La tomographie quantique

Par Thomas Alexendre, Ludovic Chabot Provencher et Maxime Fortier

20 février 2024

1 Introduction

Quand on parle de l'information sur un système quantique, il arrive parfois que l'on veuille connaitre son état aussi précisément que l'on veux. Le principe derrière le calcule de cet état quantique est la tomographie quantique. En effet, la tomographie quantique permet de faire connaître un état quantique aussi précisément qu'on le veuille à la condition que l'état puisse être reproduit de manière consistante un nombre illimité de fois.

1.1 Le déboggage quantique

Une utilité fort intéressante de la tomographie quantique est celle de pouvoir obtenir l'état d'un système à un moment donné. En effet, lorsqu'on tente de trouver où est l'erreur dans un code, on aime pouvoir mettre en pause le système afin d'examiner, étape par étape, les variables impliquées dans les opérations. En informatique classique, cette fonctionnalité est présente dans la plupart des IDE, sous la forme des breakpoints. De plus, pour une analyse en profondeur des opérations exécutés, il existe des outils comme x64dbg, qui permettent d'analyser un programme au niveau du code machine.

Or, dans la programmation quantique, le concept des breakpoints tels que implémenter dans l'informatique classique n'existe pas. Une façon d'implémenter une fonctionnalité similaire serait d'utiliser la tomographie quantique pour évaluer l'état d'un système à un moment dans l'exécution du circuit quantique. Néanmoins, la méthode présentée dans ce document n'est pas appropriée pour une telle fonctionnalité, du à son coût prohibitif en termes de temps.

2 La théorie

Comme il a été mentionné précédemment, nous avons un état quelconque $|\psi\rangle$ dont la dimension de son Hilbert est n et dont nous cherchons à calculer.

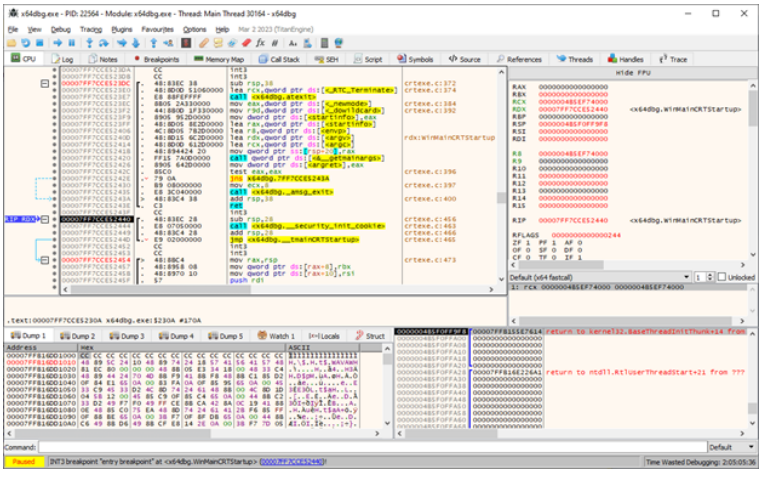


FIGURE 1 – interface de x64dbg

2.1 La matrice densité

Afin de pouvoir calculer notre état quantique inconnu, il faut utiliser une représentation alternative du vecteur d'état qui est la matrice densité $\hat{\rho}$. Elle se calcule par le produit diadique du vecteur d'état avec lui-même.

$$\hat{\rho} = |\psi\rangle\langle\psi| \tag{1}$$

On peut voir qu'il est facile de retrouver le vecteur d'état. Il suffit que calculer le vecteur propre de la matrice densité ayant comme valeur propre de 1.

$$\begin{aligned} \hat{\rho}\,|\psi\rangle &= |\psi\rangle\,\langle\psi|\psi\rangle \\ \hat{\rho}\,|\psi\rangle &= |\psi\rangle \end{aligned} \tag{2}$$

2.2 Les chaines de Pauli

Une chaine de Pauli un observable formé d'un produit tensoriel d'une combinaison de matrice de Pauli ainsi que de l'identité.

