Code quantique et tolérance aux fautes

IQ01 - Automne 2021

Groupe 2 :
Téva Ducoté (GI01)
François Guichard (GI01)
Matthieu Guiramand(GI05)
Thomas Leymonerie (GI03)
Pierre Pouliquen (GI05)

Rapport

Table des matières

1 Introduction

Toute branche de l'informatique possède son lot d'erreur. Afin d'avoir les meilleures performances et les résultats les plus précis possibles, il est de l'intérêt de l'ingénieur informaticien de réduire ses erreurs. Il en est de même pour l'informatique quantique. En effet, l'informatique quantique comme l'informatique classique se réalise grâce à des modèles physiques. Ces modèles n'étant pas parfait, dû notamment à des champs magnétiques, l'usure des composants physiques, ils se créaient des erreurs. Bien sûr, il en va de même pour l'informatique et nous allons voir les moyens pour atténuer l'impact de ces erreurs.

2 Principe des codes correcteurs

Un code correcteur a pour but de corriger les erreurs lors de la lecture, ou de la transmission de données notamment sur de longs trajets. Ces erreurs sont causées par des perturbations telles que la distorsion, la présence de bruit thermique (agitation des électrons) ou impulsif (surtension). En règle générale, une communication de données possède un taux d'erreur de 10^{-7} à 10^{-5}. Il est donc important d'endiguer ce problème afin de garantir l'intégrité de l'information. Ainsi des systèmes de détection d'erreurs ont été développées pour réduire voire éliminer ces erreurs. Il en existe deux types : les codes autocorrecteurs et autovérificateurs.[7]
Certaines techniques utilisent la redondance d'informations pour assurer un bon transfert (cf méthode naïve *Repetition code* 4.1). C'est le principe choisi pour les QR codes.
Ceci est d'autant plus vrai en informatique quantique pour lequel on utilise des phénomènes physiques à l'échelle microscopique. Il s'avère que les qubits sont très instables. La moindre perturbation peut influencer les résultats finaux. Contrairement aux ordinateurs classiques, les qubits peuvent subir plusieurs types d'erreurs. Il peut s'agir :

- d'un bit-flip représenté par la porte de Pauli X, on obtient $X|0\rangle = |1\rangle$ et $X|1\rangle = |0\rangle$
- un phase flip représenté par la porte de Pauli Z, on obtient $Z|0\rangle = |0\rangle$ et $Z|1\rangle = -|1\rangle$
- ou une combinaison de ces deux erreurs représenté par la porte de Pauli Y tel que $Y = iXZ$.

Il est donc nécessaire d'être capable de distinguer les différentes erreurs pour chaque état du qubit.

3 Définitions

Dans cette section, nous allons définir ce qu'est un code quantique puis ce qu'est la tolérence aux fautes.

Code quantique : La correction d'erreur quantique (QEC) est utilisée en informatique quantique pour protéger les informations quantiques des erreurs dues à la décohérence et à d'autres bruits quantiques. La correction d'erreur quantique est essentielle si l'on veut obtenir un calcul quantique tolérant aux pannes qui peut traiter non seulement le bruit sur les informations quantiques stockées, mais également les portes quantiques défectueuses, la préparation quantique défectueuse et les mesures défectueuses. Le code quantique s'applique donc à réduire les erreurs quantique dû à la décohérence.[2]

Tolérence aux fautes : La tolérance aux fautes est la propriété qui permet à un système de continuer à fonctionner correctement en cas de défaillance d'un ou plusieurs défauts au sein de certains de ses composants. Si sa qualité de fonctionnement diminue du tout, la diminution est proportionnelle à la gravité de la défaillance, par rapport à un système conçu naïvement, dans lequel même une petite défaillance peut provoquer une panne totale. La tolérance aux pannes est particulièrement recherchée dans les systèmes à haute disponibilité ou critiques. La capacité de maintenir la fonctionnalité lorsque des parties d'un système tombent en panne est appelée dégradation progressive. [11]

La décohérence quantique : Après la manipulation des phénomènes quantiques, il est nécessaire de les transposer aux règles de la physique classique. Cependant cette transition n'est pour le moment expliquer que par une théorie celle de la décohérence quantique. L'un des problème majeur de la physique quantique est le phénomène de superposition, qui est complètement inconnue au niveau macroscopique. Mais aussi la théorie quantique tient compte du phénomène de non-observabilité, puisque toutes observations provoque un effondrement de la fonction d'onde et perturbe l'état. La décohérence quantique donne lieu alors au postulat n°5 : la réduction du paquet d'onde. Cependant, elle rentre en contradiction avec le postulat n°6 : l'équation de Schrödinger. Étant la théorie la plus satisfaisante à ce jour, la communauté scientifique tente à prouver que la Réduction des paquets d'ondes n'est qu'une conséquence de l'équation de Schrödinger, et non une contradiciton. [1]

4 Implantation, circuit correcteur

4.1 Repetition code

Le code de répétition joue sur l'envoie en plusieurs exemplaires du même état au destinataire en espérant qu'il y a suffisamment d'états corrects afin de garantir l'intégrité de l'état, sinon il faut corriger. On peut considérer ce modèle comme une version généralisé du "three-qubit bit-flip code", ou à l'inverse le three-qubit bit-flip code comme la version la plus simple du repetition code.

Tous systèmes informatiques aussi complexes ou simples soient-ils, peuvent commettre des erreurs durant leurs opérations suite à l'apparition de bruits plus ou moins inévitables. Pour les ordinateurs classiques des algorithmes de correction ont été développé pour faire face à ce problème. Ces algorithmes sont si efficaces que leurs applications permettent de complètement palier aux bruits qui peuvent survenir. L'idée principale derrière tout ça est de protéger le message du bruit. L'exemple le plus simple est le three-bit repetition code. La logique est d'appliquer un encodage sur un bit que nous souhaitons transmettre, si l'un des bits a été modifié, nous pouvons alors effectuer un vote par majorité pour restaurer le bit modifié. Nous pouvons facilement en déduire que le three-bit repetition code appartient à un ensemble de code linéaire. Qu'il est possible d'agrandir notre vecteur encodé à k élément. Établissons un exemple : Supposons que nous souhaitons encoder un ensemble de bit noté v ayant k bits en utilisant n bits. La donnée peut être représenté comme un vecteur de k dimensions binaire.

$v_1 = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_k \end{pmatrix}$

Pour un code linéaire, l'encodage de la donnée peut être traduit par Gv tel que G soit une matrice n*k, indépendante de v. G est appelé la matrice génératrice du code. Ses colonnes forment une base pour les k dimensions codant des sous espace de n dimensions de vecteur binaire, représentant alors les codewords ou mot de passe. [16] Prenons l'exemple d'un encodage sur 3 bits. Nous souhaitons recopier 1 bits sur les 3, nous en déduisons la logique de construction de G comme : G = $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

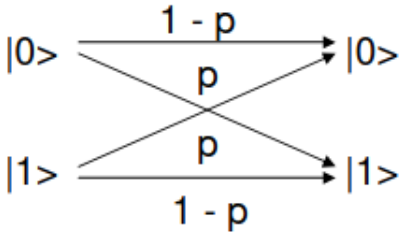
G[0] = $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ et G[1] = $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ Dans notre cas, nous avons un [3,1] code, c'est à dire, que nous avons besoin de 3 bits pour encoder 1 bit d'information, que nous pouvons voir de manière générale comme un code [n,k]. Pour établir une détection d'erreur et une correction, nous avons besoin d'une matrice de bit de parité, que l'on nommera P. La matrice doit être de dimension (n-k)*n, et de rang maximal n-k avec PG = 0. Comme les codewords sont de la forme $x = Gv$, nous avons $Px = PGv = 0v = 0$, donc P annule n'importe quel codewords. Si une erreur e surgit, alors nous devons prendre en considération que le codeword soit modifié et donc $x' = x + e$. Il est alors facile de vérifier que $Px' = Pe$, dont nous appelerons cette erreur le syndrome. Ceux ci nous donne des informations sur la position et le type d'erreur e, nous pouvons alors facilement corriger cette erreur.

4.2 The three-qubit bit-flip code

Bit flip Error : Une erreur classique est l'inversion d'un bit dans le code, soit un bit passe de 0 à 1 ou vice versa.[13]
Cette méthode utilise le principe d'intrication afin de détecter des potentielles erreurs.

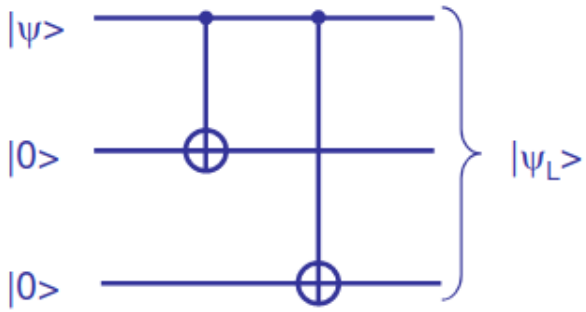
4.2.1 Encodage

Considérons que la probabilité qu'une erreur de type bit flip apparaisse soit de p, alors l'arbre binaire quantique symétrique suivant :



3 erreur	2 erreur	1 erreurs	0 erreurs
p^3	$p^2(1-p)$	$p(1-p)^2$	$(1-p)^3$

Afin de palier à ce problème, l'utilisation de 2 autres qubits permet de détecter ce type d'erreur. Nous définissons tout d'abord l'encodage suivant :



Soit $|\psi\rangle = c_0|0\rangle + c_1|1\rangle \rightarrow |\psi_L\rangle = c_0|0_L\rangle + c_1|1_L\rangle$
Avec :
• $|0\rangle \rightarrow |0_L\rangle = |000\rangle$
• $|1\rangle \rightarrow |1_L\rangle = |111\rangle$

