centres de recherche ou les universités conçoivent de nouveaux algorithmes quantiques ou découvrent de nouvelles propriétés physiques pouvant être utilisées par des ordinateurs quantiques. Ensuite, les agences gouvernementales s'intéressent aussi à l'informatique quantique. C'est le cas de l'ANSSI en France par exemple [Poupard *et al.* 2018].

Les grandes entreprises de l'informatique investissent également dans l'informatique quantique et des start-ups se créent autour de projets basés sur l'informatique quantique. Nous avons déjà présenté le processeur Sycamore de Google. Il ne s'agit pas du seul processeur quantique conçu par Google car la firme de Mountain View a également développé le processeur quantique Bristlecone traitant 72 qubits. En parallèle des recherches sur le matériel, Google possède une équipe dédiée à la conception d'algorithmes quantiques [11]. Enfin, il dispose d'un outil en ligne nommé « Quantum Computing Playground » permettant de simuler jusqu'à 22 qubits. D'autres grandes entreprises américaines œuvrent au développement de l'informatique quantique comme IBM qui effectue des recherches dans ce domaine depuis 35 ans. Depuis 2017, IBM met à disposition certains ordinateurs quantiques à travers « IBM Q

Experience » (voir partie III.B). Intel, fabricant américain de microprocesseurs, travaille sur des processeurs quantiques basés sur la technologie des qubits de spin et a décrit à la conférence ISSCC de février 2020 le processeur quantique « Horse Ridge » de 128 qubits [12]. Microsoft, de son côté, s'intéresse à la technologie des qubits topologiques. Microsoft s'intéresse également au côté logiciel de l'informatique quantique et édite le langage de programmation Q#. Ce dernier est livré avec le kit de développement quantique (en sources ouvertes) qui s'accompagne d'un simulateur quantique. Enfin, Microsoft a annoncé « Azure Quantum » en novembre 2019 [13]. Il s'agit d'une offre cloud donnant accès à des ordinateurs quantiques d'autres entreprises telles que IonQ, une start-up créée autour de la technique des ions piégés. Suite à cette annonce de Microsoft, Amazon, son concurrent sur le secteur du cloud, a présenté lors de la conférence « re:Invent 2019 » en novembre 2019 la plateforme « Amazon Braket » qui est, elle aussi, une offre cloud donnant accès à des ordinateurs quantiques d'autres entreprises. Cependant, contrairement à Microsoft, Amazon ne développe aucun ordinateur quantique mais, comme Microsoft, s'est associé à d'autres entreprises comme Rigetti une start-up californienne.

Les entreprises américaines ne sont pas les seules à contribuer au développement de l'informatique quantique. En Europe, la société de services informatiques française Atos produit depuis 2017 la « Atos Quantum Learning Machine » (Atos QLM), un appareil de la taille d'un serveur capable de simuler jusqu'à 40 qubits. Ce dernier peut être programmé avec les langages AQASM ou pyAQASM développés par Atos pour pouvoir être exécutés aussi bien sur leur simulateur que sur de futurs ordinateurs quantiques. Il peut également simuler différents types de bruits afin de mieux simuler les vrais ordinateurs quantiques. Enfin, en Asie, l'entreprise chinoise Alibaba et l'académie chinoise des sciences se sont associées en 2015 pour créer le « Alibaba Quantum Computing Laboratory » [Ling 2015].

Sur le marché naissant de l'informatique quantique, tous les acteurs cités précédemment constituent l'offre. La demande, quant à elle, est constituée des entreprises souhaitant investir dans l'informatique quantique pour se préparer à la possible révolution à venir. Parmi ces entreprises, nous pouvons citer le constructeur automobile Volkswagen, l'avionneur Lockheed Martin, la banque JPMorgan…

Nous venons de présenter les principaux acteurs de l'informatique quantique à l'exception du cas particulier qu'est D-Wave que nous allons traiter maintenant.

3. D-Wave

D-wave est une start-up canadienne fondée en 1999 et spécialisée dans l'informatique quantique adiabatique*. Cette particularité fait de D-Wave un cas à part dans le domaine de l'informatique quantique. Il s'agit de l'unique entreprise impliquée dans l'informatique quantique adiabatique. L'informatique quantique adiabatique est différente de l'informatique quantique universelle. Tous les concepts présentés dans ce rapport sont valides dans le domaine de l'informatique quantique universelle mais pas nécessairement dans celui de l'informatique quantique adiabatique. L'informatique quantique adiabatique manipule des qubits mais n'utilise pas de portes quantiques. Le concept de circuit quantique n'existe pas non plus. Au lieu d'exécuter des circuits quantiques, les ordinateurs de D-Wave ont pour seul cas d'utilisation le recuit quantique [Finnila *et al.* 1994].

Le recuit quantique est une technique pour trouver un minimum global d'une fonction objectif. Cette technique n'est donc applicable qu'aux problèmes d'optimisation. Remarquons toutefois que les problèmes d'optimisation sont couramment rencontrés, par exemple au sein des entreprises clientes de l'informatique quantique citées dans la partie précédente comme Volkswagen ou Lockheed Martin.

Etant donné un problème d'optimisation, le recuit quantique consiste à préparer un système de qubits reliés entre eux par des liaisons pondérées représentant une solution approchée de la solution optimale. L'ordinateur quantique modifie ensuite les poids du système pour l'amener vers un état d'équilibre qui correspond au minimum de la fonction objectif. Lors de ce processus, le système ne doit pas perdre d'énergie, d'où le nom d'informatique quantique adiabatique. A cause de la décohérence, ce processus ne peut pas toujours aboutir au résultat optimal et il est souvent nécessaire de le répéter jusqu'à atteindre la solution optimale.

Actuellement, D-Wave commercialise sa quatrième génération d'ordinateurs quantiques pour le recuit quantique avec le D-Wave 2000Q (Figure 15) depuis janvier 2017. Ce dernier possède 2 000 qubits et fonctionne à une température de 15mK pour limiter le risque d'échange d'énergie.

Figure 15 : D-Wave 2000Q : ordinateur entier (à gauche) et processeur quantique (à droite)

En septembre 2019, D-Wave a annoncé sa cinquième génération d'ordinateurs quantiques avec l'ordinateur « Advantage » qui possèdera 5 000 qubits et sera disponible en 2020 [13].

Ainsi, le secteur d'activité de D-Wave est l'informatique quantique adiabatique. Il s'agit d'un champ d'application restreint qui est différent de l'informatique quantique universelle. Bien que les concepts de portes quantiques et de circuits quantiques n'aient pas cours, le recuit quantique est une technique permettant de résoudre des problèmes d'optimisation utilisable dans de nombreux domaines et qui donne de meilleurs résultats que les méta-heuristiques telles que le recuit simulé. Un ordinateur de D-Wave a, par exemple, été utilisé pour le calcul de repliement de protéines [Perdomo-Ortiz *et al.* 2012].

Nous venons de présenter les principaux acteurs de l'informatique quantique ainsi que leurs secteurs d'activité. Intéressons nous maintenant plus précisément à IBM et à sa plateforme IBM Q Experience.