$$\hat{P}_i = \bigotimes_{q=0}^{n-1} \hat{\sigma}_q \tag{3}$$

avec

$$\hat{\sigma}_q \in \{\mathbb{1}, \sigma_x, \sigma_y, \sigma_z\}$$

L'ensemble de toutes les chaines de Pauli d'un espace de Hilbert forme une base pour cette même espace. Il est donc possible d'écrire tous les observables comme une somme de chaine de Pauli. On peut donc aussi écrire la matrice densité ainsi.

$$\hat{\rho} = \sum_i a_i \hat{P}_i \tag{4}$$

2.2.1 la notation ZX

La notation ZX est une façon plus concise d'écrire les chaines de Pauli.
On par simplement du fait qu'une matrice de Pauli peut être écrite comme

$$\hat{\sigma} = (-1)^{xz} \hat{X}^x \hat{Z}^z = \begin{cases} \mathbb{1} & \text{si } x = z = 0 \\ \sigma_x & \text{si } x = 1 \text{ et } z = 0 \\ \sigma_y & \text{si } x = z = 1 \\ \sigma_z & \text{si } x = 0 \text{ et } z = 1 \end{cases}$$

Pour pouvoir écrire les chaines de Pauli ainsi, il suffit de prendre x et z comme des vecteurs d'éléments binaires et le produit de xz est le produit scalaire de ces deux vecteurs.

$$x \cdot z = \sum_i x_i z_i$$

2.3 Les valeurs moyennes

Nous savons que la valeur moyenne d'un observable est égale à la trace du produit de la matrice densité et de l'observable.

$$\langle\psi|\hat{A}|\psi\rangle = \mathrm{Tr}\Big\{\hat{\rho}\hat{A}\Big\}$$

Lorsque l'on calcule la valeur moyenne des chaines de Pauli, on remarque qu'elles sont égale à leur coefficient a_i de l'équation (4) à une constante près.

$$\langle\psi|\hat{P}_i|\psi\rangle = \mathrm{Tr}\Big\{\hat{\rho}\hat{P}_i\Big\} = \mathrm{Tr}\left\{\sum_{j=0}^{n-1} a_j \hat{P}_j \hat{P}_i\right\} = \sum_{j=0}^{n-1} a_j \,\mathrm{Tr}\Big\{\hat{P}_j \hat{P}_i\Big\}$$

Puisque

$$\mathrm{Tr}\Big\{\hat{P}_j\hat{P}_i\Big\} = \begin{cases} 2^n & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

On a

$$\langle\psi|\hat{P}_i|\psi\rangle = 2^n a_i \tag{5}$$

Ensuite, pour calculer la valeurs moyenne de ces chaines de Pauli, on doit avant tous les diagonaliser, car l'ordinateur quantique ne permet seulement de faire des mesures des les observable suivant Z.

$$\langle\psi|\hat{X}|\psi\rangle = \langle\psi|\hat{H}\hat{Z}\hat{H}|\psi\rangle = \Big\langle\psi^{(x)}\Big|\hat{Z}\Big|\psi^{(x)}\Big\rangle \tag{6}$$

$$\langle\psi|\hat{Y}|\psi\rangle = \langle\psi|\hat{S}\hat{H}\hat{Z}\hat{H}\hat{S}^\dagger|\psi\rangle = \Big\langle\psi^{(y)}\Big|\hat{Z}\Big|\psi^{(y)}\Big\rangle \tag{7}$$

Pour mesurer la valeur moyenne des autres axes, nous devons diagonaliser les observables $\hat{X}$ et $\hat{Y}$ par les équation (6) et (7) respectivement. En faisant cela, nous appliquons les portes $\hat{S}^\dagger$ et/ou $\hat{H}$ aux circuits qui feront les mesures des valeurs moyennes changeant ainsi l'état pour qu'il soit mesuré dans l'axe que nous désirons.
Pour mesurer la valeur moyenne d'une chaine de Pauli diagonalisée en utilisant la notation ZX, nous n'avons besoin que du vecteur des Z, puisqu'il n'y a aucune porte $\hat{X}$ et $\hat{Y}$. La valeur moyenne est donc la somme du produit des probabilités de chaque état de base avec leur valeur propre associé.

