De la théorie à la pratique: introduction au calcul quantique sur les NISQ et IBM Q expérience

Nicolas Macris

Faculté Informatique et Communication, EPFL

Dans le cadre des cours de Traitement Quantique de L'information et de Calcul Quantique

Bachelor 3ème année - Informatique et Communication.

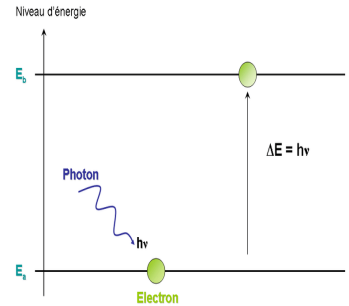
Plan

1. Qubits: rappels des principes de base
2. Le calcul quantique: un exemple simple
3. Noisy Intermediate Scale Quantum (NISQ) devices - IBM Q expérience
4. Qiskit: introduction et exemples
5. Perspectives

1. Qubits, rappel des principes de base

Qubit = unité de l'information quantique: $|0\rangle$, $|1\rangle$

cette information est "portée" par des systèmes physiques (photon, atome):



Principe de superposition: $\frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big)$, $\frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big)$, ou plus généralement $\alpha|0\rangle + \beta|1\rangle$ avec $|\alpha|^2 + |\beta|^2 = 1$.

Composition de plusieurs Qubits $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$ ($4 = 2^2$ états de base); $|000\rangle$, $|001\rangle$,... ($8 = 2^3$ états de base)

Combinons les deux principes:

$$|Bell\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big), \qquad |GHZ\rangle = \frac{1}{\sqrt{2}}\big(|000\rangle + |111\rangle\big)$$

les Qubits de $|Bell\rangle$, $|GHZ\rangle$, sont intriqués même si on les éloignent $\rightarrow$ **non-localité**. Plus généralement il y a un continuum d'états $\rightarrow$ **parallélisme quantique**

$$|\Psi\rangle = \alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|10\rangle + \alpha_4|11\rangle$$

Néanmoins...
au maximum n bits d'information classique peuvent être extraits

Mesures ou extraction de l'information: (entropie du résultat $\leq n$ avec ici $n = 2$)

$$|\Psi\rangle \rightarrow \text{MESURE} \rightarrow \begin{cases} |00\rangle & \text{prob}(00) = |\alpha_1|^2 \\ |01\rangle & \text{prob}(01) = |\alpha_2|^2 \\ |10\rangle & \text{prob}(10) = |\alpha_3|^2 \\ |11\rangle & \text{prob}(11) = |\alpha_4|^2 \end{cases}$$

L'information quantique est bizarre: ne peut pas être copiée, ne peut pas être effacée

Time-line de l'information quantique: précurseurs remontent à 1960 (Wiesner), 1970 (Holevo, Bennett, Ingarden,...), 1980 (Benioff, Manin, Feynman, Bennett, Brassard, Deutsch, ...), 1990 (Shor, Grover, ...), 2000 - 2020 (expériences + premières "machines",...), ... et en 2050... ???

https://en.wikipedia.org/wiki/Timeline of quantum computing

Aujourd'hui de nombreuses compagnies travaillent à son développement:

https://en.wikipedia.org/wiki/List of companies involved in quantum computing or communication

Plusieurs "processeurs" quantiques sont dévelopés:

https://en.wikipedia.org/wiki/List of quantum processors

2. Le calcul quantique (modèle des circuits)

Le principe du calcul quantique. Prenons par.ex la factorisation: soit $N \in \mathbb{N}$, trouver les facteurs premiers $p \cdot q = N$.

$$|O\rangle \to \text{UNITAIRE(N)} \to \text{MESURE} \to \text{candidats probables } p' \text{ et } q'$$

If $p' \cdot q' = N$ ok, else recommencez! L'art est de trouver UNITAIRE(N) pas trop "complexe" et tel que $\text{Prob}(p' = p, q' = q) \geq 1 - \epsilon$.
Cela est possible avec une complexité $O(\log(1/\epsilon)(\log N)^3)$ *(Shor).*

Un joli problème "jouet". Soit $a_1, \cdots, a_n$ bits secrets. Découvrir leurs valeurs en évaluant la fonction

$$f(x_1, \cdots, x_n) = a_1 x_1 + \cdots + a_n x_n$$

un nombre minimum de fois. *Une solution optimale est d'évaluer:*

$$f(1, 0, \ldots, 0) \to a_1, \quad f(0, 1, \ldots, 0) \to a_2, \quad \ldots \quad , f(0, 0, \ldots, 1) \to a_n$$

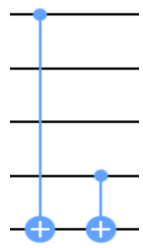
La "complexité" classique est égale à n.

