École nationale supérieure d'informatique pour l'industrie et l'entreprise

«TP2 :Quantique»

Élève : **Aurélien LHUILLIER**

Sommaire

1 TP2 :Quantique **1**

1.1 Introduction 1

1.2 Travail 1

1.2.1 La décomposition ABC 1

1.2.2 Programme du circuit ABC 1

1.3 Porte controlé CNOT 2

1.4 Porte controlée sur 2qubit 3

1.5 Implémentation de la porte à 3 qubit 4

1.6 Porte généralisé à n qubit 5

1.7 Bonus 6

1 TP2 :Quantique

1.1 Introduction

L'objectif de ce TP est de réaliser la construction d'une porte contrôlée quelconque sur un qubit.Avec n le nombre de qubit choisit.Pour réaliser la construction de ces portes contrôlé,la formule de décomposition de Sleathor-Weinfurter sera mise en oeuvre pour construire récursivement celles-ci.On utilisera comme porte controlée U la porte X.

1.2 Travail

1.2.1 La décomposition ABC

On choisit la porte X ,l'objectif est de mettre X$=e^{i\alpha}R_z(\beta)R_y(\gamma)R_z(\delta)$ Ainsi on prend la matrice du cours et on résoud le système associé On trouve ainsi $\alpha = -\pi/2$ $\beta = 0$ $\gamma = \pi$ $\delta = \pi$ On peut ainsi écrire X$=e^{i0}R_z(0)R_y(0)R_z(1)$ Or d'après le cours on a :

$$A = RZ(\beta)RY(\frac{\gamma}{2})B = RY\left(-\frac{\gamma}{2}\right)RZ\left(-\frac{\beta+\delta}{2}\right)C = RZ\left(\frac{\delta-\beta}{2}\right)$$

1.2.2 Programme du circuit ABC

L'implémentation du programme sera à l'aide de my qlm.Voici le schéma de fin du programme avec la fonction python.

```
compteur = 0
prog = Program()
#definition des constantes
alpha=-np.pi/2
beta=0
gamma=np.pi
delta=np.pi
#debut du circuit
qbits = prog.qalloc(1)
#porte C
prog.apply(RZ(delta-beta/2), qbits[0])
compteur = compteur + 1
prog.apply(X,qbits[0])
compteur = compteur + 1
#porte B
prog.apply(RZ((-beta+delta)/2), qbits[0])
compteur = compteur + 1
prog.apply(RY(-gamma/2), qbits[0])
compteur = compteur + 1
prog.apply(X,qbits[0])
compteur = compteur + 1
#porte A
prog.apply(RY(gamma/2), qbits[0])
compteur = compteur + 1
prog.apply(RZ((beta)), qbits[0])
compteur = compteur + 1
#phase
prog.apply(GlobalPhase(0), qbits)
compteur = compteur + 1
```

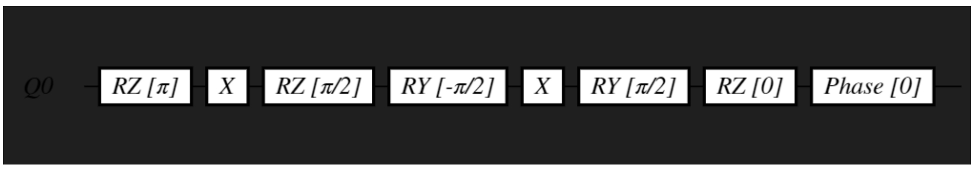

FIGURE 1 – Décomposition ABC de X.

