C.2.2 Amusons nosu avec Scilab

le script `epr.sci` est le suivant, il s'exécute avec la commande `exec("<chemin complet>/epr.sc` dans l'interface scilab (voir la GUI, soit la CLI lancée par `scilab-cli`).

Le code sera le suivant :

```
I = [1 0 ; 0 1]
H = (1/sqrt(2))*[1 1 ; 1 -1]
X = [ 0 1 ; 1 0]
CNOT = [ 1 0 0 0 ; 0 1 0 0 ; 0 0 0 1 ; 0 0 1 0 ]
EPR = CNOT*(H .*. I)
```

C.2.3 Bonus

1. Vérifier, par un programme et par un calcul par Scilab que $X = HZH$ et $Z = HXH$
2. Véfrifier que la porte SWAP peut se construire à l'aide de trois portes CNOT, par un programme et par le calcul

Annexe D

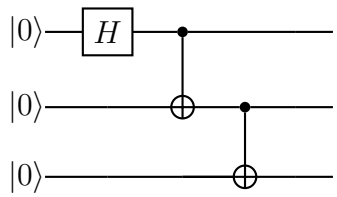
TD2 : Etats GHZ

Dans ce TD, on va s'intéresser aux états GHZ, qui sont des généralisations de la paire EPR.

D.1 Rappel sur l'état GHZ

L'état de Greenberger–Horne–Zeilinger, ou état GHZ, est une généralisation de la paire EPR.

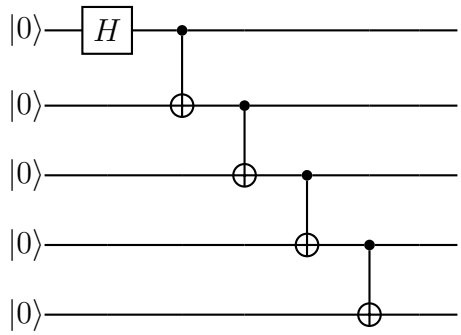
Sur 3 qubits, il s'agira de l'état $\frac{|000\rangle+|111\rangle}{\sqrt{2}}$, obtenu grâce au circuit suivant



Il est simple de généraliser cette construction en rajoutant un qubit associé au précédent par une porte CNOT et de construire l'état GHZ sur n qubits défini par

$$|GHZ_n\rangle = \frac{|00\cdots0\rangle + |11\cdots1\rangle}{\sqrt{2}} = \frac{|0\rangle^{\otimes n} + |1\rangle^{\otimes n}}{\sqrt{2}}$$

On peut faire un état GHZ à n qubits en "échellonant" portes CNOT entre deux qubits consécutits, par exemple, pour $n = 5$, on aura le circuit



Il va produire l'état $|GHZ_5\rangle = \frac{1}{\sqrt{2}}(|00000\rangle + |11111\rangle)$

L'état GHZ est souvent utilisé par les industriels pour démontrer leur capacité à construire du matériel capable d'intriquer un grand nombre de qubits. Le fait de pouvoir construire un "gros" état GHZ a donc valeur de benchmark.

L'état GHZ est utilisé dans certains protocoles de cryptographie quantique, comme le protocole des généraux byzantins.

Cet état revêt un sens particulier dans la physique quantique, car il s'agit d'un état maximalement intriqué.

D.2 Travail à réaliser

D.2.1 Utilisation de myQLM

Dans ce TDs, vous devrez

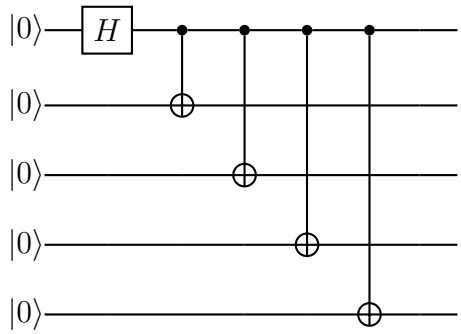
1. Utiliser le code du TD précédent pour construire un état GHZ à 3 qubits
2. Faire un code générique, via une fonction Python, qui génère le circuit à n qubits
3. Faire une évaluation du temps d'exécution du calcul de $|GHZ_n\rangle$ en fonction de n, l'utilisation de la bibliothèque `matploblib` est recommandée.