Soit $(a_1, a_2, a_3, a_4) = (1, 0, 0, 1)$

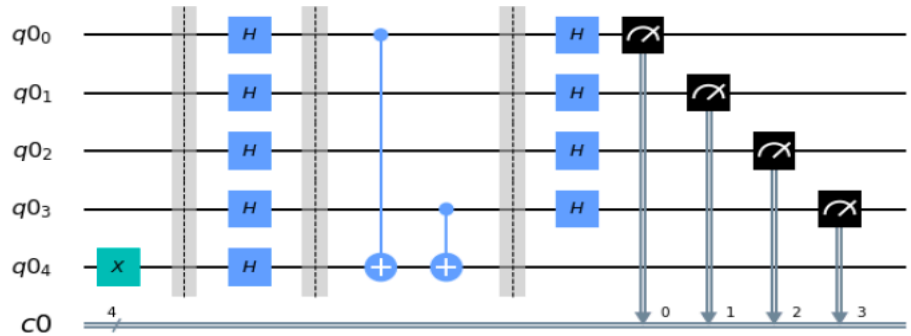
un circuit classique

pour évaluer

$f(x_1, x_2, x_3, x_4) = x_1 + x_4$



Exploitons le parallelisme quantique (Bernstein - Vazirani)



- on pose toutes les questions possibles car $H|0| = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
- on évalue f une seule fois
- on fait une analyse de "Fourier-Walsh" du "signal"

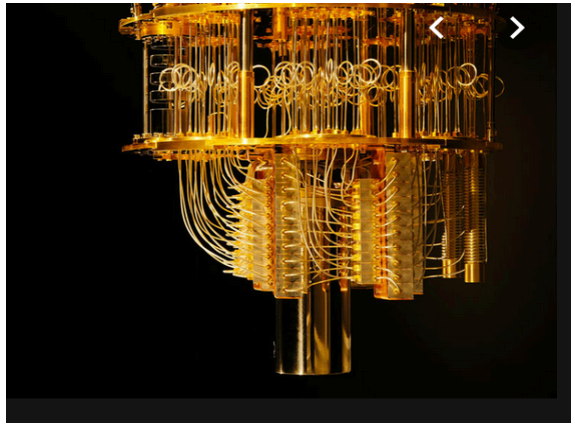
Une mesure donne les bits secrets $(1, 0, 0, 1)$ avec certitude !!!

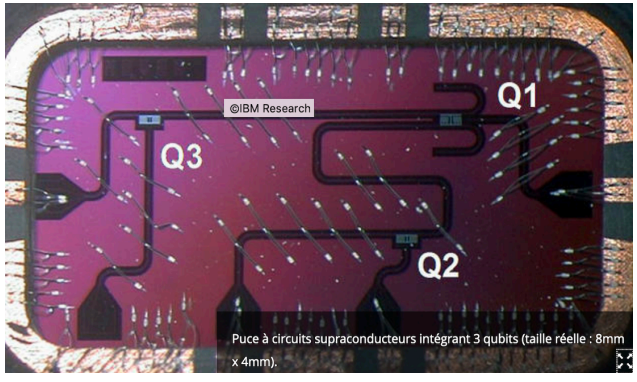
3. Machines NISQ - et exploration de IBM Q expérience

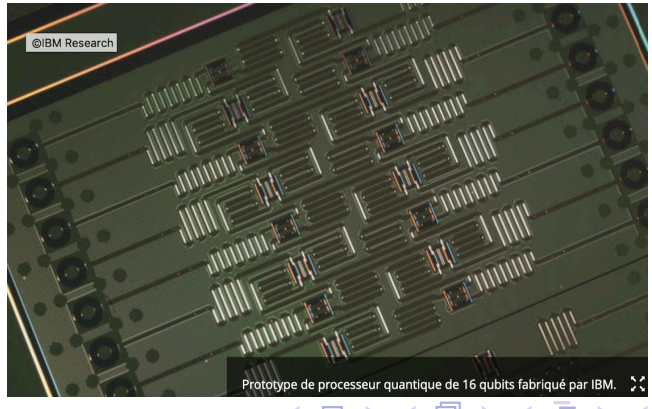
- NISQ = Noisy Intermediate Scale Quantum devices (terminologie due à J. Preskill à Caltech). Fabriqués par IBM, Google, Intel, Rigetti,...
- Petits ordinateurs quantiques qui réalisent des versions bruitées (noisy) du modèle des circuits avec un petit nombre de Qubits de l'ordre de 5 à 20 à 50 (intermediate scale).
- Les Qubits sont basés sur des technologies supraconductrices (jonctions Josephson de taille de l'ordre de quelques microns).
- Imprimés sur une puce (taille de quelques mm) à $T \approx 10\mathrm{mK}$ et connectés entre eux et au monde extérieur par des résonateurs de radiofréquences.