B. IBM Q Experience

1. Présentation de l'outil

En 2016, IBM lance « IBM Q Experience », une plateforme en ligne donnant accès gratuitement à un ordinateur quantique disposant de 5 qubits. Aujourd'hui, la plateforme dispose de cinq ordinateurs quantiques ayant 5 qubits, d'un ordinateur ayant 1 qubit et d'un ordinateur ayant 15 qubits ainsi que d'un simulateur pouvant simuler jusqu'à 32 qubits. Les ordinateurs quantiques mis à disposition sur IBM Q Experience utilisent des transmons* supraconducteurs pour implémenter leurs qubits. Voici les caractéristiques de certains de ces ordinateurs :

Ordinateur	Nombre de qubits	Connectivité moyenne	T1 moyen (µs)	T2 moyen (µs)	Taux d'erreur moyen (1 qubit)	Taux d'erreur moyen (2 qubits)
IBM Q Ourense	5	1,6	92,66	60,50	0,07%	0,8%
IBM Q16 Melbourne	15	2,67	52,2	64,8	0,8%	7%

Tableau 3 : Caractéristiques de certains ordinateurs quantiques d'IBM Q Experience

Les ordinateurs disponibles sur IBM Q Experience sont calibrés deux dois par jours à 01:00 et 13:00 GMT. Pour plus de précision concernant la connectivité et les taux d'erreurs des différents qubits, lors des calibrations des schémas comme les suivants sont produits :

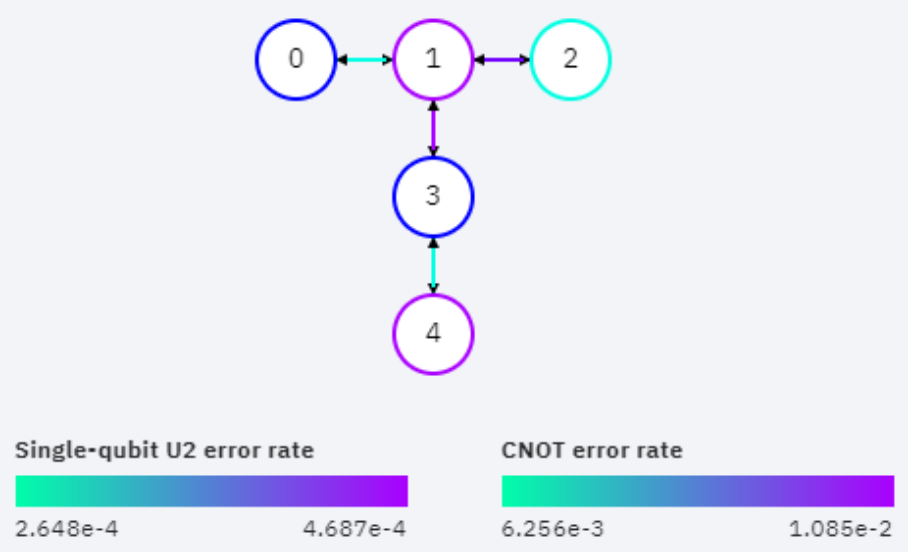


Figure 16 : Connectivité et taux d'erreurs de IBM Q Ourense le 26 février 2020 avant 13:00

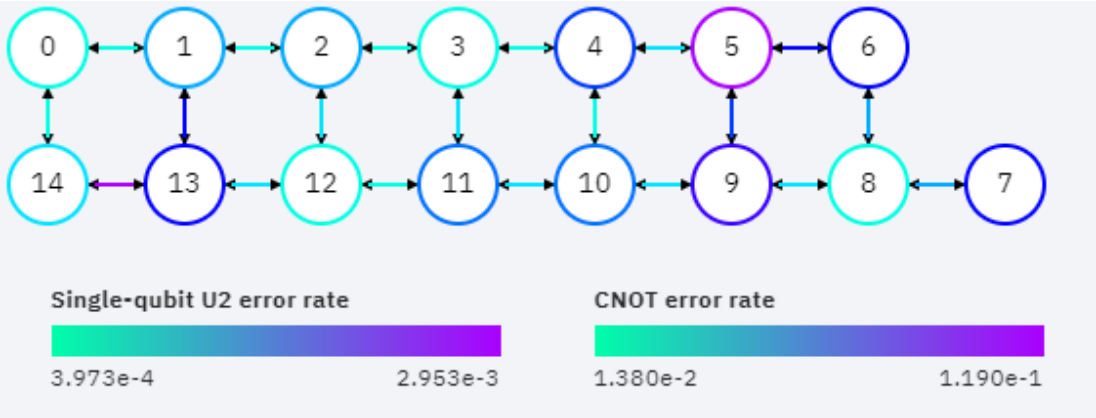


Figure 17 : Connectivité et taux d'erreurs de IBM Q16 Melbourne le 26 février 2020 avant 13:00

Le fait que seules deux portes quantiques soient testées peut sembler étrange mais en réalité, les ordinateurs quantiques ne peuvent exécuter que deux portes quantiques différentes : la porte $U(\theta, \phi, \lambda)$ qui agit sur un seul qubit et la porte CNOT. θ, ϕ, λ sont des paramètres statiques de la porte U qui est une matrice unitaire générique. Ces paramètres doivent être spécifiés lors de l'utilisation de la porte.

$$U(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda} sin(\frac{\theta}{2}) \\ e^{i\phi} sin(\frac{\theta}{2}) & e^{i\lambda + i\phi} cos(\frac{\theta}{2}) \end{bmatrix}$$

$$0 \leq \theta \leq \pi$$

$$0 \leq \phi \leq 2\pi$$

$$0 \leq \lambda \leq 2\pi$$

Toutes les portes quantiques peuvent être décomposées en portes $U(\theta, \phi, \lambda)$ et CNOT, par exemple, $Z = U(0, 0, \pi)$. Toutes les décompositions sont décrites en annexe 3.

Pour pouvoir programmer ses ordinateurs quantiques, IBM propose trois façons de décrire des circuits quantiques :

1. Le « composer » qui est un outil graphique ressemblant à une partition de musique sur laquelle il est possible de glisser-déposer des portes quantiques pour implémenter un circuit quantique.
2. Le langage « OpenQASM » (prononcé « open kasm ») qui est un langage bas niveau qui permet de décrire des circuits quantiques. De tels circuits quantiques peuvent ensuite être dessinés par le composer.
3. La bibliothèque « Qiskit ». C'est une bibliothèque Python développée par IBM et sortie en 2017. Elle sert de SDK* pour l'informatique quantique et d'API* pour IBM Q Experience.

Le composer représente les portes quantiques appliquées à un registre de qubits au cours du temps sous la forme d'une partition. Par exemple le circuit d'intrication de deux qubits de la partie II.A.2 se présente comme suit :

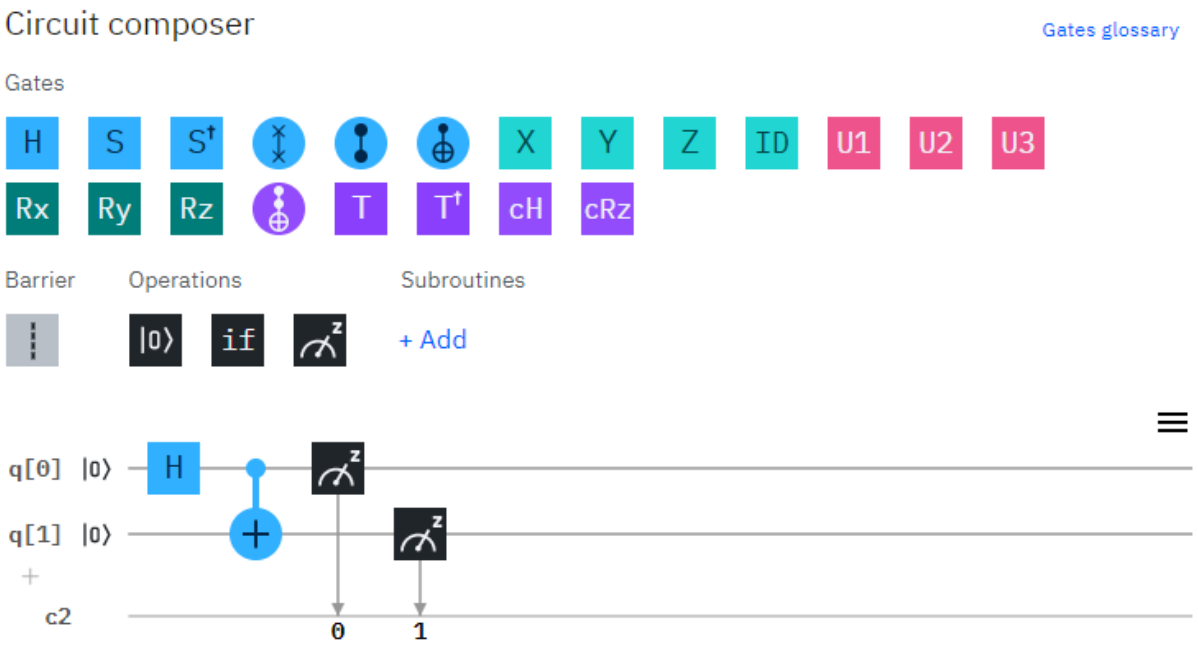


Figure 18 : Partition d'un circuit quantique sous IBM Q Experience

On peut voir les différentes portes utilisables : H, SWAP, CNOT, X, Y, Z, Toffoli… Ces portes peuvent être combinées pour former des routines plus élaborées. Par défaut, les portes ajoutées au circuit sont toujours ajoutées le plus à gauche possible (en respectant la position d'insertion). Pour éviter cela, il est possible d'utiliser des barrières. Enfin, il existe trois autres opérations : la remise à $|0\rangle$, l'application de portes quantiques conditionnelle et la mesure. Une fois le circuit terminé, il est possible de l'exécuter sur un des ordinateurs quantiques de la plateforme.