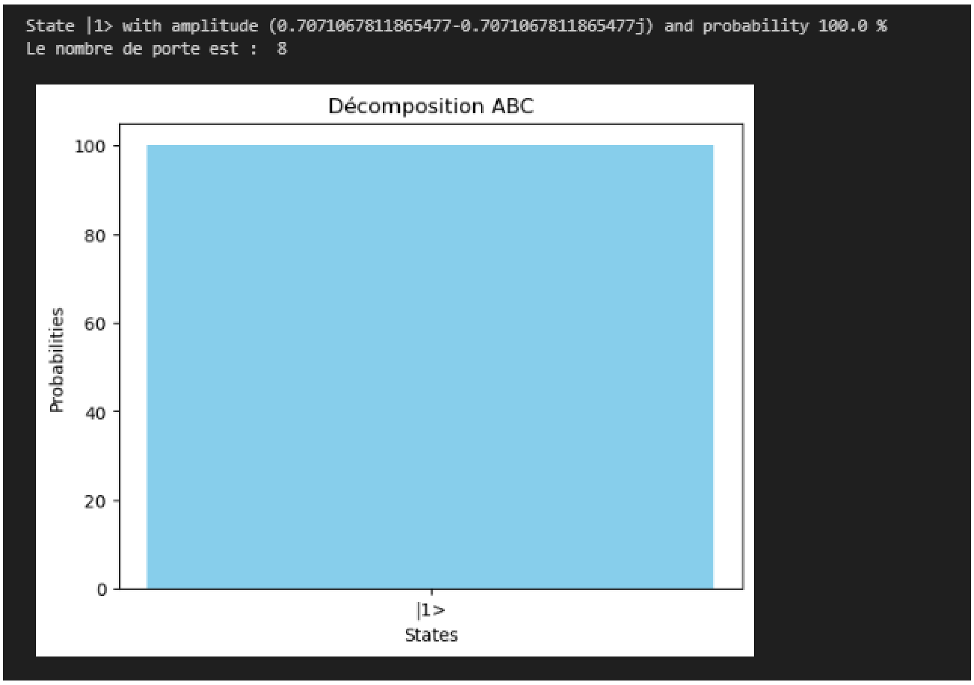

Figure 2 – Résultat sur le qubit 0.

1.3 Porte controlé CNOT

Pour réaliser la porte contrôlée CNOT,il suffit de remplacer les portes X par des Cnots.

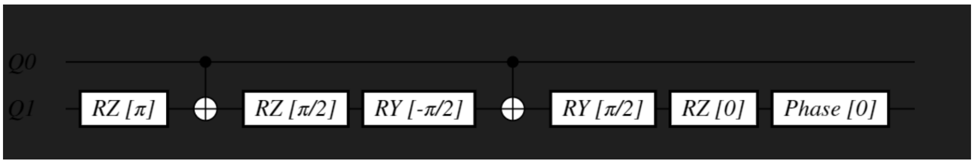

Figure 3 – Porte CNOT.

On obtient le résultat suivant :

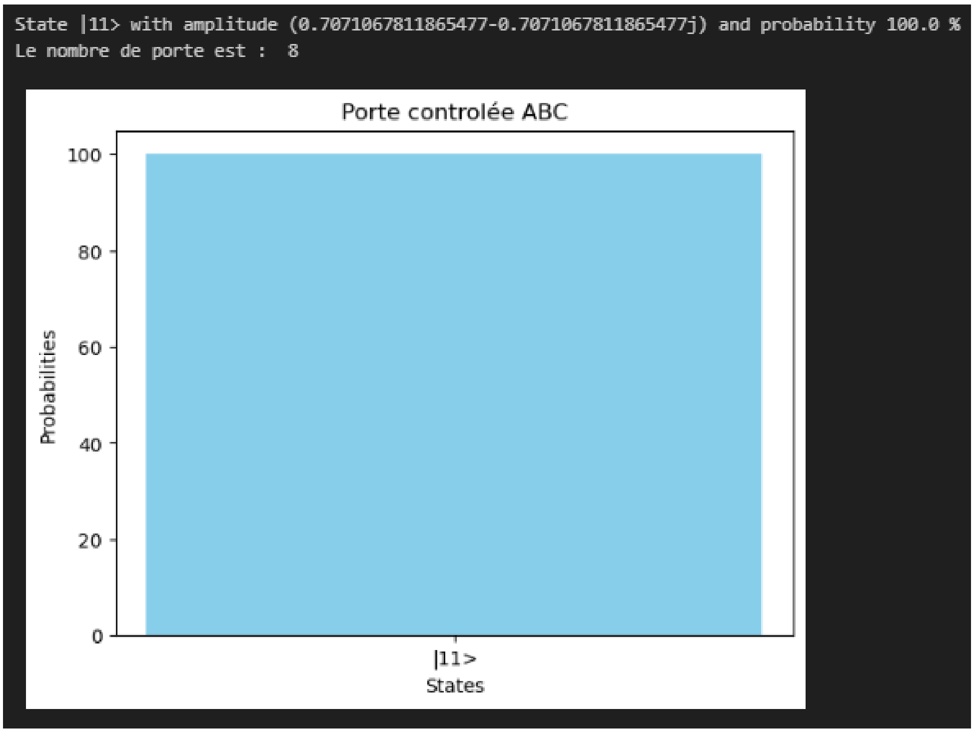

Figure 4 – Résultat du qubit 01.

1.4 Porte controlée sur 2qubit

Grâce à la décomposition de Sleathor-Weinfurter,on va construire la porte contrôlée à 2 qubits.Nous allons dans un premier temps chercher une porte tq $V^2 = CNOT$ puis implémenter la porte en suivant le schéma.On a .Dans le cadre de ce TP j'ai décidé de garder les rotations que je trouvais pertinente pour la porte CNOT.