IBM Q


IBM Q




Q1
Q3
Q2
©IBM Research
Puce à circuits supraconducteurs intégrant 3 qubits (taille réelle : 8mm x 4mm).


©IBM Research
Prototype de processeur quantique de 16 qubits fabriqué par IBM.

IBM Q experience

https://www.ibm.com/quantum-computing/technology/experience/

Interview dans Nature 6 Mars 2017:

"**The project builds on know-how developed around IBMs existing cloud computing service: Quantum Experience, which anyone can access for free.**
That system went online in May 2016 and recently received an upgraded user interface. Having it up for ten months has taught us a lot, says physicist Jerry Chow, who leads the quantum-computing laboratory at IBMs research centre in Yorktown Heights, New York.
It has provided a way for researchers around the world to practise building quantum algorithms without access to their own quantum computer. IBMs overall strategy is to build a community and an ecosystem around its technology, Chow says."

4. Qiskit: introduction et exemples

Qiskit [quantum information science kit] *est un language basé sur python. Les lignes de codes sont traduites en* **quasm [quantum assembly language]** *puis converties en pulses de radio fréquences pour réaliser les opérations unitaires, et mesures finales.*

- Environnement open source, communauté de "Qiskiters"
- Concevoir des expériences, les simuler sur le *simulateur* et les réaliser sur les *vraies machines*.
- Quatre éléments fondateurs de Qiskit
  https://qiskit.org/documentation/index.html

  1. *Qiskit Terra*: Composing quantum programs at the level of circuits and pulses with the code foundation.
  2. *Qiskit Aer* : Accelerating development via simulators, emulators, and debuggers
  3. *Qiskit Ignis*: Addressing noise and errors
  4. *Qiskit Aqua*: Building algorithms and applications

Démos sur l'interface graphique (composer) et puis sur Qiskit

1. Illustration du parallélisme quantique: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $H \otimes H|00\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$, etc...
2. Nous allons implémenter des petits circuits créant les états:

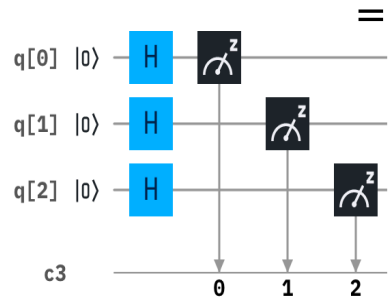
$$|Bell\rangle = \frac{1}{2}(|00\rangle + |11\rangle), \qquad |GHZ\rangle = \frac{1}{2}(|000\rangle + |111\rangle)$$

3. Puis un circuit implémentant l'algorithme de Bernstein-Vazirani: *le problème est de trouver les bits $a_1, a_2, \ldots, a_n$ en posant "une seule question" à la fonction*

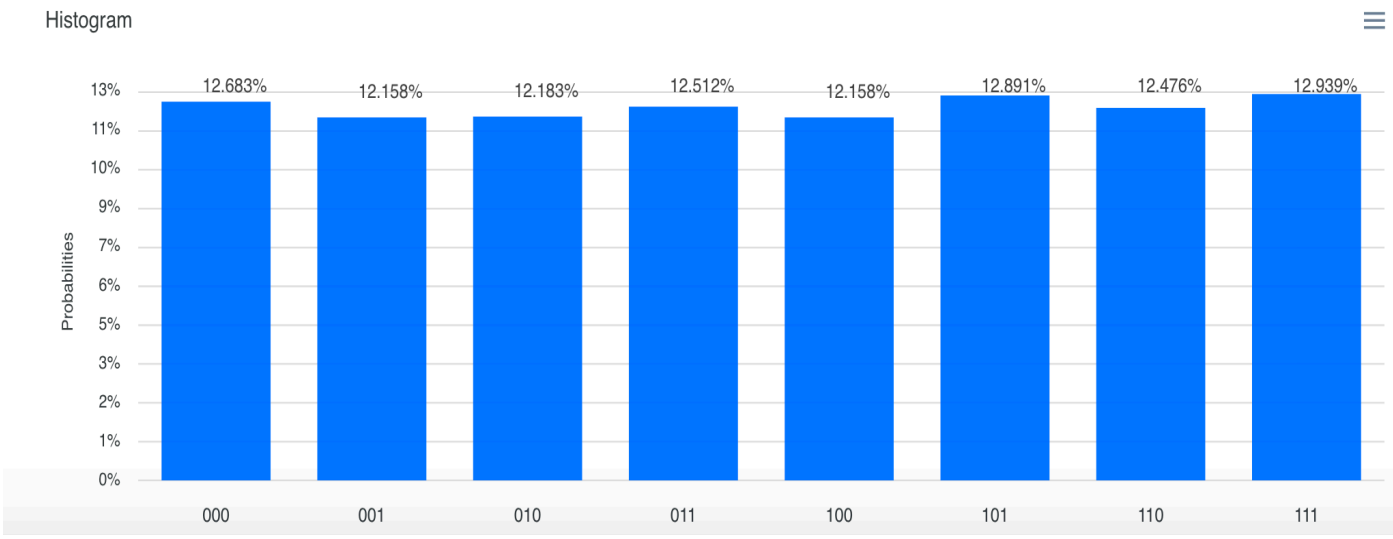
$$f(x_1, \cdots, x_n) = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