$$\langle\psi|\hat{Z}^z|\psi\rangle = \sum_i \lambda_i P_\psi(|b_i\rangle)$$

On sait que la probabilité est approximativement égale au nombre de fois que l'état est mesuré divisé par le nombre de mesures totales.

$$P_\psi(|b_i\rangle) \approx \frac{n_i}{N}$$

La valeur propre est de -1 ou 1 selon le nombre de porte $\hat{Z}$ qui ont été appliqué sur un état de 1. Il suffit alors de calculer combien de fois le vecteur Z et les chiffres de l'état de base ont un 1 à la même position.

$$\lambda_i = (-1)^{C_i}$$

où

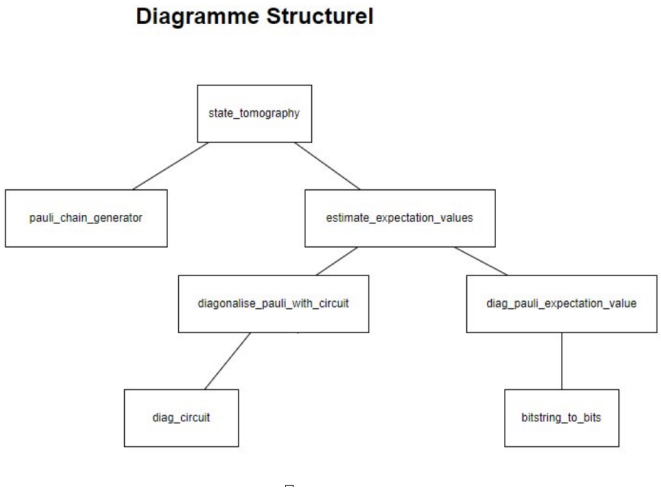
$$C_i = \sum_j z_j (b_i)_j$$

Nous pouvons donc calculer les valeurs moyennes de chaque chaine de Pauli en divisant par un facteur de 2^n selon (5). On peut ensuite construire la matrice densité selon (4). Finalement, pour retrouver l'état, il suffit de diagonaliser la matrice densité par l'équation (2).

3 L'implémentation

Notre implémentation de l'algorithme se divise en deux fichier. Le premier fichier s'appel tomologie.py et on y trouve chacune des fonctions nécessaires pour trouver l'état mystère. Le deuxième fichier se nommant tests_for_state_tomography.py nous permet de rouler ainsi que de commenter les des tests pour notre programme.

Tout d'abord, pour représenter la façon que nous avons implémenter notre code, voici le diagramme structurel qui suit l'appel des fonctions dans notre fichier tomography.py.



Lorsque nous voulons déterminer l'état de notre système nous devons appeler la fonction state tomography qui prend en entrée le circuit de notre système et le nombre de répétitions qu'on veut exécuter lors de la mesure de ce circuit. state_tomography appel deux fonctions pauli_chain_generator qui nous retourne tous les chaines de pauli possibles pour un certain nombre de qubits et estimate_expectation_value qui estime la valeur de chaque circuit de transformation ajouté au circuit de départ lié aux chaines de pauli possible. Pour estimer une valeur pour chaque chaine de pauli, nous devons d'abord appeler diagonalize_pauli_with_circuit qui nous retourne nos chaines de pauli diagonalisé en plus du circuit qui nous aide à faire cette transformation. Cette fonction appel diag_circuit qui nous retourne la circuit de transformation pour une chaine de pauli quelconque.

Avec le résultat de diagonalize_pauli_with_circuit nous avons appelé diag_pauli_expectation_value qui calcule la valeur moyenne pour chacune des chaines de pauli diagonalisé. Cette fonction appel la sous fonction bitstring_to_bits pour changer le string de bits qui vient de counts en un numpy array avec laquel on peut faire un dot product.