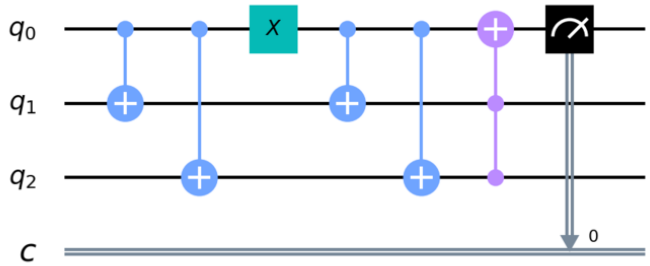
Nous pouvons donc en déduire :
$|\psi\rangle = c_0\,|0_L\rangle + c_1\,|1_L\rangle = c_0\,|000\rangle + c_1\,|111\rangle$

4.2.2 Détection d'erreur

Il existe alors 4 types d'erreurs pouvant intervenir sur chacun des trois Qubit :
•$P_1 = |000\rangle\langle 000| + |111\rangle\langle 111|$ Il n'y a auncune erreur
•$P_2 = |100\rangle\langle 100| + |011\rangle\langle 011|$ erreur sur le Qubit n°1
•$P_3 = |010\rangle\langle 010| + |101\rangle\langle 101|$ erreur sur le Qubit n°2
•$P_4 = |001\rangle\langle 001| + |110\rangle\langle 110|$ erreur sur le Qubit n°3
Lorsqu'il n'y a pas d'erreur de bit flip, $\psi|P_1|\psi = 1$ et $\psi|P_{2,3,4}|\psi = 0$, dans le cas contraire, $\psi|P_{2,3,4}|\psi = 1$. On peut en déduire que la mesure de $P_1 P_2 P_3 et P_4$ nous permet de déduire si une erreur de bit flip s'est produit ou non. Les opérateurs de projection ne changent pas l'état du Qubit logique. Ils ne font simplement que décrire quel Qubit a été "fliped". La mesure de ses opérateurs ne donne aucune indication sur les valeurs de α ou de β. Mais seulement quel Qubit doit être corrigé.

Supposons que le Qubit n°1 a subit un flip erreur dans ce cas l'état $|\psi\rangle$ est : $|\psi\rangle = c_0 |100\rangle + c_1 |011\rangle$ Alors $\psi|P_1|\psi = 1$, ce qui vérifie qu'il y a bien eu un bit flip sur le Qubit n°1.

4.2.3 Correction



Suivant l'erreur du bit flip, il faut placer un opérateur NOT sur le Qubit cible.
Nous pouvons résumer la fonction de l'opérateur de Toffoli comme un vote par la majorité suivant les états.

4.2.4 Défaut

Nous pouvons souligner un certain nombre d'inconvénient ou de limitation lié à l'architecture :

•Un défaut que l'on peut avancer sur cette algorithme est que l'on ne peut l'appliquer uniquement si nous avons la certitude que le registre à l'origine se trouve à l'état 0 ou 1.

•Il ne prend pas en compte les erreurs de phase. Cependant le programme ne prend pas en compte les erreurs de phases. Mais un autre algorithme permet corriger ce type d'erreur.

•Possible que les bits d'encodages subissent eux aussi une erreur de flip. Qui entraînerait à un faussement total du résultat.

4.3 Phase/Sign flip code

4.3.1 Présentation

Dans un ordinateur classique, les bits inversés sont les seules types d'erreurs. Cependant, en informatique quantique, il peut se produire une autre erreur : l'erreur d'inversion de phase.[14]
Ce type d'erreur affecte la phase du Qubit, c'est donc l'application d'une porte Z.
On obtient alors :$Z|0\rangle = |0\rangle$ et $Z|1\rangle = -|1\rangle$.

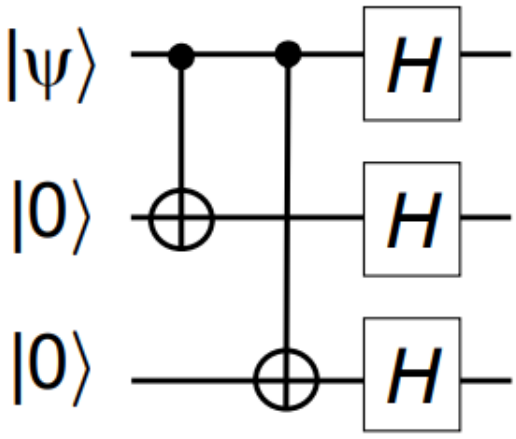
On se rend compte que $Z|+\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}} = |-\rangle$, $Z|-\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}} = |+\rangle$.
Et que, $|+\rangle = H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ et $|-\rangle = H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Alors, l'inversion de phase peut se ramener à une inversion de bit avec les équations suivantes :

$$ZH|0\rangle = Z|+\rangle = |-\rangle$$
$$ZH|1\rangle = Z|-\rangle = |+\rangle$$

Donc, en rajoutant des portes Hadamard à l'entrée et en sortie de l'Oracle de l'erreur, nous pouvons traiter l'erreur de la même manière que le Bit Flip Code avec deux autres Qubits auxiliaires et deux portes CNOT couplés avec le premier Qubit et les Qubits auxiliaires.

4.3.2 Encodage

On encode ce l'état du Qubit afin d'obtenir : $|\phi\rangle = a|0\rangle + b|1\rangle \rightarrow a|+++\rangle + b|---\rangle = a|0_L\rangle + b|1_L\rangle$[15]. Cela donne le circuit suivant :

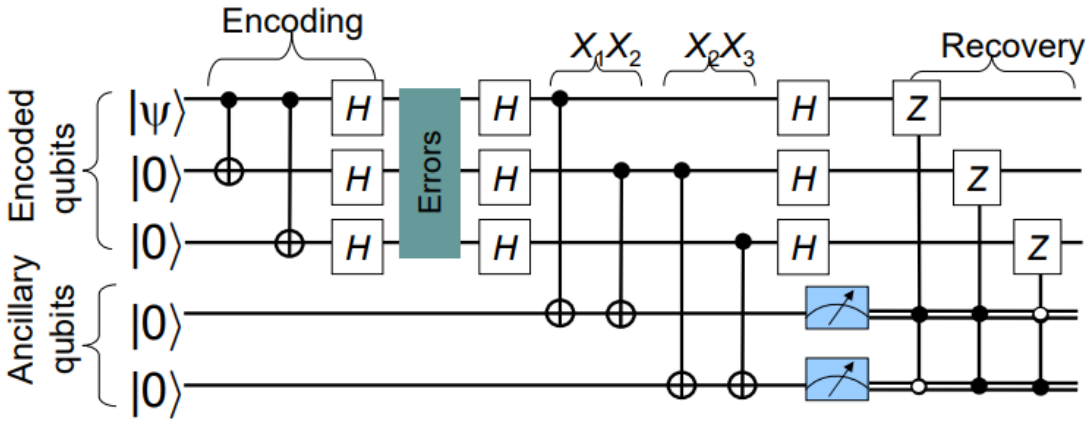


4.3.3 Détection de l'erreur

Pour déterminer le Qubit qui a subit une inversion de phase, on effectue des mesures de syndrome. Pour cela, on va utiliser 2 Qubits auxiliaires liés à 4 portes CNOT. Nous allons donc implémenter les deux calculs suivants :

$$X_1X_2 = \begin{cases} +1 \; si \; Qubit_1 = Qubit_2 \\ 0 \; si \; Qubit_1 \neq Qubit_2 \end{cases} \qquad X_2X_3 = \begin{cases} +1 \; si \; Qubit_2 = Qubit_3 \\ 0 \; si \; Qubit_2 \neq Qubit_3 \end{cases}$$

Ensuite, des portes Z contrôlées par les Qubits arbitraires sont appliquées aux Qubits principaux en fonction du syndrome déterminé. On se retrouve donc avec le circuit suivant :



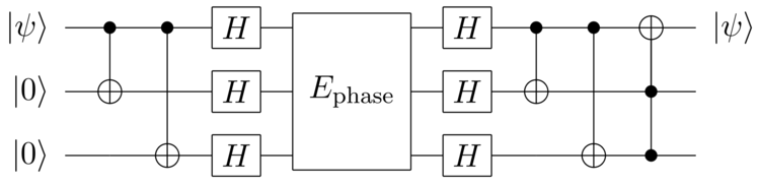
Pour illustrer ce circuit, nous allons voir 3 exemples :
•Dans le cas où il n'y a pas d'erreur ($|+++\rangle$), $X_1X_2 = +1$ et $X_2X_3 = +1$ donc il n'y a aucune correction qui est effectuée.
•Dans le cas où il y a une erreur sur le Qubit 2 ($|+-+\rangle \rightarrow |+++\rangle$), $X_1X_2 = 0$ et $X_2X_3 = 0$ donc il y a une correction qui est effectuée avec une porte Z sur le deuxième Qubit.
•Dans le cas où il y a une erreur sur le Qubit 3 ($|--+\rangle \rightarrow |---\rangle$), $X_1X_2 = +1$ et $X_2X_3 = 0$ donc il y a une correction qui est effectuée avec une porte Z sur le troisième Qubit.

La correction ne peut se faire uniquement si il n'y a qu'une erreur. On peut le montrer facilement avec un exemple.
Si il y a une inversion de phase sur les Qubit 1 et 2, par exemple $|+++\rangle \rightarrow |--+\rangle$. Par la mesure des syndromes on aurait $X_1X_2 = +1$ et $X_2X_3 = 0$, ce qui serait pris pour une erreur sur le Qubit 3. Et donc, une porte Z serait appliqué au Qubit 3 et on obtiendrait $|---\rangle$.

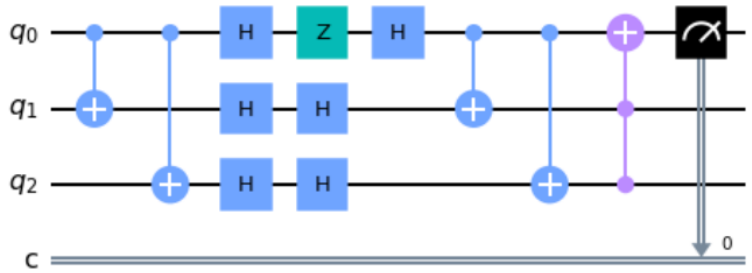
La probabilité d'echec du circuit est mesurée par la formule suivante : $3p^2(1-p)+p^3 < p$ (pour $p < \frac{1}{2}$).