Les circuits conçus avec le composer sont stockés en OpenQASM sur IBM Q Experience. C'est pourquoi il est possible d'éditer le code OpenQASM du circuit directement mais aussi de télécharger ou de téléverser du code OpenQASM sur IBM Q Experience. Le circuit Figure 18 se programme en OpenQASM de la manière suivante :

Circuit editor

```
1  OPENQASM 2.0;
2  include "qelib1.inc";
3
4  qreg q[2];
5  creg c[2];
6
7  h q[0];
8  cx q[0],q[1];
9  measure q[0] -> c[0];
10 measure q[1] -> c[1];
```

Figure 19 : Code OpenQASM d'un circuit quantique

Le langage OpenQASM sert d'intermédiaire entre les représentations de circuits quantiques de plus haut niveau et les ordinateurs quantiques. Nous ne détaillerons pas ses spécifications dans ce rapport. Cependant, voici une explication du code Figure 19 :

- Ligne 1 : la version d'OpenQASM utilisée est la version 2.
- Ligne 2 : le fichier « qelib1.inc » définissant les portes quantiques de bases est inclus. Ce fichier est disponible en annexe 3 de ce rapport.
- Lignes 4 et 5 : un registre quantique de 2 qubits et un registre classique de 2 bits sont déclarés.
- Lignes 7 et 8 : le circuit est défini, une porte de Hadamard est appliquée au qubit 0 et une porte CNOT est appliquée aux qubits 0 (qubit de contrôle) et 1 (qubit cible).
- Lignes 9 et 10 : les qubits sont mesurés et les valeurs produites sont stockées dans le registre classique.

Enfin, le dernier outil disponible sur IBM Q Experience est la bibliothèque Python Qiskit (« Quantum Information Science Kit »). A la différence du composer et de l'OpenQASM, Qiskit ne sert pas à décrire un circuit quantique mais à écrire un programme Python qui génère un circuit quantique. Le circuit quantique généré peut ensuite être exporté en OpenQASM ou directement exécuté sur un ordinateur quantique. L'inverse est possible, un code OpenQASM peut être importé avec Qiskit.

Qiskit se compose de quatre parties :

- « Terra » : contient les éléments de bases pour créer des circuits quantiques.
- « Aer » : permet de simuler les circuits quantiques.
- « Ignis » : permet d'atténuer les erreurs produites par les vrais ordinateurs quantiques.
- « Aqua » : contient des éléments pour la conception de circuits quantiques de plus haut niveau.

Toutes les fonctionnalités de Qiskit ne seront pas détaillées dans ce rapport. Voici le code Python permettant de générer le circuit quantique des précédents exemples :

```
1   from qiskit import QuantumCircuit
2
3   circuit = QuantumCircuit(2, 2)
4
5   circuit.h(0)
6   circuit.cx(0, 1)
7
8   circuit.measure([0, 1], [0, 1])
```

Figure 20 : Code Qiskit de génération d'un circuit quantique

Nous venons de présenter la plateforme IBM Q Experience, voyons maintenant comment implémenter l'algorithme de Grover présenté partie II.B sur cette plateforme.

2. Implémentation de l'algorithme de Grover

Dans cette partie, nous implémenterons l'algorithme de Grover pour rechercher l'état propre $|101\rangle$ représenté sur 3 qubits comme présenté dans la partie II.B. Nous implémenterons

les deux versions possibles de l'oracle, à savoir celle avec une porte Pauli-Z et celle avec une porte CNOT.

Avant de commencer, il est intéressant de s'attarder sur l'endianisme des ordinateurs quantiques d'IBM Q Experience. Avec ces ordinateurs, le qubit q_0 correspond au qubit de poids faible, c'est-à-dire le chiffre le plus à droite de la notation $|101\rangle$.

Pour implémenter l'algorithme de Grover, nous utilisons Qiskit. La première chose à faire est d'importer les classes et fonctions dont nous aurons besoin par la suite et de s'authentifier pour avoir accès à IBM Q Experience.

```
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.visualization import plot_histogram
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.ignis.mitigation.measurement import complete_meas_cal, CompleteMeasFitter
from math import pi, floor, sqrt
```

Figure 21 : Classes et fonctions de Qiskit à importer

```
provider = IBMQ.load_account()
```

Figure 22 : Chargement du compte IBM Q Experience

Ensuite, nous construisons les portes CCZ et CCCNOT. Pour la porte CCZ, nous n'utilisons pas de qubits auxiliaires mais choisissons d'utiliser une porte Toffoli et deux portes de Hadamard : $CCZ = (I \otimes I \otimes H)(Toffoli)(I \otimes I \otimes H)$. Nous pouvons remarquer que la porte C^nX ou porte Pauli-X contrôlée par n qubits est une fonction disponible dans le module Aqua de Qiskit. Cependant, afin de mieux comprendre et expérimenter le fonctionnement de l'algorithme de Grover, nous implémenterons cette porte comme décrite dans la partie II.B.

```
ccz = QuantumCircuit(3, name='CCZ')
ccz.h(2)
ccz.toffoli(0, 1, 2)
ccz.h(2)

ccz.draw()
```

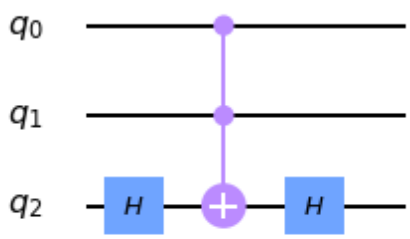


Figure 23 : Implémentation de CCZ avec Qiskit

```
cccnot = QuantumCircuit(6, name='CCCNOT')
cccnot.toffoli(0, 1, 3)
cccnot.toffoli(2, 3, 4)
cccnot.cnot(4, 5)
cccnot.toffoli(2, 3, 4)
cccnot.toffoli(0, 1, 3)

cccnot.draw()
```

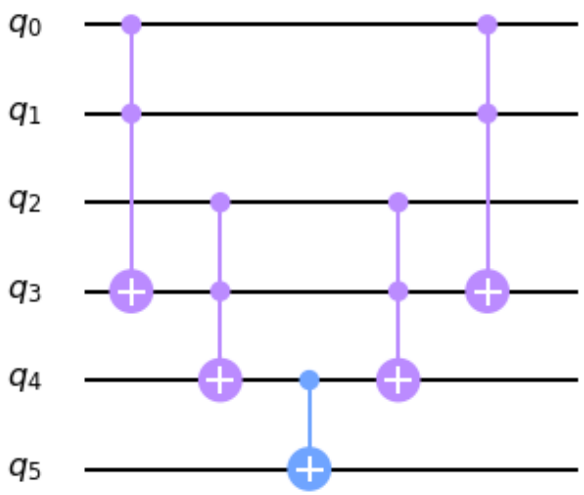


Figure 24 : Implémentation de CCCNOT avec Qiskit

Nous disposons à présent de toutes les portes quantiques nécessaires à l'implémentation des oracles U_f. Implémentons les deux versions de l'oracle décrites dans la partie II.B.

```
uf_with_z = QuantumCircuit(3, name='Uf_z')
uf_with_z.x(1)
uf_with_z.append(ccz.to_instruction(), range(3))
uf_with_z.x(1)

uf_with_z.draw()
```