D.2.2 Utiilisation de Scilab

Dans le cas $n = 3$, calculer la matrice qui représente le circuit qui produit $|GHZ_3\rangle$.

D.2.3 Bonus

S'il vous reste du temps, réaliser le code qui produit un circuit à n qubits sur le schéma suivant (exemple donné pour $n = 5$)



Qu'observez-vous ?

Utiliser Scilab pour calculer la matrice correspondant à ce nouveau circuit pour $n = 3$. Qu'observez-vous ?

D.3 Solution

D.3.1 Circuit GHZ à 3 qubits

```
#!/usr/bin/env python
from qat.lang.AQASM import *
from qat.qpus import PyLinalg
import matplotlib.pyplot as plt

prog = Program()
qbits = prog.qalloc(3)

prog.apply(H, qbits[0])
prog.apply(CNOT, qbits[0], qbits[1])
prog.apply(CNOT, qbits[1], qbits[2])

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*100,2)
    i = i+1

plt.bar(states, probabilities, color='skyblue')
plt.xlabel('States')
plt.ylabel('Probabilities')
plt.title('Etat GHZ')
plt.show()
```

Cela produit l'image de la figure D.1

D.3.2 GHZ sur un nombre variable de qubits

Le code est le suivant

```
#!/usr/bin/env python
import time
from qat.lang.AQASM import *
from qat.qpus import PyLinalg
import matplotlib.pyplot as plt
```

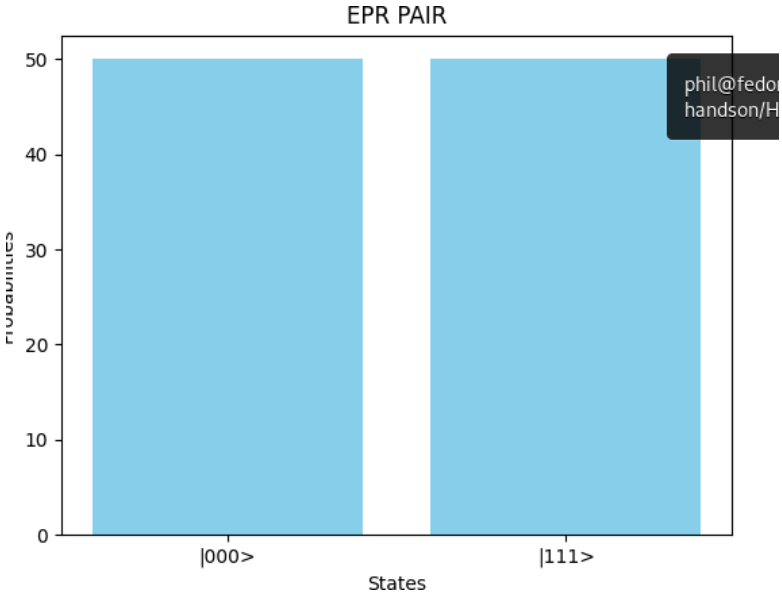


FIGURE D.1 – Sortie du script via matplotlib

```
def ghz (nq, p, q):
    p.apply(H,q[0])
    for i in range(nq-1):
        p.apply(CNOT, q[i], q[i+1])

def run_ghz (nqbits):
    prog = Program()
    qbits = prog.qalloc(nqbits)

    ghz(nqbits, prog, qbits)

    circuit = prog.to_circ()
    #circuit.display()

    job = circuit.to_job()

    linalgqpu = PyLinalg()

    result = linalgqpu.submit(job)
```

```
    for sample in result:
        print("State",sample.state,"with amplitude",
                sample.amplitude,"and probability",
                round(sample.probability*100,2),"%")

# Premier run à blanc, pour charger la lib myqlm
run_ghz(2)

# sampling
debut = 2
n = 28 ## 32 = limite pour myQLM, à 29 l'OOM killer shoote le process
elapsed = [0]*(n-debut)
run = ['']*(n-debut)

for i in range(debut, n):
    start = time.time()
    run_ghz(i)
    end = time.time()
    elapsed[i-debut] = end -start
    run[i-debut] = str(i)
    print("Run ", run[i-debut], "duree=", elapsed[i-debut], "%")

plt.bar(run, elapsed, color='skyblue')
plt.xlabel('runs')
plt.ylabel('elapsed time')
plt.title('Runs GHZ')
#plt.show()
plt.savefig('ghz-elapsed.png')
```