4.3.4 Expérimentation avec Qiskit

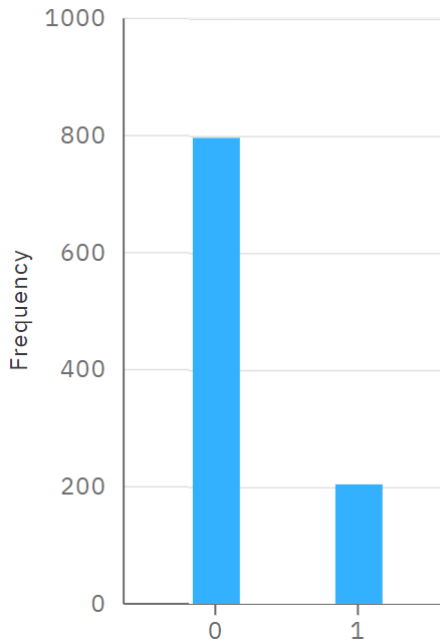
On va donc expérimenter le code correction d'erreur de phase en python avec Qiskit et un provider d'IBM Quantum. Pour ce faire nous allons implémenter le circuit suivant :



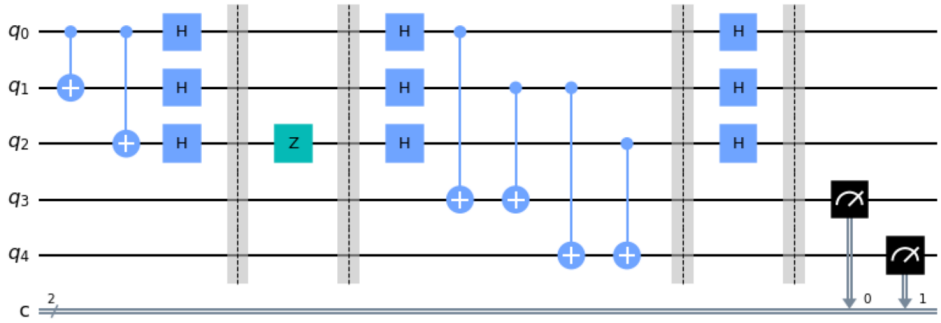
Pour simuler une erreur de phase sur le premier Qubit on place une porte Z, ce qui nous donne :



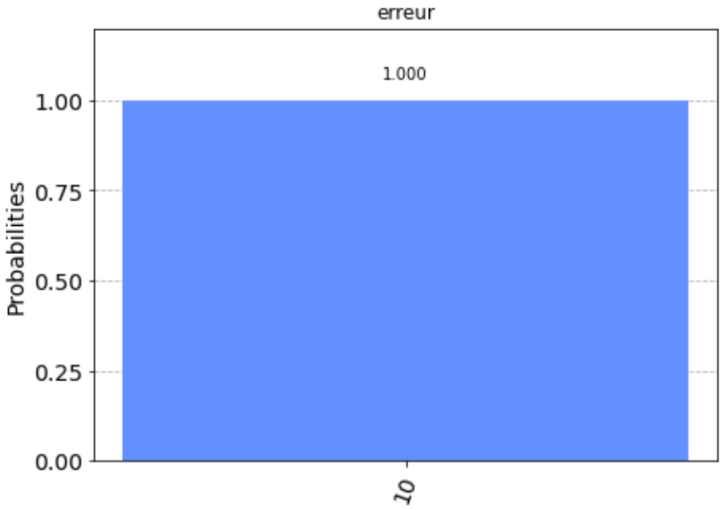
En lançant le circuit sur un ordinateur d'IBM on obtient un taux d'inversion de phase d'environ 20



On peut aussi, en implémentant le circuit avec la mesure des syndromes, retrouver le Qubit qui a subit une inversion de phase. On créé donc le circuit suivant :



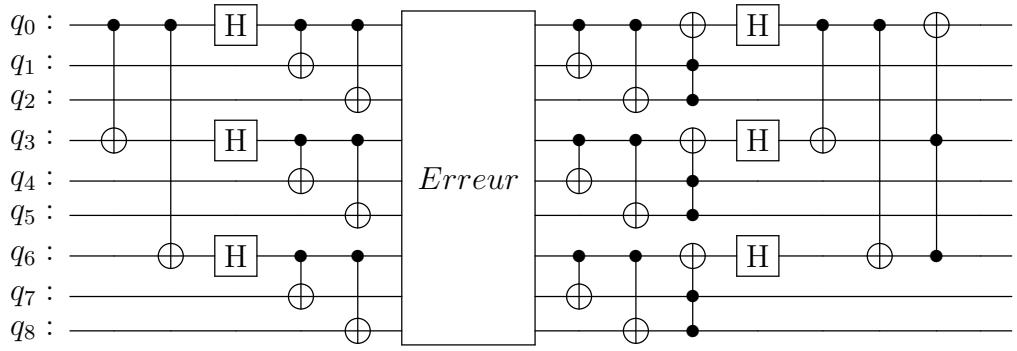
Avec le simulateur de Qiskit, on retrouve le Qubit qui a subit l'inversion de phase :



On remarque que le Qubit erroné est le Qubit 2, on peut alors corriger l'erreur en appliquant une simple porte Z.

4.4 Shor code

Nous avons vu précedement des codes permettant de corriger des erreurs de bit flip ou phase flip, nous allons maintenant voir un code qui peut corriger l'occurence des deux erreurs (ensemble ou non). Ce code, nommé Shor code, permet de corriger un qubit en utilisant 8 qubits de travail. En voici son diagramme :



Le code reprend les structure du bit flip code et du sign flip code. En effet, comme on peut le voir, le groupe $\{q_0, q_3, q_6\}$ subit une correction de phase, tandis que les sous-groupes $\{q_0, q_1, q_2\}, \{q_3, q_4, q_5\}, \{q_6, q_7, q_8\}$ subissent des correction de bit.

4.4.1 Fonctionnement du circuit

Préparation avant une potentiel Erreur
L'état initiale est :

$$|\psi\rangle \otimes |0\rangle^{\otimes 8} = (a|0\rangle + b|1\rangle) \otimes |0\rangle^{\otimes 8}$$

Application des premières portes C-Not :

$$\begin{aligned} CX_{0,3}CX_{0,6}\,|\psi\rangle \otimes |0\rangle^{\otimes 8} &= CX_{0,3}CX_{0,6}(a|0\rangle + b|1\rangle) \otimes |0\rangle^{\otimes 8} \\ &= CX_{0,3}CX_{0,6}(a|000000000\rangle + b|100000000\rangle) \\ &= aCX_{0,3}CX_{0,6}\,|000000000\rangle + CX_{0,3}CX_{0,6}b\,|100000000\rangle \\ &= a\,|000000000\rangle + b\,|100100100\rangle \\ &= a\,|000\rangle^{\otimes 3} + b\,|100\rangle^{\otimes 3} \end{aligned}$$

Application des portes de Hadamards, une sur le premier qubit de chaque sous-groupe.

$$H_0H_3H_6(a\,|000\rangle^{\otimes 3} + b\,|100\rangle^{\otimes 3}) = a\,|+00\rangle^{\otimes 3} + b\,|-00\rangle^{\otimes 3}$$

Application des portes C-Not des sous-groupes :

$$\begin{aligned} CX_{0,1}CX_{0,2}(a\,|+00\rangle^{\otimes 3} + b\,|-00\rangle^{\otimes 3}) &= aCX_{0,1}CX_{0,2}\,|+00\rangle^{\otimes 3} + bCX_{0,1}CX_{0,2}\,|-00\rangle^{\otimes 3} \\ &= a\,\left|+^3\right\rangle|+00\rangle^{\otimes 2} + b\,\left|-^3\right\rangle|-00\rangle^{\otimes 2} \end{aligned}$$

Avec : $\left|+^3\right\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$

et $\left|-^3\right\rangle = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$

Application des dernires porte CX : $a\,\left|+^3\right\rangle^{\otimes 3} + b\,\left|-^3\right\rangle^{\otimes 3}$

C'est maintenant que l'état du registre peux admettre des erreurs de bit flip et phase flip sur le premier qubit qui seront ensuite corrigés. Faisons subir l'erreur de phase et de bit flip à notre vecteur d'état, sachant que ces erreurs sont équivalente à l'application des portes X et Z. On sait que la structure du circuit corrige les erreurs de bit flip dans les trois sous-groupe puis corrige les erreurs de signes sur les trois bits $\{0, 3, 6\}$. En fait, l'état des sous-groupes est tel que le resultat d'une erreurs de signe ne dépend pas du bits qui la subit. En effet :

$$\text{Avec } x \in \{0, 1, 2\}$$

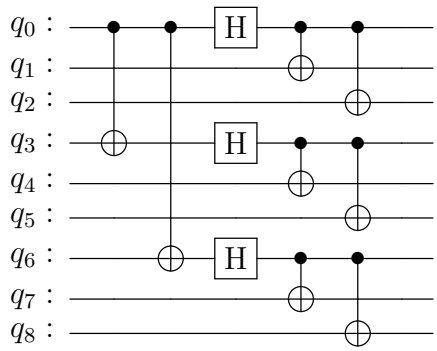
$$\begin{aligned} Z_x(a\left|+^3\right\rangle + b\left|-^3\right\rangle) &= \frac{a}{\sqrt{2}} Z_x(|000\rangle + |111\rangle) + \frac{b}{\sqrt{2}} Z_x(|000\rangle - |111\rangle) \\ &= \frac{a}{\sqrt{2}}(Z_x\,|000\rangle + Z_x\,|111\rangle) + \frac{b}{\sqrt{2}}(Z_x\,|000\rangle - Z_x\,|111\rangle) \\ &= \frac{a}{\sqrt{2}}(|000\rangle - |111\rangle) + \frac{b}{\sqrt{2}}(|000\rangle + |111\rangle) \\ &= a\left|-^3\right\rangle + b\left|+^3\right\rangle \end{aligned}$$

Intéressons nous maintenant à la correction du bit flip dans les sous groupe. C'est cette correction qui est la plus proche des erreurs, car la correction de signe consiste en une correction de bit après avoir utilisé les portes de Hadamard pour transformer les erreurs de signe en erreur de bit. Donc sur chaque sous groupe il y a une correction de bit sur le premier qubit, comme décrit dans la partie bit-flip code.