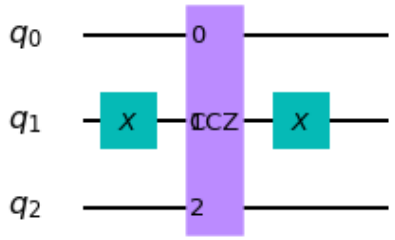


Figure 25 : Oracle U_f implémenté avec une porte CCZ

```
uf_with_not = QuantumCircuit(6, name='Uf_not')
uf_with_not.x(1)
uf_with_not.append(cccnot.to_instruction(), range(6))
uf_with_not.x(1)

uf_with_not.draw()
```

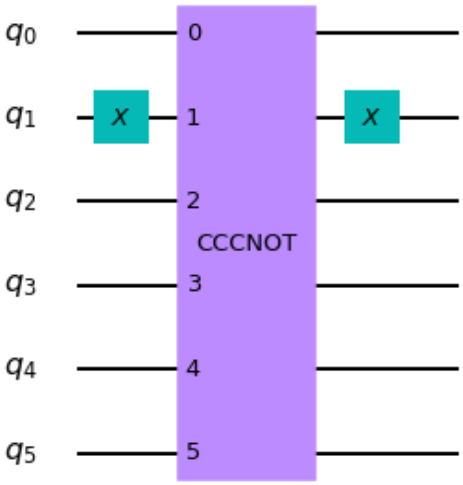


Figure 26 : Oracle U_f implémenté avec une porte CCCNOT

Maintenant que nous disposons des oracles U_f, nous devons implémenter l'opérateur U_s.

```
us = QuantumCircuit(3, name='Us')
us.h(range(3))
us.x(range(3))
us.append(ccz.to_instruction(), range(3))
us.x(range(3))
us.h(range(3))

us.draw()
```

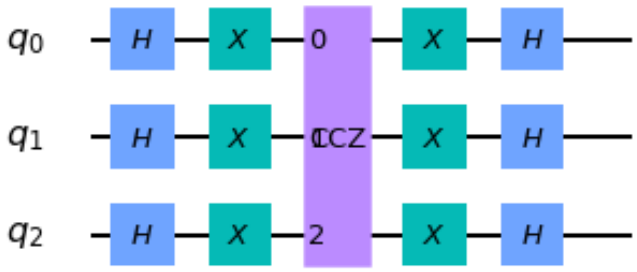


Figure 27 : Implémentation de l'opérateur U_s

Nous avons enfin de quoi implémenter les deux versions de l'algorithme de Grover. Dans le cas de la seconde version (avec CCCNOT), le qubit auxiliaire doit être initialisé à $|1\rangle$ afin de ne pas avoir la même phase que les autres après application de la porte de Hadamard. Comme décrit dans la partie II.B, chaque qubit du registre (sauf les qubits auxiliaires de la porte CCCNOT) se voient appliquer la porte de Hadamard. Puis, nous appliquons $\left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ les circuits U_f puis U_s. Enfin, nous ajoutons une opération de mesure du registre quantique pour pouvoir lire le résultat final de l'algorithme.

```python
def build_grover_circuit(oracle: QuantumCircuit, ancilla: bool = False) -> QuantumCircuit:
    """
    Function to build a circuit implementing Grover's algorithm given an oracle Uf.

    :param oracle: the oracle Uf to put in the circuit
    :param ancilla: True if the circuit has an ancilla qubit that must be initialized to |1>
    """
    nb_qubits = oracle.n_qubits
    circuit = QuantumCircuit(nb_qubits, nb_qubits)
    if(ancilla):
        circuit.x(nb_qubits - 1)
        circuit.h(nb_qubits - 1)
    circuit.h(range(us.n_qubits))
    for i in range(floor(pi/4 * sqrt(2**nb_qubits))):
        circuit.append(oracle.to_instruction(), range(nb_qubits))
        circuit.append(us.to_instruction(), range(us.n_qubits))
    circuit.measure(range(us.n_qubits), range(us.n_qubits))
    return circuit
```

Figure 28 : Fonction générant le circuit de l'algorithme de Grover

```python
circuit_z = build_grover_circuit(uf_with_z)
circuit_z.draw()
```

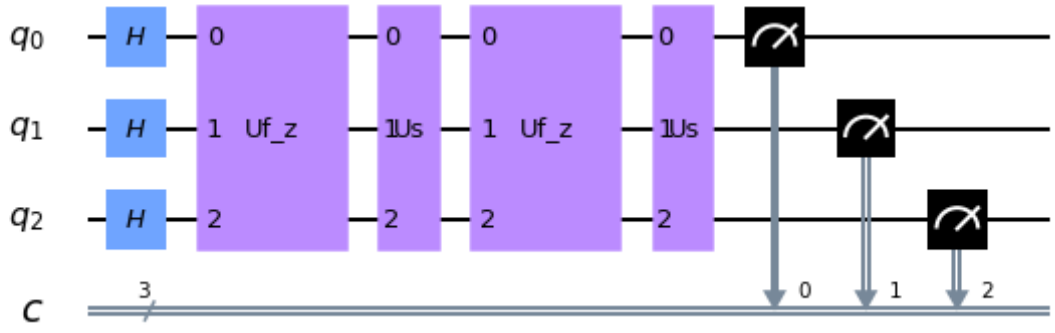


Figure 29 : Algorithme de Grover avec une porte CCZ

```
circuit_not = build_grover_circuit(uf_with_not, True)
circuit_not.draw()
```

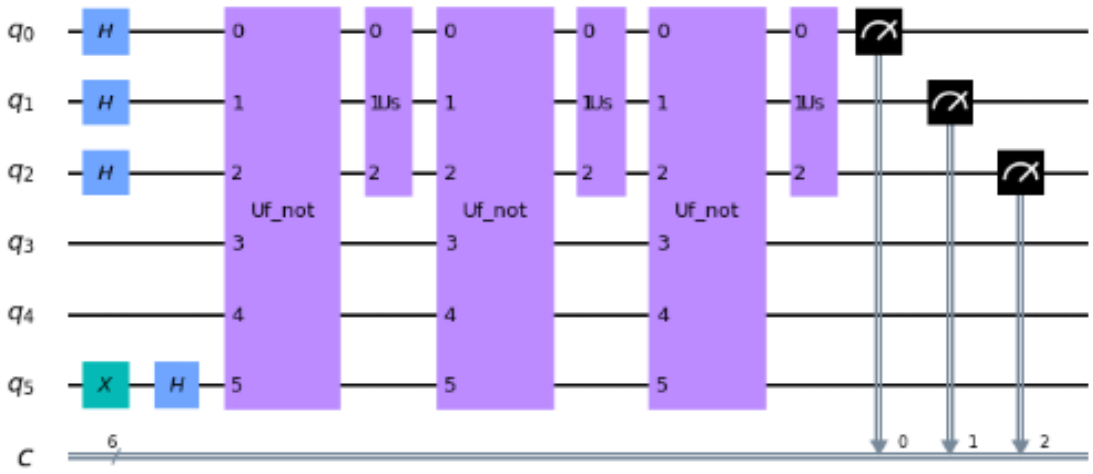


Figure 30 : Algorithme de Grover avec une porte CCCNOT

Ainsi, nous avons implémenté deux versions possibles de l'algorithme de Grover pour rechercher l'état propre $|101\rangle$ représenté sur 3 qubits avec Qiskit. Maintenant que nous disposons d'implémentations, exécutons l'algorithme de Grover sur un véritable ordinateur quantique d'IBM Q Experience.