Le temps monte rapidement avec n, de manière exponentiel. On notera qu'il est simple de simuler jusqu'à une vingtaine de qubits, mais le coût augmente ensuite de manière exponentielle.

Sur mon PC, l'OOM killer dégomme le processus à partir de $n = 29$, myQLM n'autorise pas de dépasser la valeur $n = 32$.

On peut voir le temps *elapsed* sur les graphes des figures D.2 et D.3.

L'ouput brut du script est le suivant

```
State |00> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |00> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  2 duree= 0.0015747547149658203 %
State |000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  3 duree= 0.0016224384307861328 %
State |0000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  4 duree= 0.0016758441925048828 %
State |00000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  5 duree= 0.0016918182373046875 %
State |000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  6 duree= 0.0017495155334472656 %
State |0000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  7 duree= 0.002004384994506836 %
State |00000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  8 duree= 0.0021533966064453125 %
State |000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  9 duree= 0.002348184585571289 %
State |0000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  10 duree= 0.0028171539306640625 %
State |00000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  11 duree= 0.003998994827270508 %
State |000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  12 duree= 0.005914211273193359 %
State |0000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  13 duree= 0.009755849838256836 %
State |00000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  14 duree= 0.017910003662109375 %
State |000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  15 duree= 0.035486459732055664 %
State |0000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  16 duree= 0.11772727966308594 %
State |00000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  17 duree= 0.18035578727722168 %
State |000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  18 duree= 0.3017442226409912 %
State |0000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  19 duree= 0.5568468570709229 %
State |00000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  20 duree= 1.0492165088653564 %
State |000000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  21 duree= 2.0762128829956055 %
State |0000000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  22 duree= 4.186420440673828 %
State |00000000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  23 duree= 8.42594289779663 %
State |000000000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |111111111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  24 duree= 16.814985036849976 %
State |0000000000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |1111111111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  25 duree= 33.806443214416504 %
State |00000000000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  26 duree= 67.42673707008362 %
```

```
State |00000000000000000000000000> with amplitude (0.7071067811865475+0j) and probability 50.0 %
State |11111111111111111111111111> with amplitude (0.7071067811865475+0j) and probability 50.0 %
Run  27 duree= 135.76831150054932 %
```