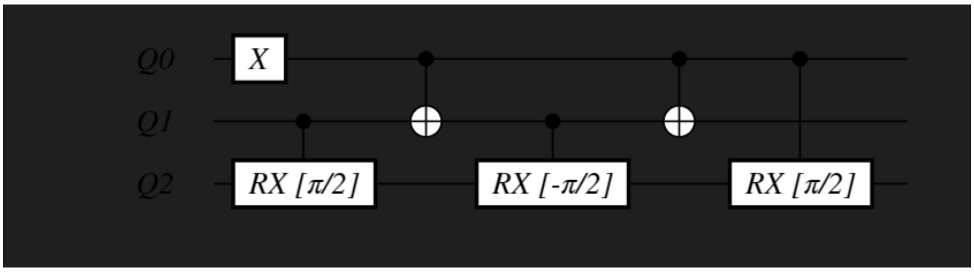

Figure 5 – Circuit CNOT à 2 qubit.

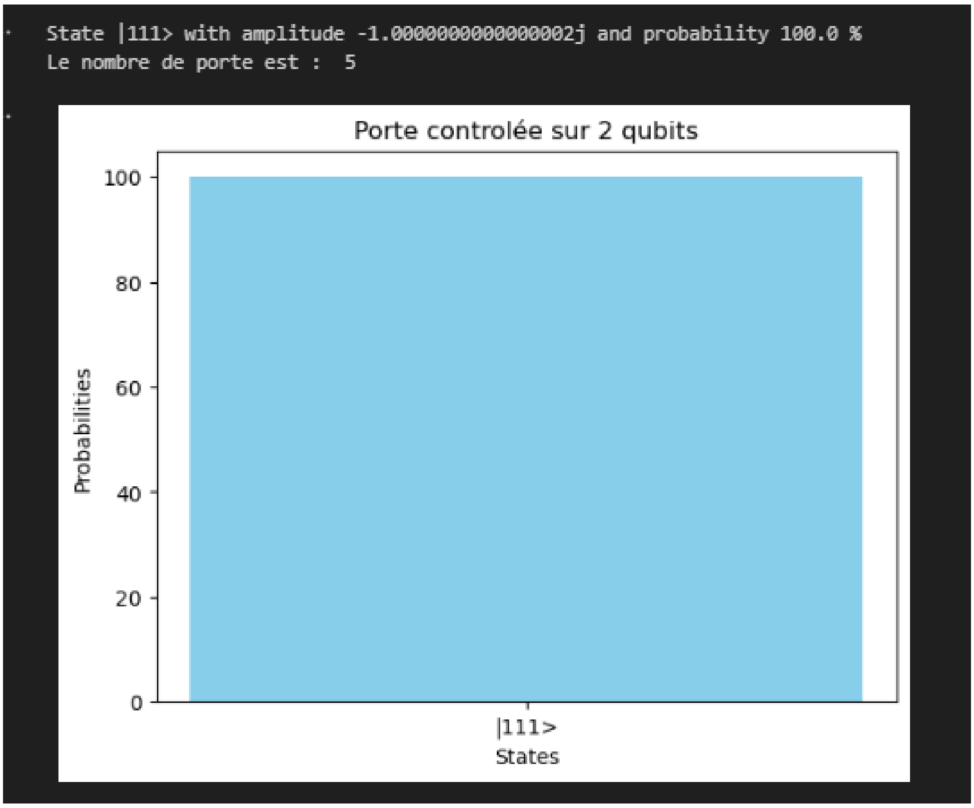

FIGURE 6 – Résultat du qubit 110.

1.5 Implémentation de la porte à 3 qubit

On utilisera cette fois-ci le principe de réduction de Sleator-Weinfurter.Cette implémentation se fera avec les rotations RX ce qui est plus intuitifs en visualisant la sphère de bloch.Donc on veut $V^4 = CNOT$ donc c'est une rotation de $\frac{\pi}{4}$

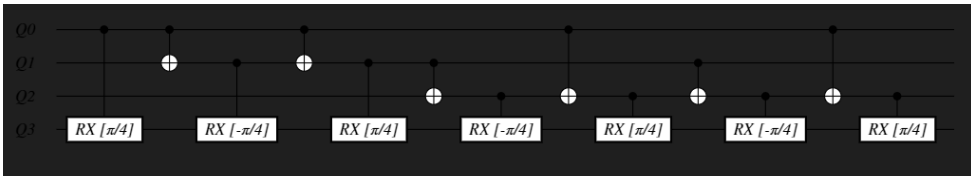

FIGURE 7 – Circuit CNOT à 3 qubit.