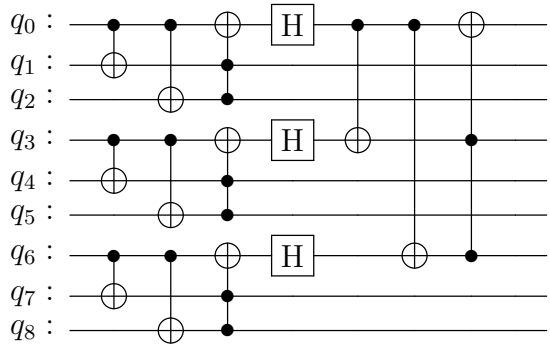
4.4.2 Application avec qiskit

```
from qiskit import execute, QuantumRegister, ClassicalRegister,␣
 ↪QuantumCircuit, BasicAer
from qiskit.visualization import *
import qiskit.quantum_info as qi
backend = BasicAer.get_backend('qasm_simulator')
```

```
def preparation_shor_code():
    preparation_circuit = QuantumCircuit(9)
    #étendre l'etat de q_0 sur q_3 et q_6
    for i in [3,6] : preparation_circuit.cx(0, i)
    #appliquer les portes de hadamard
    for i in [0,3,6] : preparation_circuit.h(i)
    #étendre les etats des q_0/3/6 sur leur sous groupes
    for i in [0,3,6] :
        for j in range(1, 3):
            preparation_circuit.cx(i, i+j)
    #circuit de préparation fini
    print(preparation_circuit.draw(output="latex_source") )
    preparation_gate = preparation_circuit.to_gate()
    preparation_gate.name = "Preparation shore code"
    return preparation_gate
preparation_gate = preparation_shor_code()
```



```
def correction_shor_code():
    correction_circuit = QuantumCircuit(9)
    #on applique la correction de bit flip aux sous-groupes
    for i in [0,3,6] :
        for j in [1,2] : correction_circuit.cx(i, i+j)
        correction_circuit.ccx(i+1, i+2, i)
    #on applique la correction de sign flip au groupe [0,3,6]
    for i in [0, 3, 6] : correction_circuit.h(i)
    for i in [3, 6] : correction_circuit.cx(0, i)
    correction_circuit.ccx(6, 3, 0)
    #circuit de correction fini
    print(correction_circuit.draw(output = "latex_source"))
    correction_gate = correction_circuit.to_gate()
    correction_gate.name = "Correction shor code"
    return correction_gate
correction_gate = correction_shor_code()
```



On veut vérifier que le code de shor ne modifie pas q_0 si il n'y a pas d'erreur.

```
def print_state_vector(circuit):
    """Pour afficher proprement un vecteur d'etat"""
    vector_dict = ector_dict = qi.Statevector.from_instruction(circuit).
 ↪to_dict()
    print('\n'.join(["{} : {}".format(key, vector_dict[key]) for key in␣
 ↪vector_dict.keys()]))
```

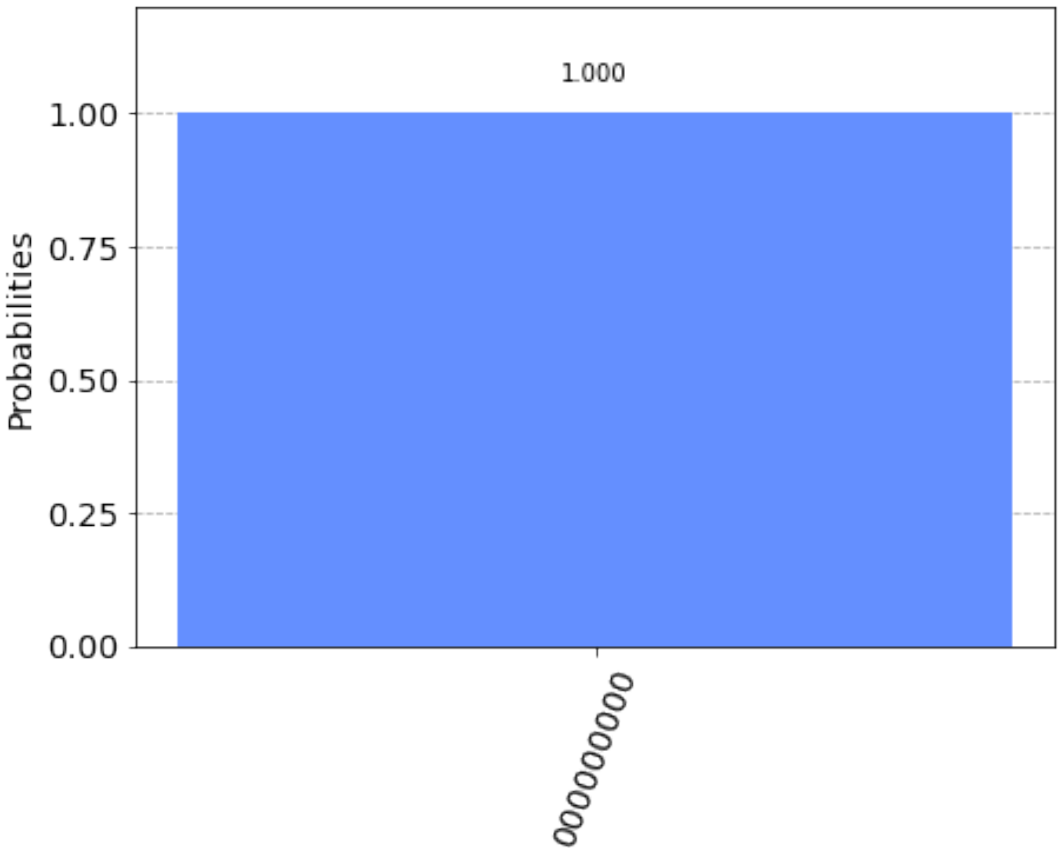
```
#test non modiffication du bit corrigé
circuit_test_1 = QuantumCircuit(9, 9)
print("Etat Initiale")
print_state_vector(circuit_test_1)

circuit_test_1.append(preparation_gate, range(9))
circuit_test_1.append(correction_gate, range(9))
print("\nEtat Finale")
print_state_vector(circuit_test_1)
```

```
Etat Initiale
000000000 : (1+0j)
```

```
Etat Finale
000000000 : (0.9999999999999996+0j)
000000001 : (1.905050747901445e-52+0j)
000001000 : (9.52420782539596e-18+0j)
000001001 : (-1.2183947074714076e-35+0j)
001000000 : (2.3615800235104456e-17+0j)
001000001 : (1.9205844302555546e-35+0j)
001001000 : (6.223285321735372e-19+0j)
001001001 : (1.2631615347037043e-34+0j)
```

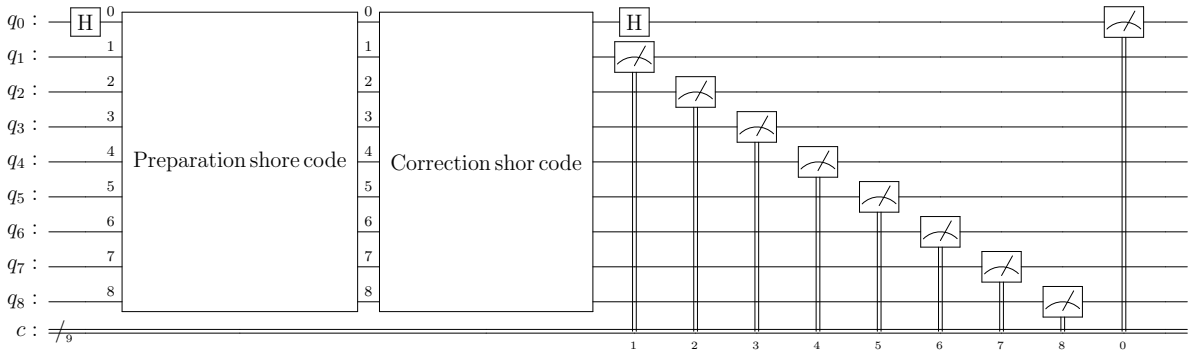
Cependant le vecteur d'etat final n'est pas le même que celui de l'etat initial. Cela est dû aux imprécisions de calcul. On peut cependant verifier qu'experimentalement cela fonctionne bien.

```
circuit_test_1.measure(range(9), range(9))
job = execute(circuit_test_1, backend, shots = 65000)
res = job.result()
plot_histogram(res.get_counts())
```