Une série de 9 vidéos très pédagogiques d'Abraham Asfaw (2h en totalité): https://www.youtube.com/playlist?list=PLOFEBzvs-Vvp2xg9-POLJhQwtVktlYGbY

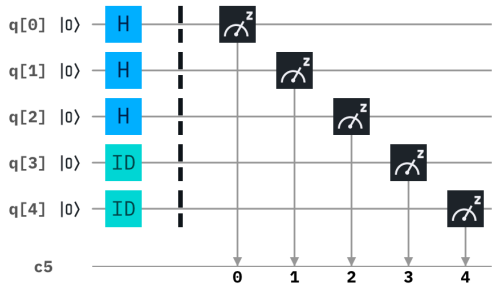
Parallélisme quantique: superposition de 8 états classiques (SIMULATEUR idéal, 8192 mesures)

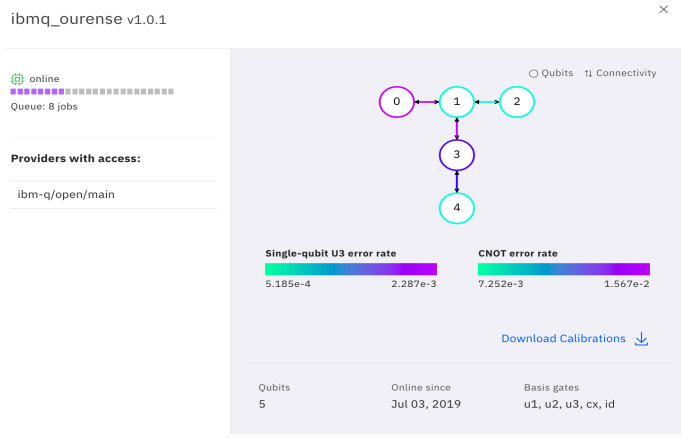


Result



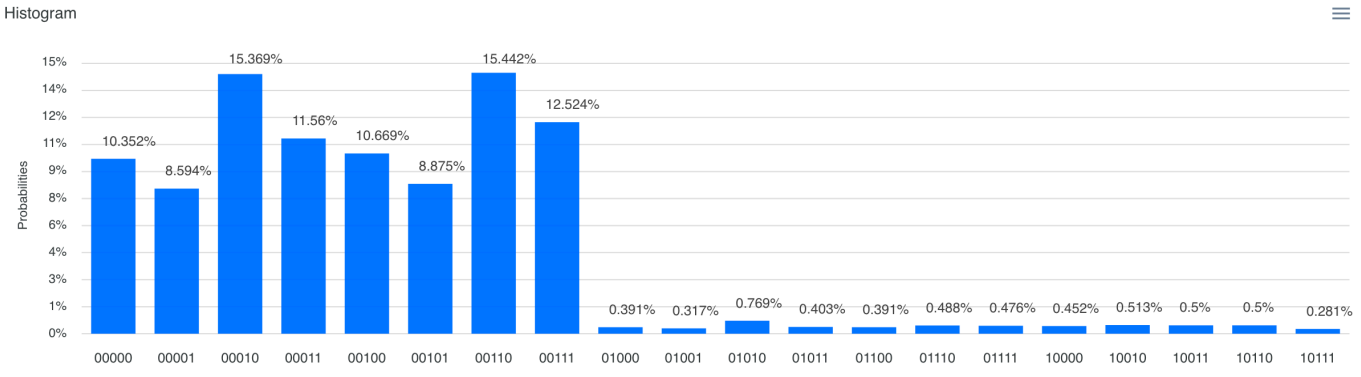
Parallélisme quantique: superposition de 8 états classiques + du bruit (device OURENSE, 8024 mesures)





Result

Displaying a subset of the 31 counts. Download the result to see the full details.