3. Exécution de l'algorithme de Grover

Après avoir conçu des circuits implémentant l'algorithme de Grover, nous pouvons les exécuter sur de vrais ordinateurs quantiques. Mais avant cela, il est utile de simuler l'exécution de ces circuits. En effet, simuler un circuit quantique permet de connaître le résultat théorique auquel il faut s'attendre lors de l'exécution du circuit. Si le résultat de la simulation n'est pas proche de celui désiré, il faut alors déboguer. Puisque les deux circuits implémentent l'algorithme de Grover et que la simulation et l'exécution de ces derniers produit des résultats similaires, nous ne présenterons que la simulation et l'exécution du circuit utilisant la porte CCZ.

```
def simulate(circuit):
    simulator = Aer.get_backend('qasm_simulator')
    job = execute(circuit, backend=simulator, shots=1024)
    return job.result().get_counts(circuit)

plot_histogram(simulate(circuit_z))
```

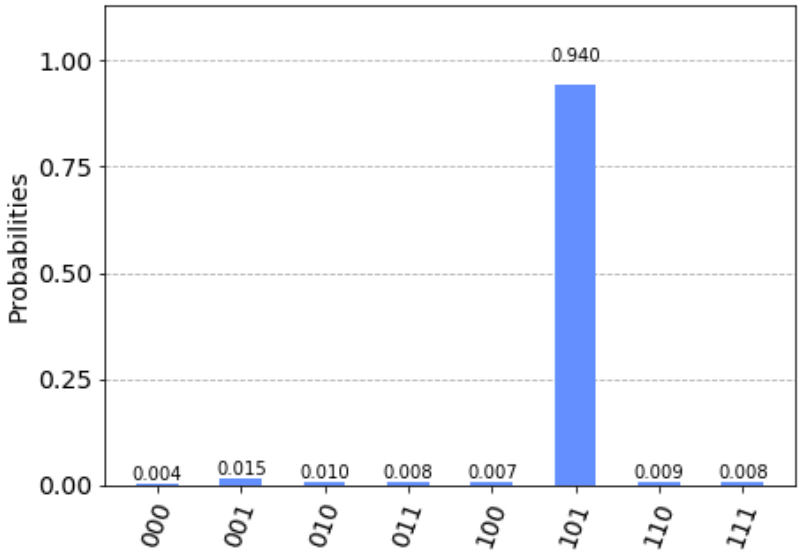


Figure 31 : Résultat de la simulation de l'algorithme de Grover implémenté avec une porte CCZ

Sur l'histogramme de la Figure 31, nous observons le résultat attendu de l'exécution du circuit, à savoir, une très forte fréquence de mesure de l'état propre $|101\rangle$ (94% des 1024 exécutions) et des fréquences très faibles pour les autres états propres. Il est normal de ne pas avoir 100% de probabilité pour l'état recherché car l'algorithme de Grover n'est pas déterministe comme expliqué dans la partie II.B. Le résultat théorique est de mesurer l'état propre $|101\rangle$ avec une probabilité supérieure à $1 - 2^{-3} = 87{,}5\%$.

Maintenant que connaissons le résultat théorique de notre circuit, nous pouvons l'exécuter sur un véritable ordinateur quantique.

```
device = least_busy(provider.backends(filters=lambda b: not b.configuration().simulator
                                            and b.status().operational
                                            and b.configuration().n_qubits >= circuit_z.n_qubits))
print(device)
```

```
ibmq_ourense
```

```
job = execute(circuit_z, backend=device, shots=1024)
job_monitor(job)
```

```
Job Status: job has successfully run
```

Figure 32 : Exécution d'un circuit sur un ordinateur quantique d'IBM Q Experience

Nous cherchons d'abord quel ordinateur quantique est le moins utilisé parmi ceux pouvant exécuter notre circuit. Il s'agissait alors de « IBM Q Ourense ». Puis nous soumettons notre circuit qui devient alors une tache à exécuter ajoutée à la file d'attente de l'ordinateur quantique choisi. Comme avec la simulation, nous spécifions que nous souhaitons exécuter 1024 fois le circuit.

```
counts = job.result().get_counts()
plot_histogram(counts)
```

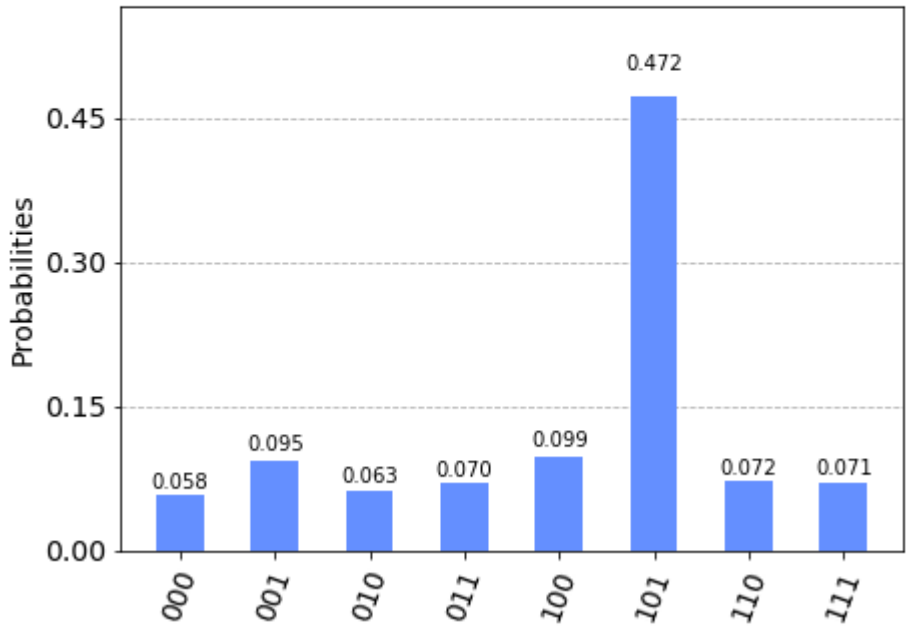


Figure 33 : Résultat de l'exécution du circuit quantique de l'algorithme de Grover sur l'ordinateur IBM Q Ourense

L'histogramme des résultats semble similaire à celui de la simulation mais les fréquences ne sont plus les mêmes. En effet, sur un véritable ordinateur quantique, le résultat est bruité à cause des erreurs lors de l'application des portes quantiques aux qubits. Pour cette raison, bien que l'état propre $|101\rangle$ soit l'état le plus fréquemment mesuré, la fréquence de 47,2% est très inférieure à celle de 94% attendue. Par conséquent, il est plus probable que le circuit donne un mauvais résultat plutôt que le bon.

Toutefois, il est possible d'atténuer ce bruit en se basant sur la calibration de l'ordinateur quantique et en utilisant les fonctionnalités du module Ignis de Qiskit.

```
calibration_circuits, state_labels = complete_meas_cal(qubit_list=range(circuit_z.n_qubits))
calibration_jobs = execute(calibration_circuits, backend=device, shots=1024*4, optimization_level=0)
job_monitor(calibration_jobs)
calibration_results = calibration_jobs.result()
measurement_fitter = CompleteMeasFitter(calibration_results, state_labels)
measurement_filter = measurement_fitter.filter

mitigated_result = measurement_filter.apply(job.result())
mitigated_counts = mitigated_result.get_counts(circuit_z)

plot_histogram([counts, mitigated_counts], legend=['raw', 'mitigated'])
```

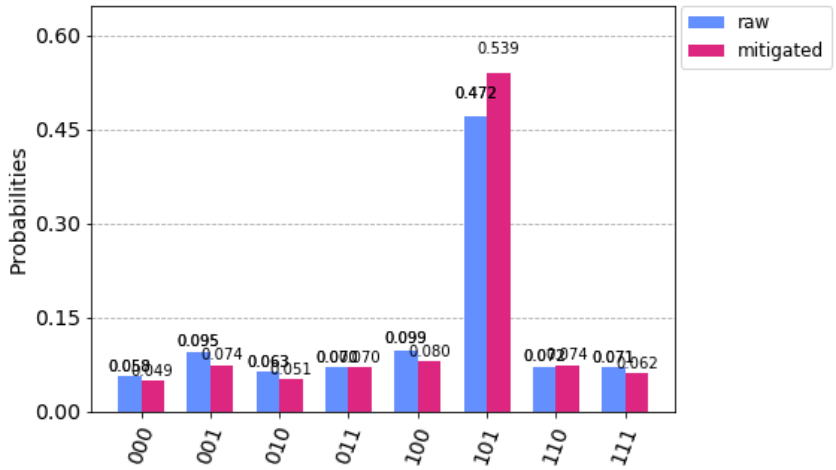


Figure 34 : Résultats bruts et atténués de l'exécution du circuit quantique de l'algorithme de Grover sur l'ordinateur IBM Q Ourense

Nous observons que les résultats avec atténuation du bruit sont légèrement meilleurs que les résultats originaux. Cette fois, il est plus probable que le circuit donne un bon résultat (53,9%) plutôt qu'un mauvais (46,1%).