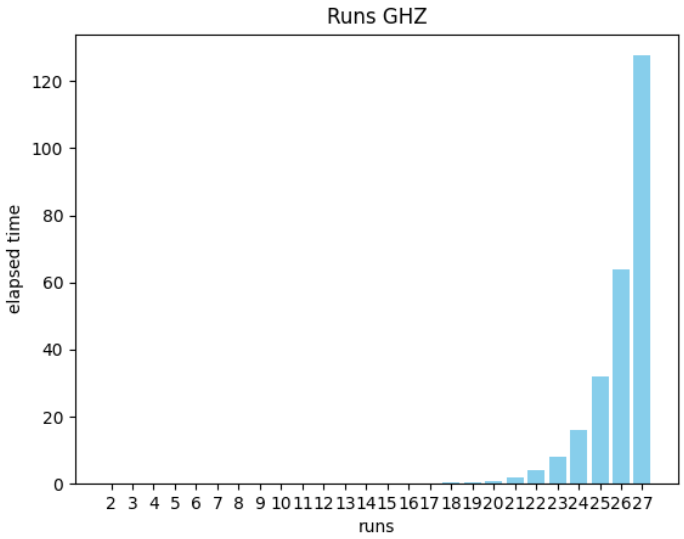


FIGURE D.2 – GHZ : Temps elapsed en 2 et 28 qubits

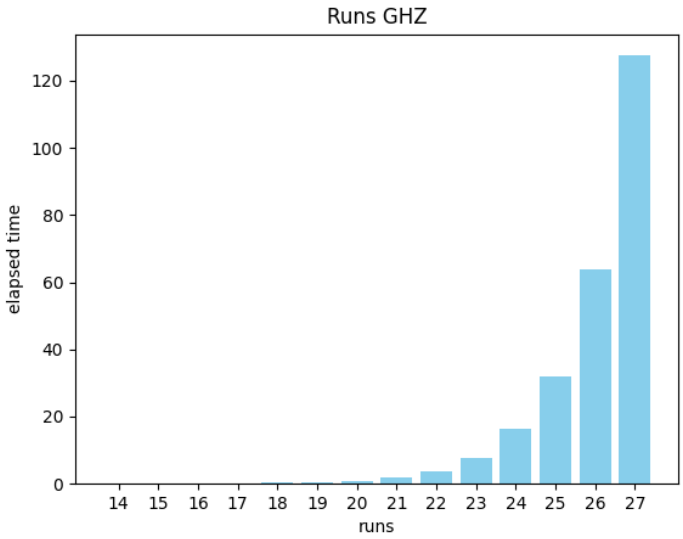


FIGURE D.3 – GHZ : Temps elapsed en 14 et 28 qubits

D.3.3 GHZ3 vu par Scilab

Le code Scilab est le suivant

```
I = [1 0 ; 0 1]
H = (1/sqrt(2))*[1 1 ; 1 -1]
X = [ 0 1 ; 1 0]
CNOT = [ 1 0 0 0 ; 0 1 0 0 ; 0 0 0 1 ; 0 0 1 0 ]
```

D.3.4 Solutions des Bonus

La version "alternative" du circuit produit le code suivant

```
#!/usr/bin/env python
from qat.lang.AQASM import *
from qat.qpus import PyLinalg
import matplotlib.pyplot as plt
```

```
def ghz (nq, p, q):
    p.apply(H,q[0])
    for i in range(nq-1):
        p.apply(CNOT, q[0], q[i+1])

nqbits = 4

prog = Program()
qbits = prog.qalloc(nqbits)

ghz(nqbits, prog, qbits)

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*100,2)
    i = i+1

plt.bar(states, probabilities, color='skyblue')
plt.xlabel('States')
plt.ylabel('Probabilities')
plt.title('GHZ State')
plt.show()
```

Si tous les qubits sont initialisés à $|0\rangle$, on construit également l'état $|GHZ_n\rangle$.

Le code Scillab sera le suivant pour 3 qubits

```
I = [1 0 ; 0 1]
H = (1/sqrt(2))*[1 1 ; 1 -1]
X = [ 0 1 ; 1 0]
CNOT = [ 1 0 0 0 ; 0 1 0 0 ; 0 0 0 1 ; 0 0 1 0 ]

HII = H .*. I .*. I
ICNOT = I .*. CNOT
CNOTI = CNOT .*. I

GHZ = ICNOT * CNOTI * HII

CNOT2 = [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 0 1 0 0 ;
     0 0 0 0 1 0 0 0 ;
     0 0 0 0 0 0 0 1 ;
     0 0 0 0 0 0 1 0 ]

GHZ2 = CNOT2 * CNOTI * HII
```

le code qui décrit la CNOT entre le premier et le second qubit est défini en écrivant les images des vecteurs de la base canonique par cet opérateur, ce dernier va swapper le troisième qubit si le premier vaut $|1\rangle$,

On observe que

```
GHZ2  =

   0.7071068   0.          0.          0.          0.7071068   0.          0.          0.
   0.          0.7071068   0.          0.          0.          0.7071068   0.          0.
   0.          0.          0.7071068   0.          0.          0.          0.7071068   0.
   0.          0.          0.          0.7071068   0.          0.          0.          0.7071068
   0.          0.          0.          0.7071068   0.          0.          0.         -0.7071068
   0.          0.          0.7071068   0.          0.          0.         -0.7071068   0.
   0.          0.7071068   0.          0.          0.         -0.7071068   0.          0.
   0.7071068   0.          0.          0.         -0.7071068   0.          0.          0.
```

Mais