Etat GHZ: (SIMULATEUR, 1024 mesures)

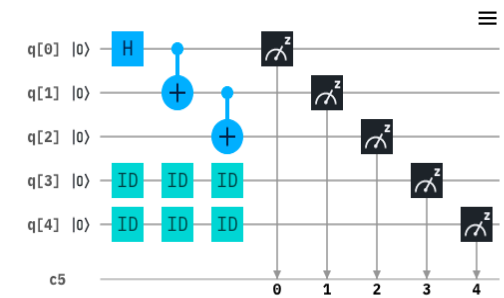
Created	Transpiling	Validating	In queue	Running	Completed
	705ms	710ms	1.3s	8ms	

Run details

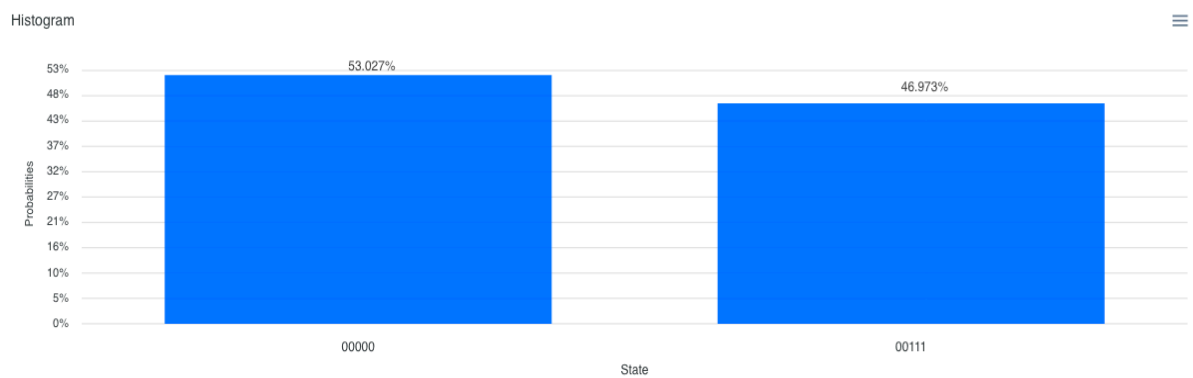
Backend	Shots	Status	Time taken	Last Update
ibmq_qasm_simulator		COMPLETED	3s	Nov 13, 2019 10:30 PM