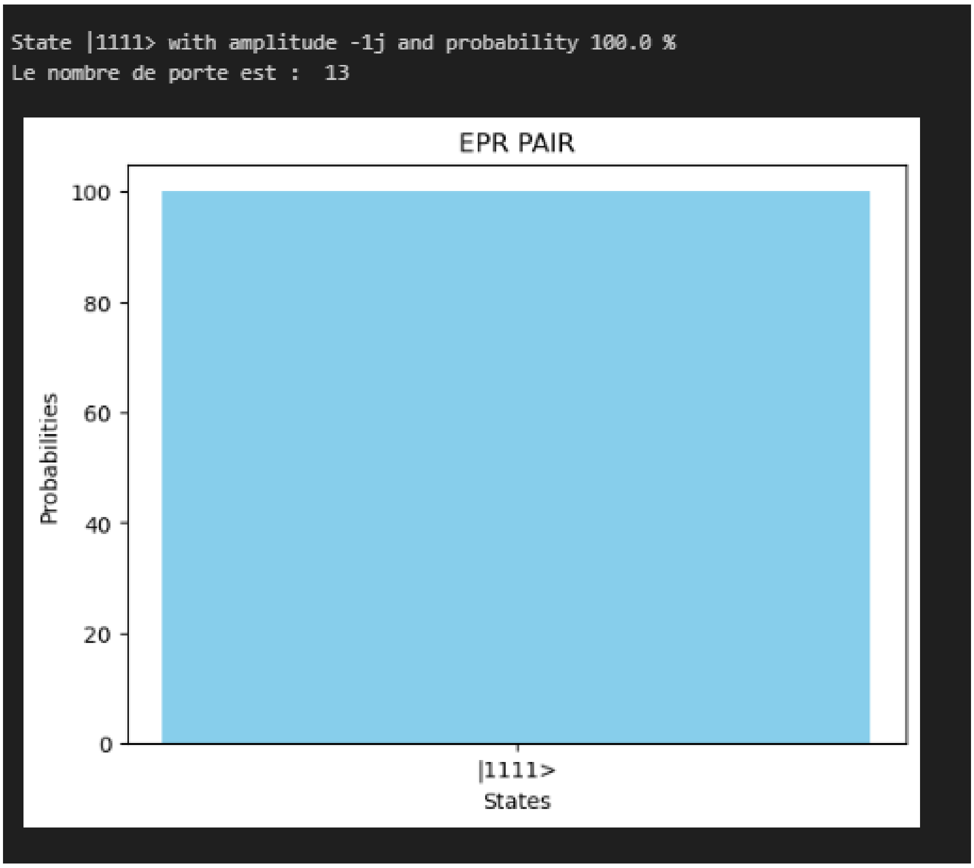

FIGURE 8 – Circuit CNOT à 2 qubit.

1.6 Porte généralisé à n qubit

Pour réaliser cette porte,il suffit par récurrence connaissant les portes précédentes de suivre la décomposition généralisé de de Sleathor-Weinfurter.On veut également compter le nombre de portes dont on aura besoin pour la construction de celle-ci.Pour ce faire il y a 2 solutions :

- Réaliser l'implémentation de la porte par une fonction récursive partant de la 3ème porte.Cette solution ne sera pas présenté car elle rend l'implémentation en matière de code assez compliqué et n'utilise pas les possibilités qu'offre myqlm.
- Déduire analytiquement une fonction qui calcul le nombre de porte en fonction de n>3 par récurrence.Cette solution permet un implémentation simple et clair de la porte CNOT contrôlé sur n tout en se généralisant sur une porte U quelconque en changeant la porte V.

On considère les portes CNOT comme unitaire même si on pourrait les décomposer grâce à la décomposition ABC.Nous allons déterminer la formule en toute généralitée donc CNOT sera remplacé par V.
On a donc une porte contrôlée sur 3 qubit qui comporte 13 portes.
Construisons la porte suivante sur 4 qubits. Le circuit contient :

- une porte contolée V qui vaut 1
- une porte CNOT sur 3 qubit qui vaut 13 portes
- une porte controlé $V^\dagger$ qui vaut également 1
- une porte CNOT sur 3 qubit qui vaut 13 portes
- une porte controlée V sur 3 qubit qui vaut également 13 portes