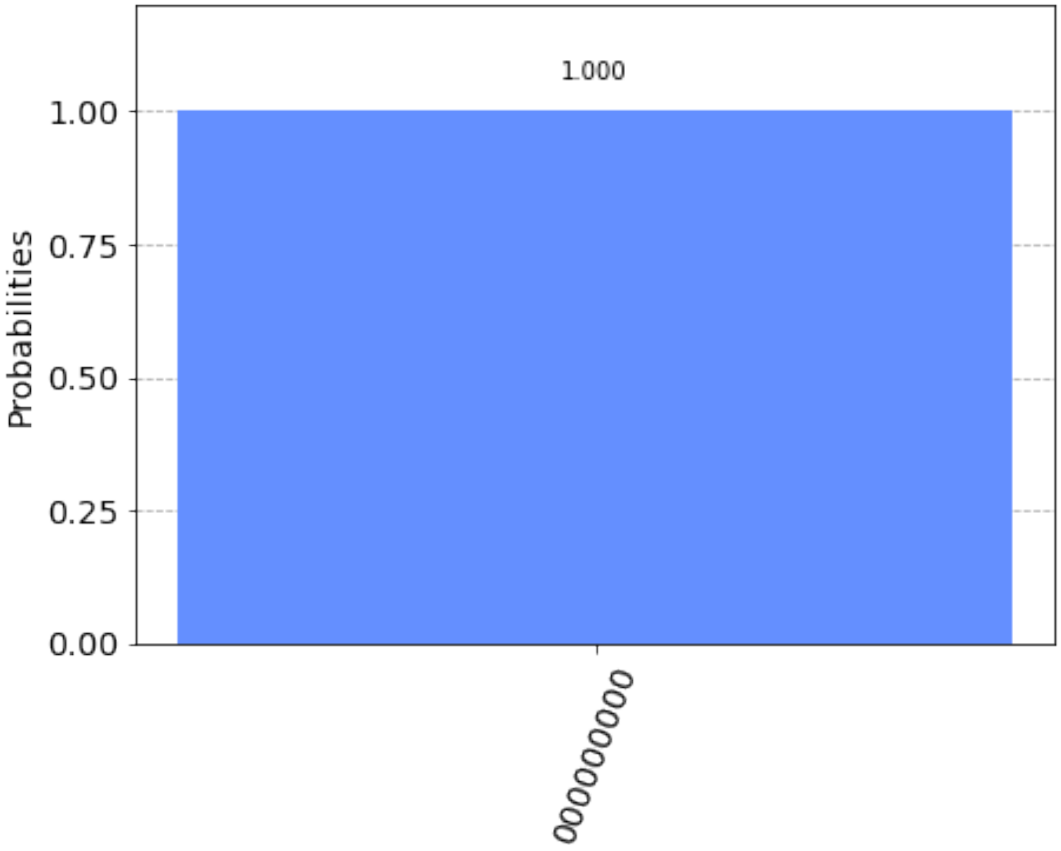


Cependant s'il y avait eu un changement de phase, nous ne l'auriont pas noté avec cette état initiale. On peut donc appliquer des portes de hadamard avant et après le circuit pour vérifier.

```
phase_test = QuantumCircuit(9, 9)
phase_test.h(0)
phase_test.append(preparation_gate, range(9))
phase_test.append(correction_gate, range(9))
phase_test.h(0)
phase_test.measure(range(9), range(9))
print(phase_test.draw(output = "latex_source"))
```




```
job = execute(phase_test, backend)
plot_histogram(job.result().get_counts())
```



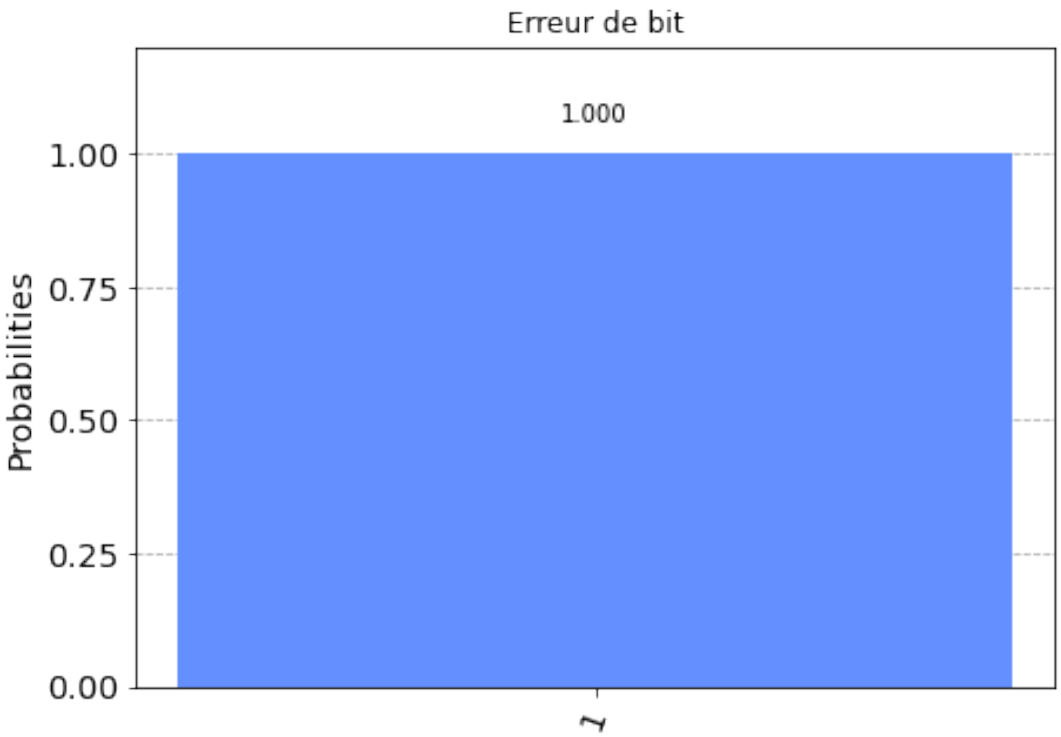
On retrouve bien l'état initial. Comme attendu le code de shor est équivalent à une

opération identité si il n'y a pas d'erreur à corriger.

```
#réalisation porte erreur
def erreur(bits_flip = [], phase_flip = []):
    erreur_circuit = QuantumCircuit(9)
    for i in bits_flip : erreur_circuit.x(i)
    for i in phase_flip : erreur_circuit.z(i)
    erreur_gate = erreur_circuit.to_gate()
    erreur_gate.name = "Erreur"
    return erreur_gate
```

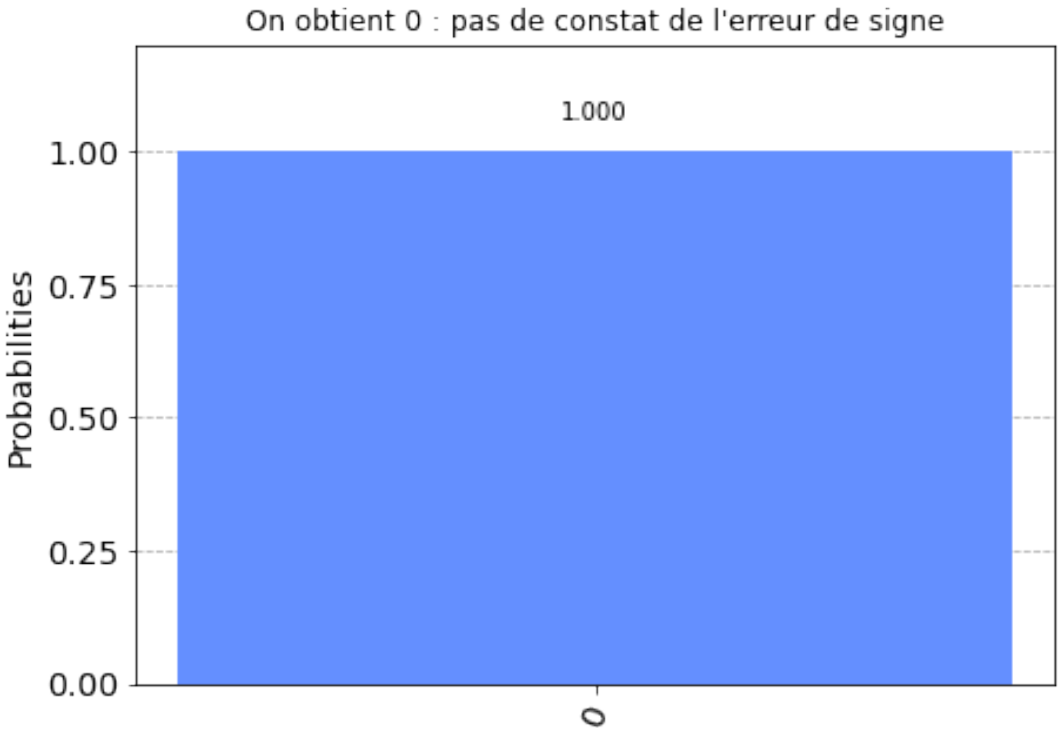
On ne peut pas obtenir un vecteur d'état pleinement satisfaisant, nous ne pouvons donc qu'utiliser la mesure pour constater l'éfficacité du code correcteur. D'abord, constatons que l'on visualise bien les erreurs.

```
#voir l'erreur de bit
cr = QuantumCircuit(9, 1)
cr.append(erreur([0]), range(9))
cr.measure(0, 0)
job = execute(cr, backend)
plot_histogram(job.result().get_counts(), title = "Erreur de bit")
```



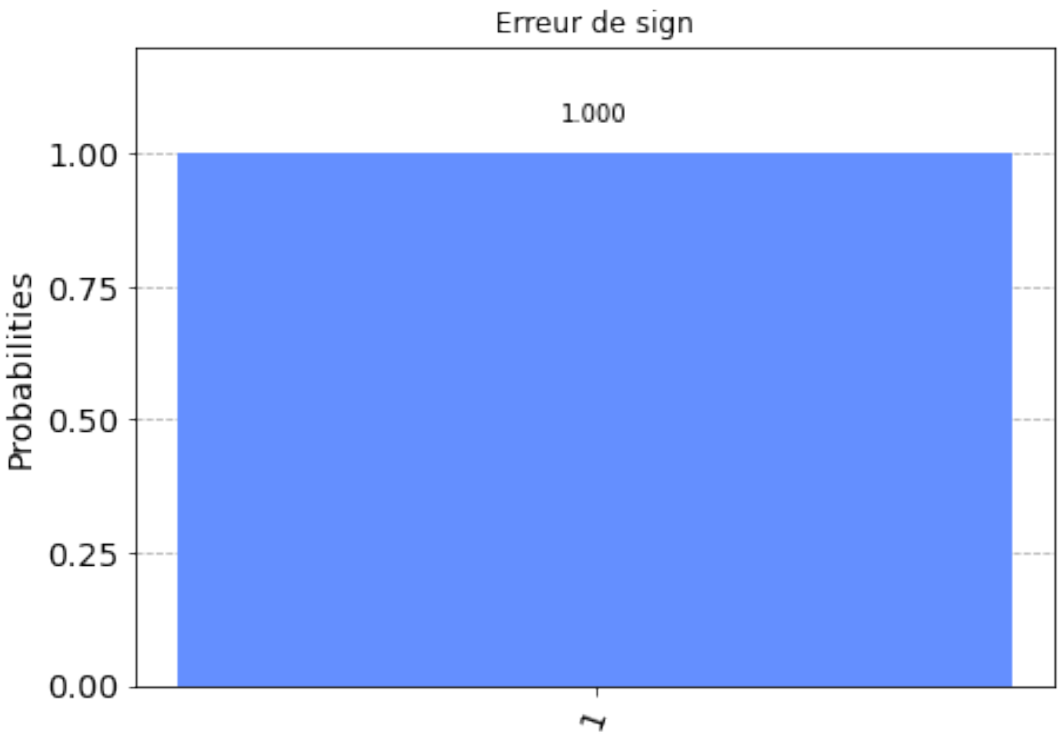
On constate bien l'erreur de bit, cependant on ne constaterait pas l'erreur de signe.

```
#ne pas voir l'erreur de sign
cr = QuantumCircuit(9, 1)
cr.append(erreur(phase_flip = [0]), range(9))
cr.measure(0, 0)
job = execute(cr, backend)
plot_histogram(job.result().get_counts(), title = "On obtient 0 : pas␣
 ↪de constat de l'erreur de signe")
```