4 Les tests

Pour tester notre circuit nous avons utiliser un test simple et un test plus complexe. Pour vérifier nos résultats nous avons utilisés circuit composer de IBM.

Pour le premier test, nous avons crée un circuit avec 4 qubits qui a des portes hadamard, z, s, s dag et t dag. La valeur attendue pour notre circuit était d'une amplitude de 0.25 pour tous les états possibles à 4 qubits et c'est ce résultat que nous avons eu lors de l'éxecution de notre programme.

Pour le deuxième test, nous avons crée un circuit avec 1 qubit sur laquel nous avons appliqué une porte hadamard. La valeur attendue était une amplitude de 0,707 pour les deux états possibles et nous avons aussi constaté ce résultat lors de l'exécution de notre code.

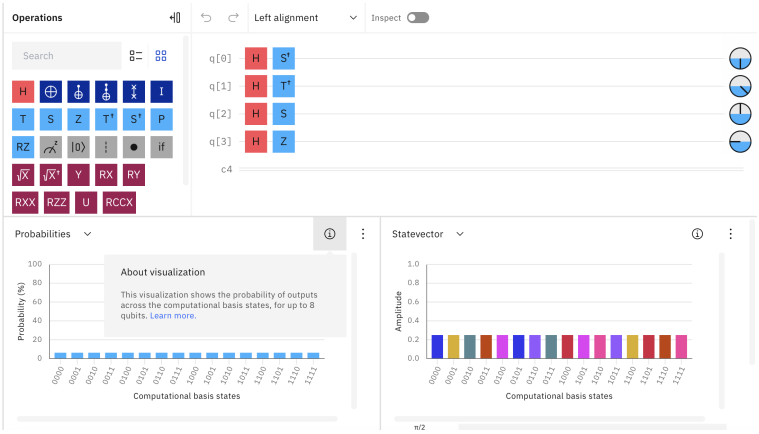


FIGURE 2 – Test 1

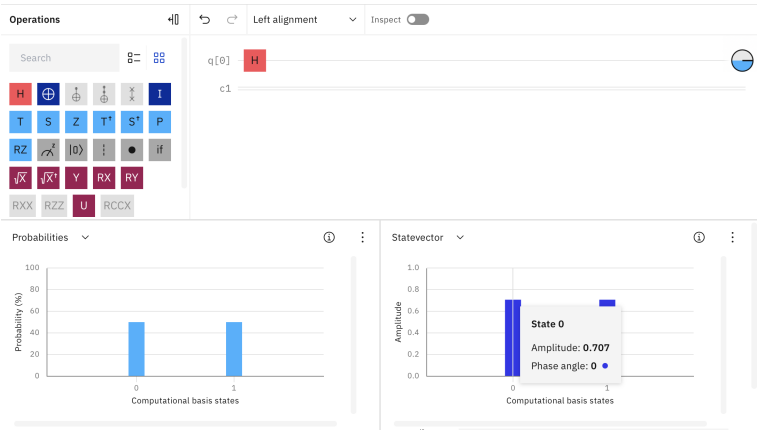


FIGURE 3 – test 2

```
#Test 1:
state_vector = [-2.49543734e-01-0.00000000e+00j -8.96642565e-04+2.49655645e-01j
                -1.76608838e-01+1.76804024e-01j  1.77283939e-01+1.75894252e-01j
                1.01665660e-04-2.49943123e-01j -2.50247031e-01-8.02927714e-05j
                -1.76670932e-01-1.76827321e-01j -1.77231718e-01+1.78172277e-01j
                2.49898223e-01-9.45811887e-04j -2.21232141e-04-2.50013259e-01j
                1.76712242e-01-1.77389663e-01j -1.77275329e-01-1.77168028e-01j
                7.34181653e-04+2.50342464e-01j  2.48997491e-01-4.03811364e-04j
                1.76008946e-01+1.77042632e-01j  1.76301559e-01-1.76932167e-01j]

#Test 2:
state_vector = [-0.70743904+0.j         -0.70677097+0.00219099j]
```

FIGURE 4 – Resultats