Ainsi, nous avons montré, d'une part, que l'algorithme de Grover n'est pas déterministe (voir partie II.B) et d'autre part, que les ordinateurs quantiques actuels sont bruités (voir partie III.A.1). Lors de l'exécution d'un circuit quantique, les résultats obtenus avec un ordinateur quantique correspondent à ceux attendus mais ne sont pas d'aussi bonne qualité. Une dernière remarque est que nous avons implémenté l'algorithme de Grover porte par porte pour mieux pouvoir l'étudier. Cependant, Qiskit étant une bibliothèque, cet algorithme est déjà implémenté dans le module Aqua.

Conclusion

A travers ce rapport, nous avons découvert l'informatique quantique. L'informatique quantique repose principalement sur trois principes de la physique quantique : la superposition quantique, l'intrication quantique et la réduction du paquet d'ondes. Ces trois principes permettent de définir le concept de qubit qui est le support de l'information quantique. Contrairement aux bits, les qubits peuvent se trouver dans une superposition d'états et peuvent être intriqués. Cependant, ils possèdent aussi leurs limites comme la décohérence qui empêche de manipuler des qubits sur de longues durées ou le théorème de non-clonage qui empêche de copier l'état d'un qubit. Les qubits peuvent être manipulés avec des portes quantiques. Ces dernières permettent de créer des circuits quantiques qui, eux-mêmes, peuvent implémenter des algorithmes quantiques.

De nombreux algorithmes quantiques existent déjà et peuvent être appliqués à différents domaines de l'informatique. Dans ce rapport, nous avons analysé puis implémenté l'algorithme de Grover et l'avons finalement exécuté sur un véritable ordinateur quantique disponible via IBM Q Experience. Lors de ces tests, nous avons pu constater des problèmes de reproductibilité. IBM comme Microsoft, Google ou encore Atos sont actuellement des acteurs majeurs de l'informatique quantique.

L'informatique quantique est née il y a 40 ans et s'est développé depuis. Le rapport sur la stratégie de la France en informatique quantique, remis le 9 janvier 2020, anticipe la démocratisation des ordinateurs quantiques. 37 propositions sont énoncées pour se préparer à ce « virage technologique » [Forteza *et al.* 2020]. Parmi elles, nous pouvons citer la mise en place, dès 2022, de la cryptographie post-quantique pour se prémunir des futurs outils de cryptanalyse quantique ou encore, l'inclusion de 6 crédits ECTS* pour l'algorithmie quantique dans les principaux cycles d'ingénieurs en informatiques.

Références bibliographiques

[Arute *et al.* 2019] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov,, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven , John M. Martinis, 2019. « Quantum supremacy using a programmable superconducting processor », Nature, Vol. 574, pp. 505-510

[Aspuru-Guzik *et al.* 2005] Alán Aspuru-Guzik, Anthony D. Dutoi, Peter J. Love, Martin Head-Gordon, 2005. « Simulated Quantum Computation of Molecular Energies », Science 309, p. 1704, 20 pages.

[Barenco *et al.* 1995] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John Smolin, Harald Weinfurter, 1995. « Elementary Gates for Quantum Computation. » Physical Review A, Vol. 52.5, pp. 3457-3467.

[Benioff 1980] Paul Benioff. 1980, « 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, No. 5.

[Bennett 1973] Charles H. Bennett, 1973. « Logical Reversibility of Computation » IBM Journal of Research and Development, Volume 17, numéro 6, pp. 525-532.

[Bennett Brassard 1984] Charles H. Bennett, Gilles Brassard, 1984. « Quantum cryptography: Public key distribution and coin tossing". Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, Volume 175, page 8.

[Bloch 1946] Félix Bloch, 1946. « Nuclear Induction », Phys. Rev., Vol. 70, Issue 7-8, pp. 460-474.

[Boyer *et al.* 1996] Michel Boyer, Gilles Brassard, Peter Hoeyer, Alain Tapp, 1996. « Tight bounds on quantum search », Proceedings of the Workshop on Physics of Computation: PhysComp ’96, Institute of Electrical and Electronic Engineers Computer Society, 8 pages.

[Coppersmith 1994] Don Coppersmith, 1994. « An approximate Fourier transform useful in quantum factoring ». Technical Report RC19642, IBM, 9 pages.

[Cross *et al.* 2017] Andrew W. Cross, Lev S. Bishop, John A. Smolin, Jay M. Gambetta, 2017. « Open Quantum Assembly Language », 24 pages.

[Deutsch Jozsa 1992] David Deutsch, Richard Jozsa, 1992. « Rapid solutions of problems by quantum computation », Proceedings of the Royal Society of London A, Vol. 439, pp. 553-558.

[Deutsch Penrose 1985] David Deutsch, Roger Penrose, 1985. « Quantum theory, the Church–Turing principle and the universal quantum computer », Proceedings of the Royal Society of London A, Vol. 400, Issue 1818.

[Dirac 1939] Paul A. M. Dirac, 1939. « A new notation for quantum mechanics », Mathematical Proceedings of the Cambridge Philosophical Society, Vol. 35, Issue 3, pp. 416-418.

[DiVincenzo 2000] David P. DiVincenzo, 2000. « The Physical Implementation of Quantum Computation. » Fortschritte der Physik 48.9-11, pp. 771–783.

[Fickler *et al.* 2013] Robert Fickler, Mario Krenn, Radek Lapkiewicz, Sven Ramelow, Anton Zeilinger, 2013. « Real-Time Imaging of Quantum Entanglement », Scientific Reports Vol. 3, Article 1914, 5 pages.

[Figgatt *et al.* 2017] C. Figgatt, D. Maslov, K. A. Landsman, N. M. Linke, S. Debnath, C. Monroe, 2017. « Complete 3-Qubit Grover Search on a Programmable Quantum Computer », Nature Communications 8.1, 11 pages.

[Finnila *et al.* 1994] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, J.D. Doll, 1994. « Quantum Annealing: A New Method for Minimizing Multidimensional Functions. » Chemical Physics Letters 219.5-6 : pp. 343–348.

[Forteza *et al.* 2020] Paula Forteza, Jean-Paul Herteman, Iordanis Kerenidis, 2020. « Quantique : le virage technologique que la France ne ratera pas », rapport gouvernemental, 68 pages.

[Grover 1996] Lov K. Grover, 1996. « A fast quantum mechanical algorithm for database search », Proceedings, 28th Annual ACM Symposium on the Theory of Computing, pp. 212-219.

[Harrow *et al.* 2009] Aram W. Harrow, Avinatan Hassidim, Seth Lloyd, 2009. « Quantum algorithm for solving linear systems of equations », Physical Review Letters 103.15, 15 pages.

[Havlicek *et al.* 2018] Vojtech Havlicek, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, Jay M. Gambetta, 2018. « Supervised learning with quantum enhanced feature spaces », Nature. Volume 567, pp. 209-212.

[Heisenberg 1927] Werner K. Heisenberg, 1927. « Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik », Z. Phys. 43, pp. 172–198.

[Jones Mosca 1998] J. A. Jones, M. Mosca, 1998. « Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer », The Journal of Chemical Physics, Vol. 109, Issue 5.

[Kassal *et al.* 2011] Ivan Kassal, James D. Whitfield, Alejandro Perdomo-Ortiz, Man-Hong Yung, Alán Aspuru-Guzik, 2010. « Simulating Chemistry Using Quantum Computers. » Annual Review of Physical Chemistry 62.1 : pp. 185–207.

[L'Ecuyer *et al.* 2002] Pierre L'Ecuyer, Richard J. Simard, E. Jack Chen, David Kelton, 2002. « An object-oriented random number package with many long streams and substreams. Operations Research », 50(6):1073–1075.