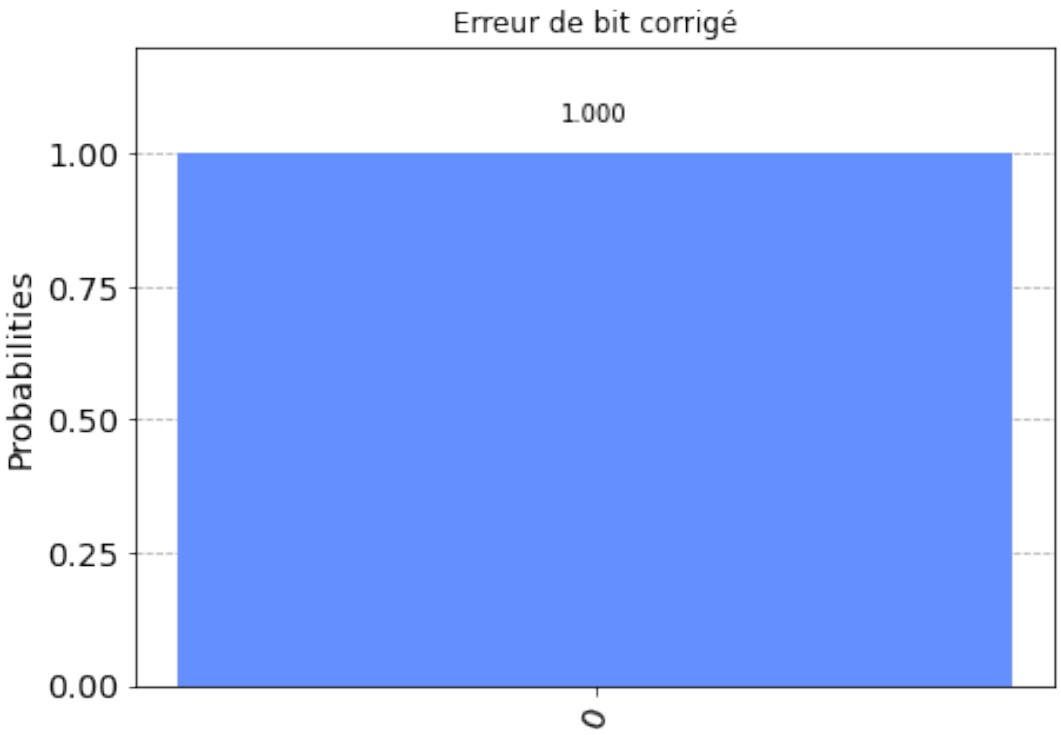
Pour constater l'erreur de signe, on applique un changement de base, on va dans la base (|+>,|->). En effet on a, HZH = X, donc une erreur de signe sera équivalente à une erreur de bit et sera constatable.

```
#voir l'erreur de sign
cr = QuantumCircuit(9, 1)
#Changer de base
cr.h(0)
#realiser l'erreur
cr.append(erreur(phase_flip = [0]), range(9))
#retourner dans le base de calcule
cr.h(0)

#mesurer l'erreur
cr.measure(0, 0)
job = execute(cr, backend)
plot_histogram(job.result().get_counts(), title = "Erreur de sign")
```

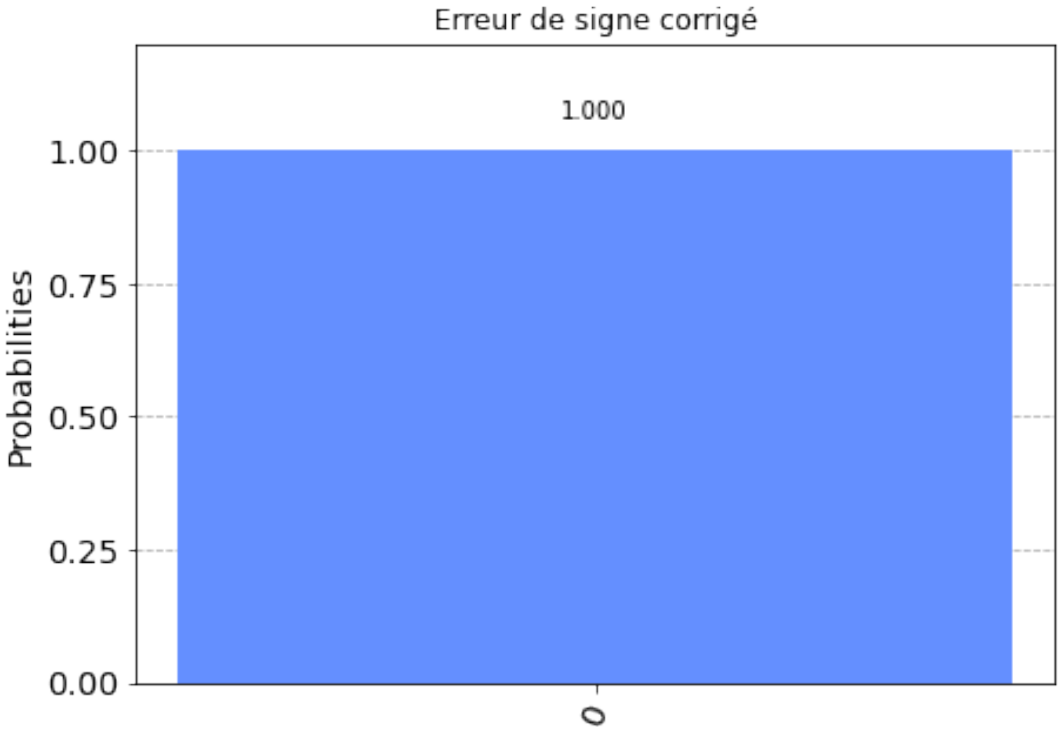


Ainsi pour constater la correction des erreurs de bit et de signe nous devrons réaliser deux circuits, avec et sans changement de base.

```
#Visualisation de la correction de bit
circuit = QuantumCircuit(9,1)

#initialisation du code correctif
circuit.append(preparation_gate, range(9))

#application de l'erreur (de bit et de phase)
circuit.append(erreur([0], [0]), range(9))

#fin du code correctif
circuit.append(correction_gate, range(9))

#mesure du resultat
circuit.measure(0, 0)
job = execute(circuit, backend)
plot_histogram(job.result().get_counts(), title = "Erreur de bit␣
 ↪corrigé")
```

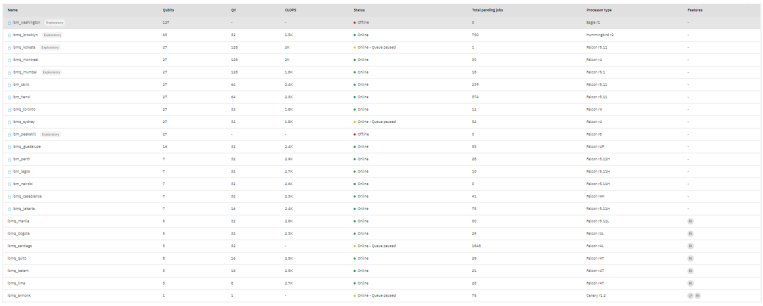



```
#Visualisation de la correction de signe
circuit = QuantumCircuit(9,1)

#changement de base
circuit.h(0)

#initialisation du code correctif
circuit.append(preparation_gate, range(9))

#application de l'erreur (de bit et de phase)
circuit.append(erreur([0], [0]), range(9))

#fin du code correctif
circuit.append(correction_gate, range(9))

#retour en base de calcul
circuit.h(0)

#mesure du resultat
circuit.measure(0, 0)
job = execute(circuit, backend)
plot_histogram(job.result().get_counts(), title = "Erreur de signe␣
 ↪corrigé")
```

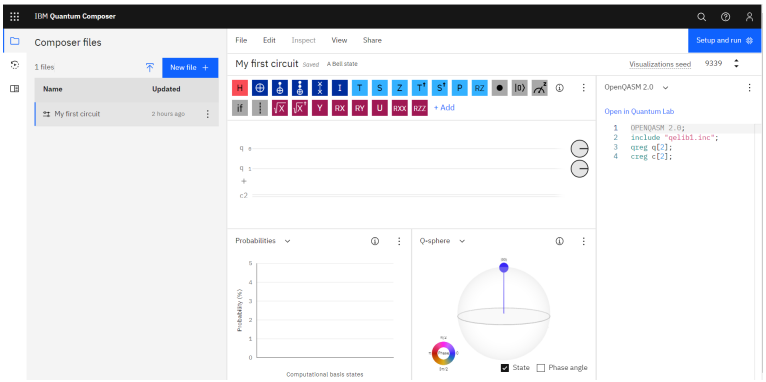

Erreur de signe corrigé
1.000
1.00
0.75
0.50
0.25
0.00
Probabilities
0

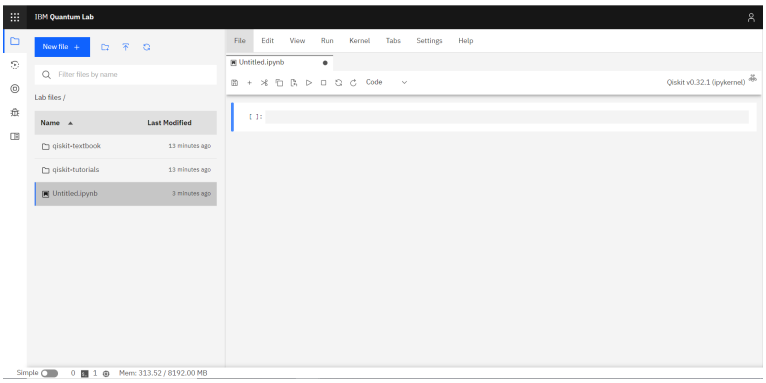
5 Tolérance aux fautes chez IBM

IBM pour International Business Machines Corporation est une entreprise multinationale américaine présente dans de nombreux domaines de l'informatique dont l'informatique quantique. En effet en 2014, la société a annoncé investir 3 milliards de dollars pour repousser les limites technologiques notamment grâce à la quantique. Au fil des quatre dernières années, IBM a conçu plus d'une trentaine d'ordinateurs quantiques. Ces ordinateurs utilisent des atomes froids piégés et refroidis par laser (à une température frisant les -273 °C, le zéro absolu) [6].