Circuit diagram

Original circuit diagram



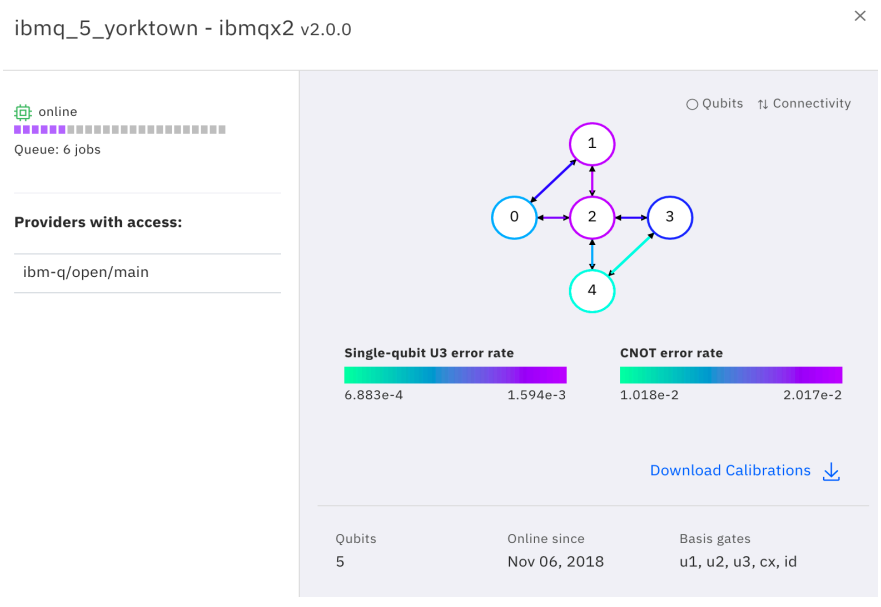
Result



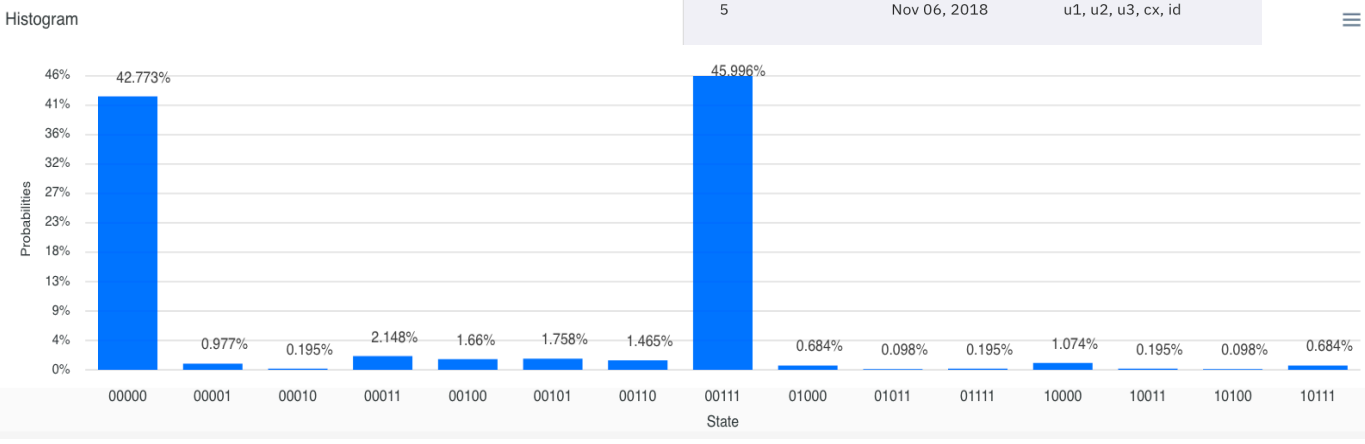
Etat GHZ: (device IBMQX2, 1024 mesures)

Original circuit diagram

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[5];
creg c[5];

h q[0];
id q[3];
id q[4];
cx q[0],q[1];
id q[3];
id q[4];
cx q[1],q[2];
id q[3];
id q[4];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
```



Result



Code Qiskit pour GHZ: (simulation + expérience)

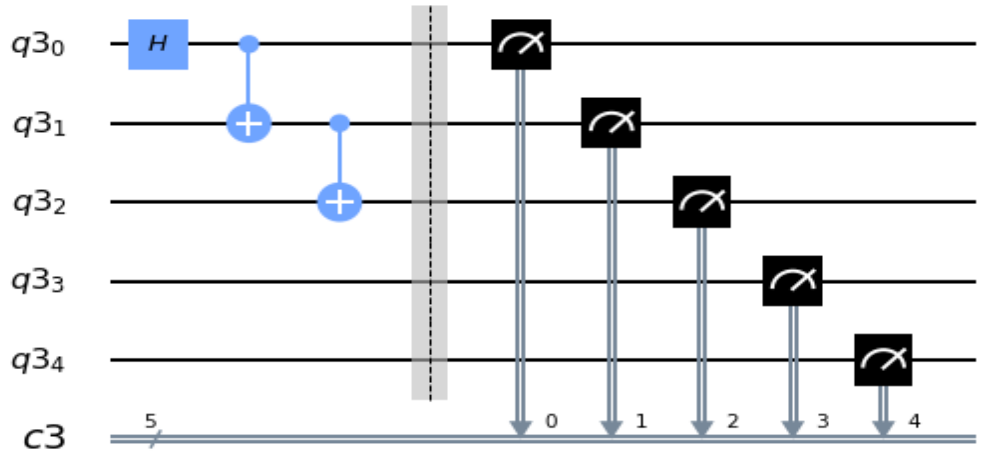
SEV-GHZ

November 14, 2019

[11]:
```
%matplotlib inline
# Importing standard Qiskit libraries and configuring account
# from qiskit import QuantumRegister, ClassicalRegister (included in␣
 ↪QuantumCircuit ?)
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

[12]:
```
#creation of the circuit
n = 5
q = QuantumRegister(n)
c = ClassicalRegister(n)
GHZcirc = QuantumCircuit(q, c)

#unitary operations
GHZcirc.h(q[0])
GHZcirc.cx(q[0], q[1])
GHZcirc.cx(q[1],q[2])

#optional
GHZcirc.barrier(q)

#measurement
GHZcirc.measure(q[0], c[0])
GHZcirc.measure(q[1], c[1])
GHZcirc.measure(q[2], c[2])
GHZcirc.measure(q[3], c[3])
GHZcirc.measure(q[4], c[4])

#visualisation of the circuit
GHZcirc.draw()
```

[12]:

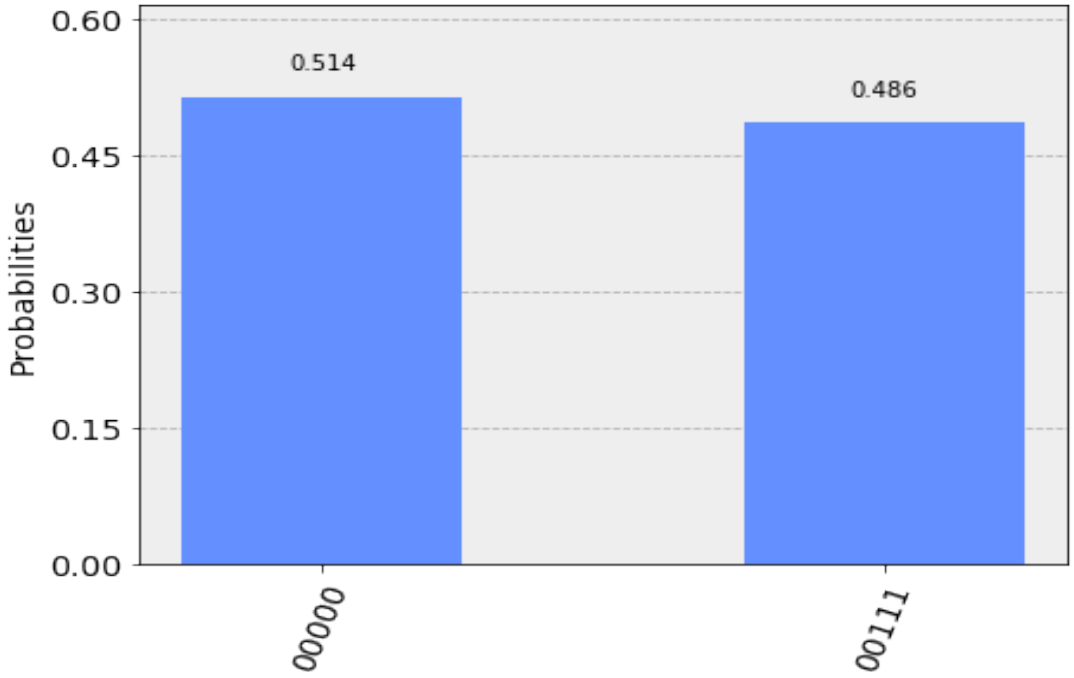


[13]:
```
#classical ideal simulation
backend = Aer.get_backend('qasm_simulator')
job_sim = execute(GHZcirc, backend)
sim_result = job_sim.result()
```

[14]:
```
#print and visualaise measurement results (an application of the Born rule)
print(sim_result.get_counts(GHZcirc))
plot_histogram(sim_result.get_counts(GHZcirc))
```

```
{'00111': 498, '00000': 526}
```

[14]:



[20]:
```
#find available devices for the experiment
print("Available backends:")
#previously we did `IBMbackends()` this is dicsouraged now
IBMQ.get_provider()
provider.backends()
```

```
Available backends:
```

[20]:
```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_london') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_burlington') from IBMQ(hub='ibm-q', group='open',
project='main')>,
```

```
 <IBMQBackend('ibmq_essex') from IBMQ(hub='ibm-q', group='open',
project='main')>]
```

[23]:
```
#trick to select a device so as to minimize queuing
from qiskit.providers.ibmq import least_busy

large_enough_devices = provider.backends(filters=lambda x: x.configuration().
 ↪n_qubits > 2 and
                                                    not x.configuration().
 ↪simulator)
backend = least_busy(large_enough_devices)
print("The best backend is " + backend.name())
```