[Ling 2015] Xin Ling, 2015. « Joint quantum computing venture is a first for China », Physics World, Volume 28, numéro 9.

[Llewellyn *et al.* 2019] Daniel Llewellyn, Yunhong Ding, Imad I. Faruque, Stefano Paesani, Davide Bacco, Raffaele Santagati, Yan-Jun Qian, Yan Li, Yun-Feng Xiao, Marcus Huber, Mehul Malik, Gary F. Sinclair, Xiaoqi Zhou, Karsten Rottwitt, Jeremy L. O'Brien, John G. Rarity, Qihuang Gong, Leif K. Oxenlowe, Jianwei Wang, Mark G. Thompson, 2019. « Chip-to-chip quantum teleportation and multi-photon entanglement in silicon ». Nature Physics. Doi : 10.1038/s41567-019-0727-x.

[Matsumoto Nishimura 1998] Makoto Matsumoto, Takuji Nishimura, 1998. « Mersenne twister: A 623-dimensionnally equidistributed uniform pseudorandom number generator. » ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation, 8(1):3–30.

[Nielsen Chuang 2000] Michael Nielsen, Isaac Chuang, 2000. « Quantum Computation and Quantum Information », ISBN 978-1-107-00217-3, Cambridge University Press, 675 pages.

[Ömer 1998] Bernhard Ömer, 1998. « A procedural formalism for quantum computing », Département de physique théorique, université technique de Vienne.

[Park 1970] James L. Park, 1970. « The concept of transition in quantum mechanics ». Foundations of Physics, Vol. 1, numéro 1, pp. 23-33.

[Pednault *et al.* 2019] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, Robert Wisnieff, 2019. « Leveraging secondary storage to simulate deep 54-qubit sycamore circuits », arXiv:1910.09534, 39 pages.

[Perdomo-Ortiz *et al.* 2012] Alejandro Perdomo-Ortiz, Neil Dickson, Marshall Drew-Brook, Geordie Rose, Alán Aspuru-Guzik, 2012. « Finding low-energy conformations of lattice protein models by quantum annealing. » Nature Scientific Reports 2, article numéro 571.

[Poupard *et al.* 2018] Guillaume Poupard, Anne-Catherine Belliot, Séverine Oger, Anne-Catherine Belliot, Marc Renaudin, 2018. « Rapport Annuel 2018 », Agence nationale de la sécurité des systèmes d'information (ANSSI), 52 pages.

[Prechtel *et al.* 2016] Jonathan H. Prechtel, Andreas V. Kuhlmann, Julien Houel, Arne Ludwig, Sascha R. Valentin, Andreas D. Wieck, Richard J. Warburton, 2016. « Decoupling a hole spin qubit from the nuclear spins », Nature Materials Vol. 15, pp. 981–986.

[Preskill 2012] John Preskill, 2012. « Quantum computing and the entanglement frontier », Proceedings of the 25th Solvay Conference on Physics, 18 pages.

[Prieur 2019] Benoît Prieur, 2019. « Informatique quantique : de la physique quantique à la programmation quantique en Q# », ISBN 978-2-409-01741-4, Éditions ENI, 244 pages.

[Rieffel Polak 2000] Eleanor G. Rieffel, Wolfgang Polak, 2000. « An Introduction to Quantum Computing for Non-Physicists », ACM Comput.Surveys 32 : 300-335, 45 pages.

[Rivest *et al.* 1978] Ronald Rivest, Adi Shamir, Leonard Adleman, 1978. « A method for obtaining digital signatures and public-key cryptosystems », Communications of the ACM, Volume 21, numéro 2, pp. 120–126.

[Shor 1994] Peter W. Shor, 1994. « Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer », Proceedings of the 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, pp. 124-134.

[Toffoli 1980] Tommaso Toffoli, 1980. « Reversible computing. » International colloquium on automata, languages, and programming, Springer, Berlin, Heidelberg, pp. 632-644.

[Van der Corput 1935] Johannes Gaultherus Van der Corput, 1935. « Verteilungsfunktionen (Erste Mitteilung) » Proceedings of the Koninklijke Akademie van Wetenschappen te Amsterdam, 38 : 813-821.

[Vandersypen *et al.* 2001] Lieven M.K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, Isaac L. Chuang, 2001. « Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance », Nature, Vol. 414, pp. 883-887.

[Vion 2003] Denis Vion, 2003. « Josephson quantum bits based on a cooper pair box », Quantum entanglement and information processing, session LXXIX (Proceedings of the Les Houches Summer School), 41 pages.

[Wecker *et al.* 2015] Dave Wecker, Matthew B. Hastings, Nathan Wiebe, Bryan K. Clark, Chetan Nayak, Matthias Troyer, 2015. « Solving strongly correlated electron models on a quantum computer », Phys. Rev. A 92, 062318, 25 pages.

[Wittek 2016] Peter Wittek, 2016. « Quantum Machine Learning: What Quantum Computing Means to Data Mining », Elsevier, ISBN-13 : 978-0128100400, 176 pages.

[Zadeh 1965] Lotfi A. Zadeh, 1965. « Fuzzy sets », Information and Control, Elsevier, 16 pages.

[Zeh 1970] Heinz-Dieter Zeh, 1970. « On the interpretation of measurement in quantum theory », Foundations of Physics, Volume 1, Issue 1, pp. 69-76.

Références webographiques

[1] J. Russell, « IBM Quantum Update: Q System One Launch, New Collaborators, and QC Center Plans,» 10 January 2019. [En ligne]. Available: https://www.hpcwire.com/2019/01/10/ibm-quantum-update-q-system-one-launch-new-collaborators-and-qc-center-plans/. [Accès le 30 October 2019].

[2] E. Pednault, J. Gunnels, D. Maslov et J. Gambetta, « On “Quantum Supremacy” | IBM Research Blog,» 21 October 2019. [En ligne]. Available: https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/. [Accès le 30 October 2019].

[3] S. Shankland, « IBM's new 53-qubit quantum computer is its biggest yet - CNET,» 18 September 2019. [En ligne]. Available: https://www.cnet.com/news/ibm-new-53-qubit-quantum-computer-is-its-biggest-yet/. [Accès le 30 October 2019].

[4] Smite-Meister, « Bloch sphere, a geometrical representation of a two-level quantum system.,» 30 January 2009. [En ligne]. Available: https://en.wikipedia.org/wiki/File:Bloch_sphere.svg. [Accès le 19 November 2019].

[5] J. Marrow, «diagrams - What's the best way to represent logic gates? - TeX - LaTeX Stack Exchange,» 22 Novembre 2019. [En ligne]. Available: https://tex.stackexchange.com/questions/517597/whats-the-best-way-to-represent-logic-gates. [Accès le 18 Janvier 2020].

[6] «Produit de Kronecker — Wikipédia,» [En ligne]. Available: https://fr.wikipedia.org/wiki/Produit_de_Kronecker. [Accès le 20 Janvier 2020].

[7] Trenar3, «File:Q fourier nqubits.png - Wikipedia,» 14 Février 2018. [En ligne]. Available: https://upload.wikimedia.org/wikipedia/commons/6/61/Q_fourier_nqubits.png. [Accès le 20 Janvier 2020].

[8] I. Quantique, «Products - ID Quantique,» [En ligne]. Available: https://www.idquantique.com/random-number-generation/products/. [Accès le 12 février 2020].

[9] S. Jordan, «Quantum Algorithm Zoo,» [En ligne]. Available: http://quantumalgorithmzoo.org/. [Accès le 12 Février 2020].

[10] Q. C. Report, «Qubit Quality | Quantum Computing Report,» Quantum Computing Report, [En ligne]. Available: https://quantumcomputingreport.com/scorecards/qubit-quality/. [Accès le 17 Février 2020].

[11] T. Simonite, «Alphabet Has a Second, Secretive Quantum Computing Team | WIRED,» WIRED, 31 Janvier 2020. [En ligne]. Available: https://www.wired.com/story/alphabet-second-secretive-quantum-computing-team/. [Accès le 18 Février 2020].