IBM se démarque notamment grâce à sa plateforme en ligne de simulation d'algorithmes quantiques IBM Quantum Service avec notamment : IBM Quantum Composer ou encore son "laboratoire" en ligne IBM Quantum Lab.

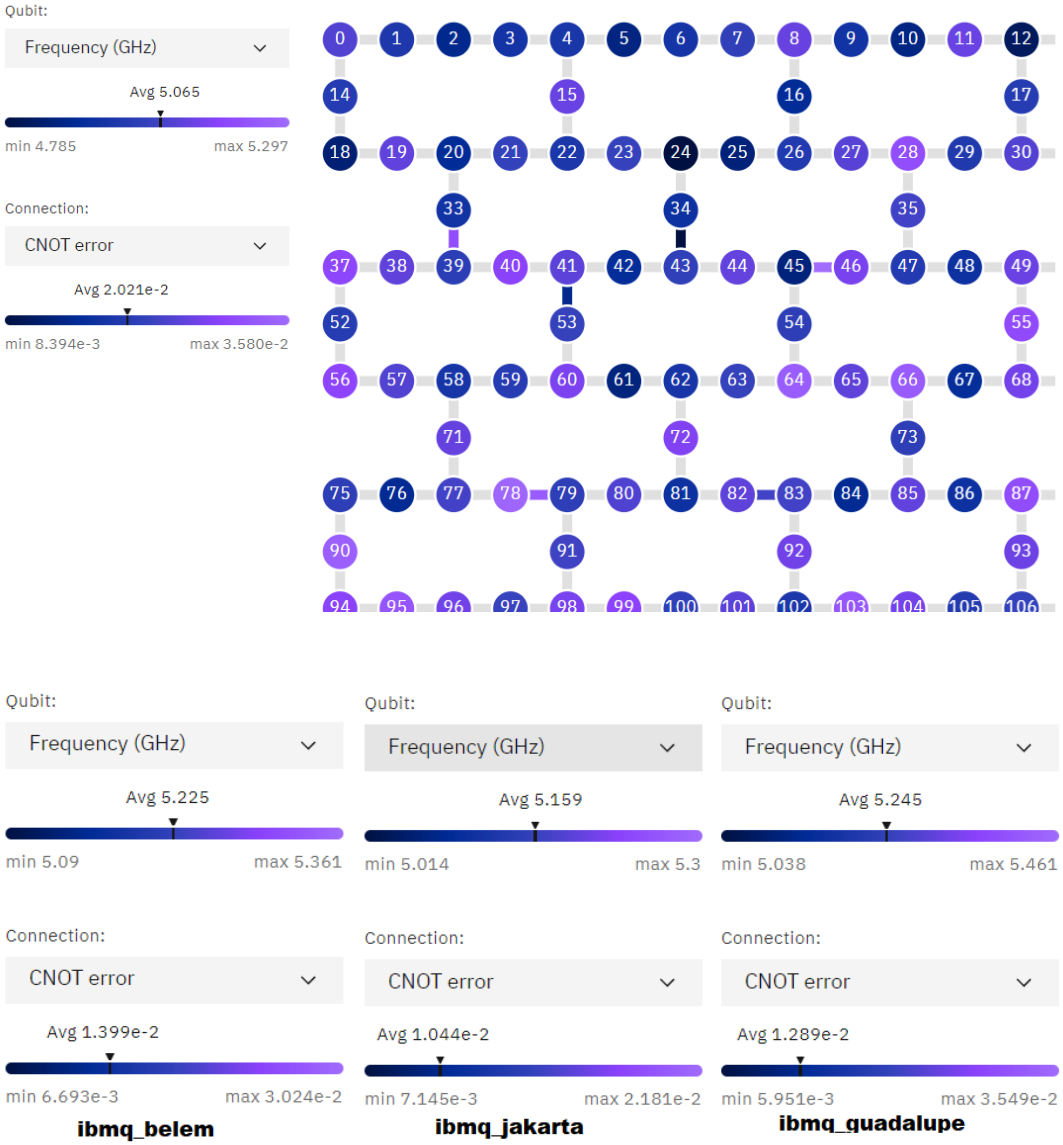




Cette plateforme permet à n'importe qui de réaliser n'importe quel circuit quantique très facilement soit graphiquement avec IBM Quantum Composer, qui généra d'ailleurs

un code python avec qiskit, ou en codant avec le laboratoire compilateur en ligne IBM Quantum Lab, sous python avec encore une fois la librairie Qiksit.

L'intérêt est de rendre accessible à tous l'informatique et ainsi faire passer IBM comme une référence en termes d'informatique quantique mais aussi de permettre à tous de pouvoir utiliser un ordinateur quantique car en effet chaque code produit, via leurs outils ou la librairie qiskit, peut être compilé sous un ordinateur quantique existant. Ainsi une des problématiques qu'il en résulte est de créer l'ordinateur quantique le plus fiable possible. En effet, comme dit précédemment, un ordinateur quantique essaye de s'approcher au maximum du zéro absolu pour fonctionner et cela représente un réel défi physique. À ce jour, IBM a atteint les 127 qubits mais ce n'est pas sans sacrifice car le processeur Eagle R1 (à Washington) possède un taux d'erreur sur les CNOTs plus important que d'autres processeurs.



Sur les différents processeurs pris au hasard [1], on tombe sur les moyennes suivantes :

1. Moyenne des minimums : 6.596e-3 $<$ 8.394e-3 soit 27% d'écart relatif,
2. Moyenne des maximums : 2.918e-2 $<$ 3.580e-2 soit 22% d'écart relatif,
3. Moyenne des moyennes : 1.244e-2 $<$ 2.021e-2 soit 62% d'écart relatif.

Ainsi, le processeur avec 127 qubits possède globalement plus de probabilités d'erreurs sur les portes CN0Ts qu'un processeur avec moins de qubits. Finalement on peut se poser la question suivante : comment fait IBM pour répondre au besoin d'avoir le moins d'erreurs possible ? Pour répondre entièrement à cette question il faudrait entrer dans des détails techniques. Or ceci ne sera pas possible pour plusieurs raisons : IBM ne donnera jamais exactement son fonctionnement en détails notamment pour des raisons de concurrence. Également parce qu'il nous sera impossible de répondre entièrement à la question au vu du niveau de détail possible pour chaque élément, il nous faudrait un rapport entier pour chaques points. C'est pourquoi nous allons nous intéresser aux points suivants :

— La topologie
— L'analyse comparative aléatoire des portes quantiques (Randomized Benchmarking of Quantum Gates)

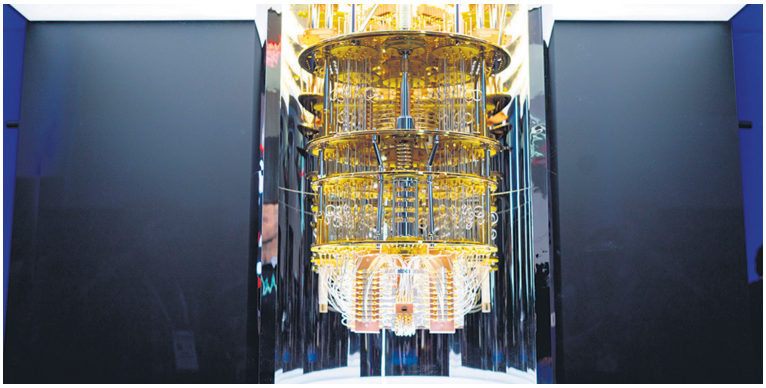
Mais d'abord nous allons expliquer comment IBM ou les autres entreprises font pour fabriquer un ordinateur quantique. [10]

5.1 Comment est fait un ordinateur Quantique ?

Les qubits de nos jours sont faits avec des éléments atomiques très sensibles à tout bruit. Pour cela, les entreprises comme IBM maximisent leurs chances. Les ordinateurs quantiques reposeront sur des supports peu sensibles aux vibrations de l'environnement. Ces ordinateurs seront eux-même dans une chambre vide pour réduire encore les champs magnétiques résiduels ou autres. Puis à l'intérieur, on y trouvera encore une chambre essayant d'approcher le 0 absolu car les qubits ont des contraintes thermiques de fonctionnement notamment pour leur stabilité qui pourrait être perturbée par la chaleur produite par les circuits thermiques. À l'intérieur de cette dernière chambre se trouvent les qubits. Maintenant pour ce qui est de faire réellement les qubits, il existe deux approches majeures :

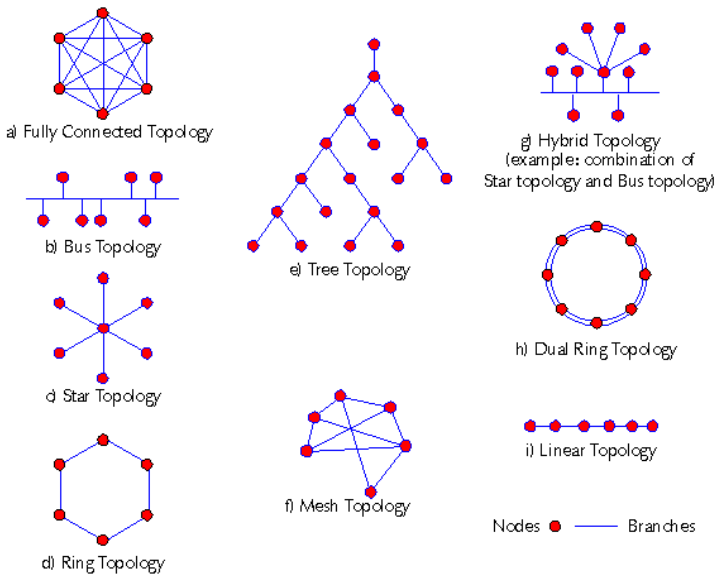
— les ions piégés
Un ordinateur quantique à ions piégés utilise des ions comme particules et les manipule avec des lasers. Les ions sont logés dans un piège fait de champs électriques. Les entrées laser indiquent aux ions quelle opération faire en causant la rotation de l'état du qubit sur la sphère.
— les qubits supraconducteurs Les ordinateurs quantiques à qubit supraconducteur le font différemment : ils utilisent une puce à circuits électriques au lieu d'un piège à ions. Les états de chaque circuit électrique se traduisent en état du qubit. Ils peuvent être manipulés par des entrées électriques sous forme de micro-ondes.