On en déduit la formule suivante $nbr-de-porte-4 = 1+13+1+13+13 = 2+13*3$.
On réalise la même chose pour 4 qubit en utilisant le résultat trouvé précédemment.
On en déduit $nbr-de-porte-5 = 3*nbr-de-porte-4+2 = 3^2*13+3*2+2$.
On fait de même pour 6 $nbr-de-porte-6 = 3^3*13+3^2*2+3*2+2$.
Ainsi on voit le pattern se dessiner et on a

$$nbr-de-porte-n = 3^{n-3}*13+\sum_{i=0}^{n-4} 3^i*2$$

On réalise donc l'implémentation en python de cette fonction.
Fonction de calcul du nbr de porte n>4.

```python
def calculate_sum(n):
    return 3**(n-3)*13 + sum(3**i * 2 for i in range(n-4))
```

Pour la suite du code voir code

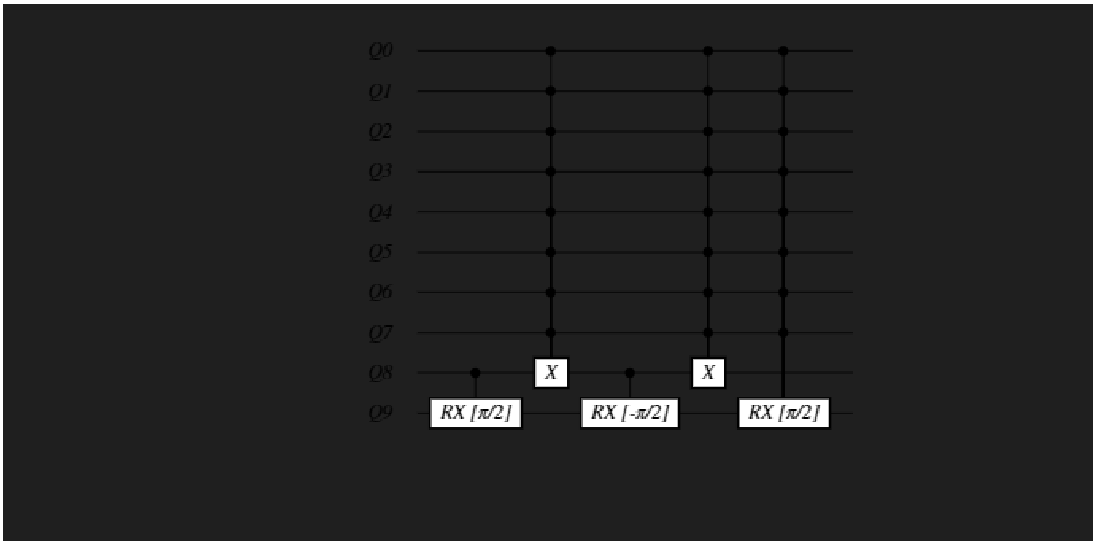

FIGURE 9 – Circuit CNOT à 10 qubit.

1.7 Bonus

J'ai réalisé dans le cadre de ce TP une CNot sur n composé seulement de Cnot simple en réimplémentant le code de quiskit

```python
def build_mcx_circuit_v_method(n):
    prog = Program()
    #d but du circuit
    qbits = prog.qalloc(n)
    #essaie de qbit 111

    if n == 1:
        prog.apply(X,qbits[0])
    elif n == 2:
        prog.apply(CNOT,qbits[0],qbits[1])
    else:
        # Apply CNOT gates to form the V-pattern
        for i in range(n-1):
            prog.apply(CNOT,qbits[i],qbits[n-1])
            for j in range(i+1, n-1):
                prog.apply(CNOT,qbits[i],qbits[j])
                prog.apply(CNOT,qbits[j],qbits[n-1])
                prog.apply(CNOT,qbits[i],qbits[j])  # Uncompute the previous CNOT
    circuit = prog.to_circ()
    circuit.display()
    job = circuit.to_job()
    linalgqpu = PyLinalg()
    result = linalgqpu.submit(job)
    l = len(result)
    states = ['']*l
    probabilities= [0]*l
    i=0
    for sample in result:
        print("State",sample.state,"with amplitude",
            sample.amplitude,"and probability",
            round(sample.probability*100,2),"%")
        states[i] = str(sample.state)
        probabilities[i] = round(sample.probability*100)
        i = i+1

    plt.bar(states, probabilities, color='skyblue')
```

```python
    plt.xlabel('States')
    plt.ylabel('Probabilities')
    plt.title('mcnot')
    plt.show()
    return prog
```

Je voudrais également souligné la manière de construire la racine nème de X avec le changement de base grâce à H dans ce papier