[12] S. Pellerano, «Intel and QuTech Unveil Details of First Cryogenic Quantum Computing Control Chip, ‘Horse Ridge’ | Intel Newsroom,» 18 Février 2020. [En ligne]. Available: https://newsroom.intel.com/news/intel-qutech-unveil-details-first-cryogenic-quantum-computing-control-chip-horse-ridge/. [Accès le 19 Février 2020].

[13] T. Simonite, «Microsoft Is Taking Quantum Computers to the Cloud | WIRED,» 04 Novembre 2019. [En ligne]. Available: https://www.wired.com/story/microsoft-taking-quantum-computers-cloud/. [Accès le 18 Février 2020].

[14] J. Sanders, «TechRepublic: D-Wave announces 5,000-qubit fifth generation quantum annealer | D-Wave Systems,» 25 Septembre 2019. [En ligne]. Available: https://www.dwavesys.com/media-coverage/techrepublic-d-wave-announces-5000-qubit-fifth-generation-quantum-annealer. [Accès le 19 Février 2020].

Lexique

Adiabatique : sans échange de chaleur.

API : « Application Programming Interface » : Ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

Bit : « Binary digit », vaut 0 ou 1.

Booléen ou variable booléenne : variable pouvant prendre les valeur « vrai » ou « faux » souvent représentées par les nombres 0 et 1.

Circuit quantique : succession de portes quantiques.

Décohérence : perte des propriétés quantiques d'un système due aux interactions avec l'environnement extérieur.

ECTS : « European Credits Transfer System », unité de mesure du volume de travail à fournir pour atteindre un niveau d'étude.

Endianisme : ordre des chiffres dans un système de numération positionnel.

État propre : dans un état superposé, il s'agit d'un état de base ; un état possible pour une particule quantique avant superposition ou après mesure.

GitHub : plateforme d'hébergement de codes source de logiciels versionnés avec l'outil git.

Glisser-déposer : appuyer sur le clic gauche de la souris lors du survol d'un élément pour « l'attraper », puis déplacer la souris, puis relâcher le clic de la souris pour « déposer » l'élément dans la zone où se trouve le curseur.

Intrication quantique ou enchevêtrement quantique : lien existant entre deux particules quantiques ; connaissant l'état de l'une, on peut déduire l'état de l'autre.

Octet : groupe de 8 bits.

Ordinateur quantique ou processeur quantique : processeur capable de d'appliquer des portes quantiques à des qubits.

Partition : représentation temporelle graphique des qubits et des portes d'un circuit quantique.

Pio : « Pébioctet », péta-binaire-octet, 2^{50} octets.

Porte quantique : élément de base d'un circuit quantique opérants sur des qubits.

Python : langage de programmation interprété de haut niveau créé en 1991 par Guido van Rossum.

Réduction du paquet d'onde : mesure de l'état d'un système quantique (la mesure détruit l'état quantique du système réduisant son état à son état mesuré).

Registre quantique : groupe de qubits sur lesquels des opérations quantiques peuvent être effectuée.

SDK : « Software Development Kit » : Kit de développement logiciel. Ensemble d'outils logiciels destinés aux développeurs, facilitant le développement d'un logiciel sur une plateforme donnée.

Superposition quantique : combinaison des états possibles, associés à une probabilité, dans lesquels peut se trouver une particule quantique.

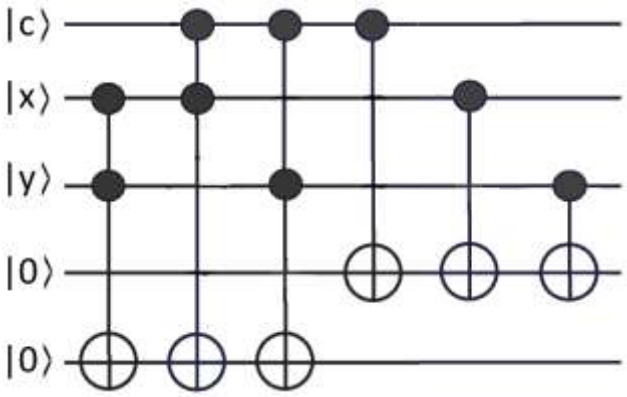
Suprématie quantique : limite à partir de laquelle un ordinateur quantique devient plus puissant que tout ordinateur classique (autre définition : seuil à partir duquel un ordinateur classique ne peut plus simuler un ordinateur quantique).

Taux d'erreur : nombre de résultats corrects donnés divisé par le nombre d'essais.

Transmon : abréviation de « transmission line shunted plasma oscillation qubit » : type de qubit de charge supraconducteur conçu pour réduire la sensibilité au bruit de charge.

ANNEXES

Annexe 1 : Circuit quantique d’addition de deux qubits



Ce circuit additionne les qubits x et y et la retenue c dans les deux autres qubits qui représentent la somme et la retenue suivante.

Annexe 2 : Matrice du circuit quantique d'addition de deux qubits

(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0)

Annexe 3 : Fichier « qelib1.inc » implémentant les portes quantiques à partir des portes U et CX

Source : https://github.com/Qiskit/openqasm/blob/master/examples/generic/qelib1.inc

```
// Quantum Experience (QE) Standard Header
// file: qelib1.inc

// --- QE Hardware primitives ---

// 3-parameter 2-pulse single qubit gate
gate u3(theta,phi,lambda) q { U(theta,phi,lambda) q; }
// 2-parameter 1-pulse single qubit gate
gate u2(phi,lambda) q { U(pi/2,phi,lambda) q; }
// 1-parameter 0-pulse single qubit gate
gate u1(lambda) q { U(0,0,lambda) q; }
// controlled-NOT
gate cx c,t { CX c,t; }
// idle gate (identity)
gate id a { U(0,0,0) a; }

// --- QE Standard Gates ---

// Pauli gate: bit-flip
gate x a { u3(pi,0,pi) a; }
// Pauli gate: bit and phase flip
gate y a { u3(pi,pi/2,pi/2) a; }
// Pauli gate: phase flip
gate z a { u1(pi) a; }
// Clifford gate: Hadamard
gate h a { u2(0,pi) a; }
// Clifford gate: sqrt(Z) phase gate
gate s a { u1(pi/2) a; }
// Clifford gate: conjugate of sqrt(Z)
gate sdg a { u1(-pi/2) a; }
// C3 gate: sqrt(S) phase gate
gate t a { u1(pi/4) a; }
// C3 gate: conjugate of sqrt(S)
gate tdg a { u1(-pi/4) a; }

// --- Standard rotations ---
// Rotation around X-axis
gate rx(theta) a { u3(theta,-pi/2,pi/2) a; }
// rotation around Y-axis
gate ry(theta) a { u3(theta,0,0) a; }
// rotation around Z axis
gate rz(phi) a { u1(phi) a; }

// --- QE Standard User-Defined Gates  ---
```

```
// controlled-Phase
gate cz a,b { h b; cx a,b; h b; }
// controlled-Y
gate cy a,b { sdg b; cx a,b; s b; }
// controlled-H
gate ch a,b {
h b; sdg b;
cx a,b;
h b; t b;
cx a,b;
t b; h b; s b; x b; s a;
}
// C3 gate: Toffoli
gate ccx a,b,c
{
  h c;
  cx b,c; tdg c;
  cx a,c; t c;
  cx b,c; tdg c;
  cx a,c; t b; t c; h c;
  cx a,b; t a; tdg b;
  cx a,b;
}
// controlled rz rotation
gate crz(lambda) a,b
{
  u1(lambda/2) b;
  cx a,b;
  u1(-lambda/2) b;
  cx a,b;
}
// controlled phase rotation
gate cu1(lambda) a,b
{
  u1(lambda/2) a;
  cx a,b;
  u1(-lambda/2) b;
  cx a,b;
  u1(lambda/2) b;
}
// controlled-U
gate cu3(theta,phi,lambda) c, t
{
  // implements controlled-U(theta,phi,lambda) with  target t and control c
  u1((lambda-phi)/2) t;
  cx c,t;
  u3(-theta/2,0,-(phi+lambda)/2) t;
  cx c,t;
  u3(theta/2,phi,0) t;
}
```