1. À noter qu'un échantillon de 3 éléments n'est pas représentatif d'une moyenne réelle puisque qu'IBM propose plus de 25 processeurs. L'idée est surtout de prendre quelques processeur avec 5-7 qubits et de comparer avec le nouveau processeur.



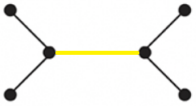
5.2 La topologie [9]

Maintenant que nous savons faire des qubits, il s'agit maintenant de réfléchir à l'interaction qu'ils auront entre eux. Et on retrouve classiquement le problème de topologie que l'on a pour les réseaux où on serait tenter d'utiliser une topologie complètement connectée. Cependant comme pour les réseaux classiques multiplier le nombre de connexions ne résout pas le problème et en créé un. En effet, plus on multiplie le nombre de connexions plus on augmente le risque d'obtenir une erreur et moins les qubits seront stables. Alors on pourrait aussi se poser la question de la topologie en ligne mais dans cette condition on perd l'efficacité que l'on souhaiterait avoir lorsque l'on utilise des portes à plusieurs qubits ou des portes réutilisant le résultat d'un qubit pour un calcul avec un tierce qubit. De ce fait, il existe de nombreuses topologies possibles et les différentes entreprises voulant créer des qubits se sont retrouvées face à ce dilemme : quelle topologie choisir ? Ces entreprises auraient pu pendre parmi ce qui existait déjà et ce qui était déjà fait pour les réseaux mais cela n'aurait pas été aussi efficient (photo ci-jointe)



Cependant, une première solution a été choisie à savoir une topologie en étoile ce qui a pour effet de n'avoir qu'un seul qubit avec un taux d'erreur important [5]

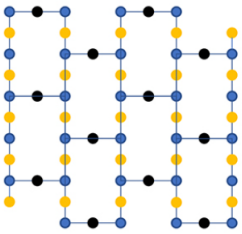
Cependant ce graphe n'étant pas optimal car de degré 4 et ce degré étant encore trop élevé pour son taux d'erreur, il a fallu réduire encore le degré pour ça une idée est venue découper ce qubit "en deux" afin de n'avoir que deux qubits de degré 3 reliés entre eux pour obtenir la topologie suivante :



Toutefois un problème réside encore : la liaison entre les 2 qubits de degré 3 qui reste instable (en jaune). Pour résoudre ce problème, IBM a eu l'idée d'intercaler un qubit entre ces deux qubits obtenant ainsi la configuration suivante :

Ainsi, nous obtenons la configuration actuelle d'une interconnexion de 3 qubits chez IBM, à savoir que cette configuration peut être amenée à évoluer. Nous avons donc vu que la topologie a suivi cette évolution :

Donc si maintenant, on cherche à connecter entre eux les différents morceaux d'interconnexion on obtiendra le maillage en hexagone lourd suivant



On voit très bien que cette configuration n'excède pas le degré 3 ce qui lui permet d'être une des topologies les plus stables à ce jour et celle d'ailleurs utilisé chez IBM comme vu avec le processeur Eagle R1. Cependant, la topologie est une des causes des erreurs et optimiser la topologie permet de réduire drastiquement le taux d'erreur. En effet, une interconnexion aléatoire mènerait probablement à un ordinateur moins stable. Il est à noter que ce qui est dit dans ce rapport se rapporte à ce jour (Janvier 2022). Le secteur

de l'informatique quantique est en grande expansion et il plus que probable qu'IBM ou un de ses concurrents sorte une topologie révolutionnaire ou un moyen d'interconnexion qui permettrait d'utiliser des topologies qui nous seraient absurdes aujourd'hui. Également, il faut savoir que certains ordinateurs quantiques sont optimisés pour certains algorithmes, on pense notamment aux algorithmes de Grover, de Shor. Ainsi, ce choix de topologie est aussi partie pris pour améliorer la performance de ces algorithmes et montrer la supériorité quantique par rapport aux ordinateurs classiques. Cependant, nous l'avons évoqué mais la topologie n'est pas le seul facteur à prendre en compte puisque le choix des matériaux, de la technologie, la durée de vie des qubits etc. Et à ce fait nous allons maintenant voir une technique qui s'appelle "Randomized Benchmarking of Quantum Gates"

5.3 L'analyse comparative aléatoire des portes quantiques

Il existe plusieurs méthodes pour détecter le comportement des portes quantiques et savoir si elles produisent des erreurs et l'une d'entre elles est l'analyse comparative aléatoire des portes quantiques. [8] [4] Pour cela, classiquement, on effectuerait une tomographie de processus mais celle-ci souffre de plusieurs problèmes notamment de détection d'erreurs quand les portes sont disposées en séquence longue. De plus, l'ordre des erreurs et assez faible puis on sera de l'ordre de 10^{-4} ou moins, ce qui est difficile à démontrer. L'idée est donc de choisir des qubits aléatoirement et mettre en œuvre une analyse comparative aléatoire sur les ions atomiques piégés, établissant une probabilité d'erreur d'un qubit par impulsion $\pi/2$ randomisée de 0,00482. Pour des raisons de clarté et pour ne pas finir avec un rapport avec de trop nombreuse page, nous n'allons pas détailler le processus mais si l'envie vous vient de vous y intéresser il existe un article qui parle très bien du sujet[8]. Nous allons donc surtout nous intéresser à l'impact que cela a sur la performance des ordinateurs quantiques. En effet, tout cela serait vain si cela n'apporte aucune amélioration. Mais si on définit la fiabilité par $F(\varepsilon) = \int d\psi < \psi|\varepsilon(|\psi><\psi|)|\psi>$. On peut observer une réelle diminution des erreurs notamment des erreurs SPAM [3] Notamment avec le code [4,2,2][12] on peut observer une diminution des erreurs allant de 5,8% à 0,60%[4]

5.4 Et donc ?

Ainsi nous avons vu qu'il existe plusieurs approches possibles pour diminuer le taux d'erreurs, d'abord un choix de matériel pour être le moins sensible aux vibrations, un choix de technique pour simuler les qubits suivants si on veut des qubits +/- stables et plus ou moins précis aux changements de rotation, un choix de topologie et enfin choix de protocole de détection d'erreurs. C'est en combinant tous ces éléments qu'IBM pourra obtenir l'ordinateur quantique le plus fiable possible.

6 conclusion

Les codes correcteurs classiques emploient la redondance. La méthode la plus simple est de stocker un nombre suffisamment de fois pour que nous puissions établir un vote à la majorité. Cette méthode est facilement adaptable pour un ordinateur quantique permettant de corriger des erreurs de bit-flip. Nous pouvons l'adapter sur une autre base grâce à l'opérateur d'Hadamard pour corriger les erreurs de phases. Une combinaison de fonctionnement des deux précédentes méthodes résulte de l'algorithme de Shor. Comme il n'est pas possible de stocker l'état d'un registre, il faut absolument travailler avec des registres quantiques temporaires pour "stocker" un maximum d'informations. Les algorithmes présentés sont simples à mettre en place et à comprendre. Cependant si d'autres algorithmes de correction viennent à être plus précis, corrigeant plus d'erreurs, il ne serait pas étonnant qu'ils manipuleraient un nombre plus important de registre, comme nous pouvons voir la transition entre le bit-flip / phase-flip code au code de Shor.

Références

[1] Les codes correcteurs quantiques. *Collège de France*, 2004.

[2] Schaetz Chiaverini, Leibfried. Realization of quantum error correction. *Nature 432*, 2004.

[3] Michael R. Geller and Mingyu Sun. Efficient correction of multiqubit measurement errors.

[4] Robin Harper and Steven T. Flammia for Sydney University. Fault-tolerant logical gates in the ibm quantum experience.

[5] IBM. Hardware-aware approach for fault-tolerant quantum computation.

[6] Leïla Marchand. Ordinateur quantique : cinq questions pour (enfin) tout comprendre. *LesEchos*, 2021.

[7] Brice Mayag. Contrôle d'erreurs, 2011.

[8] National Institute of Standards and Technology. Randomized benchmarking of quantum gates.

[9] Bernard Ourghanlian pour Microsoft. Randomized benchmarking of quantum gates.

[10] Chiara DECAROLI pour TED. the high stakes race to make quantum computers work.

[11] Oscar González / H Shrikumar / John A. Stankovic / Krithi Ramamritham. Adaptive fault tolerance and graceful degradation under dynamic hard real-timescheduling. *University of Massachusetts Amherst*, 1997.

[12] science advances. Fault-tolerant quantum error detection.

[13] Quantum Computing UK. Quantum error correction : Bit flip code in qiskit. *Quantum Computing UK*, 2020.

[14] Quantum Computing UK. Quantum error correction : Phase flip code in qiskit. *Quantum Computing UK*, 2020.

[15] Fumiko Yamaguchi. Quantum error correction. *Stanford University*, 2005.

[16] Boyu Zhou. Quantum error correction codes, 2020.