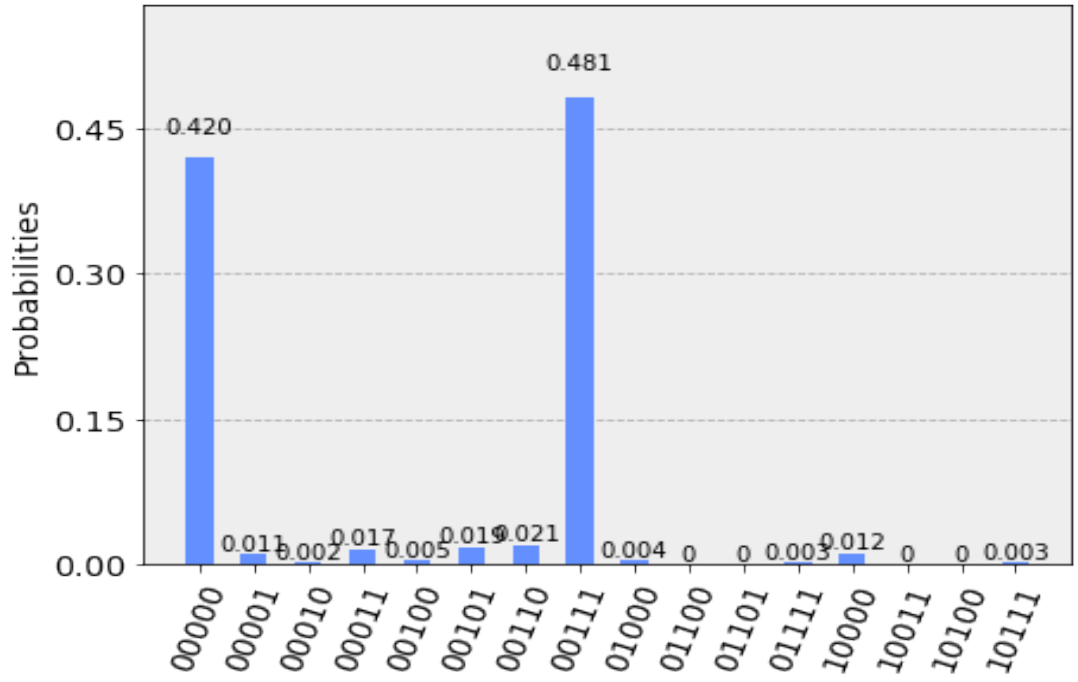
```
The best backend is ibmqx2
```

[24]:
```
#Run the job
from qiskit.tools.monitor import job_monitor
shots = 1024          # Number of shots to run the program (experiment); maximum␣
 ↪is 8192 shots.
max_credits = 3        # Maximum number of credits to spend on executions.

job_exp = execute(GHZcirc, backend=backend, shots=shots,␣
 ↪max_credits=max_credits)
job_monitor(job_exp)
```

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

[25]:
```
#print experimental result
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(GHZcirc)

plot_histogram([counts_exp])
```

[25]:


Probabilities
0.00
0.15
0.30
0.45
0.420
0.011
0.002
0.017
0.005
0.019
0.021
0.481
0.004
0
0
0.003
0.012
0
0
0.003
00000
00001
00010
00011
00100
00101
00110
00111
01000
01100
01101
01111
10000
10011
10100
10111

[ ]:

Code Qiskit pour l'algorithme de Bernstein-Vazirani: (simulation + expérience)

SEV-Bernstein-Vazirani-small

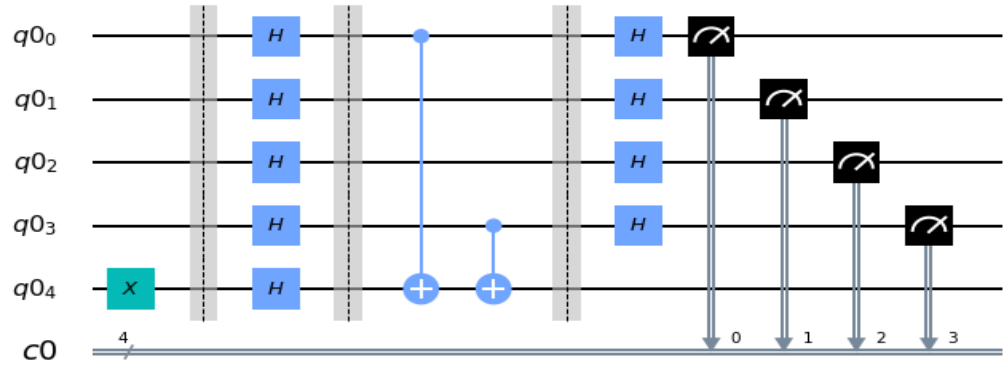
November 14, 2019

[1]:
```
%matplotlib inline
# Importing standard Qiskit libraries and configuring account
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *

# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

[2]:
```
#the unknown bit string a = 'b1,...,bn'
a = '1001'

#creation of the circuit
n = len(a)
q = QuantumRegister(n+1)
c = ClassicalRegister(n)
circuit = QuantumCircuit(q, c)

#unitary operations
circuit.x(q[n]) #initialisation of ancilla qubit
circuit.barrier(q)
circuit.h(range(n+1)) #exploitation of quantum parallelism
circuit.barrier(q)

#unitary representation of the function f(x1,...,xn) = a1 x1 + ... + an xn and␣
 ↪result registered in ancilla qubit
for ii, yesno in enumerate(reversed(a)):
    if yesno == '1':
        circuit.cx(q[ii], q[n])

circuit.barrier()
circuit.h(range(n)) #"Fourier-Walsh" analysis of state vector after the barrier

#measurement of final state vector
circuit.measure(range(n),range(n))
```

```
#visualisation of the circuit
circuit.draw()
```

[2]:

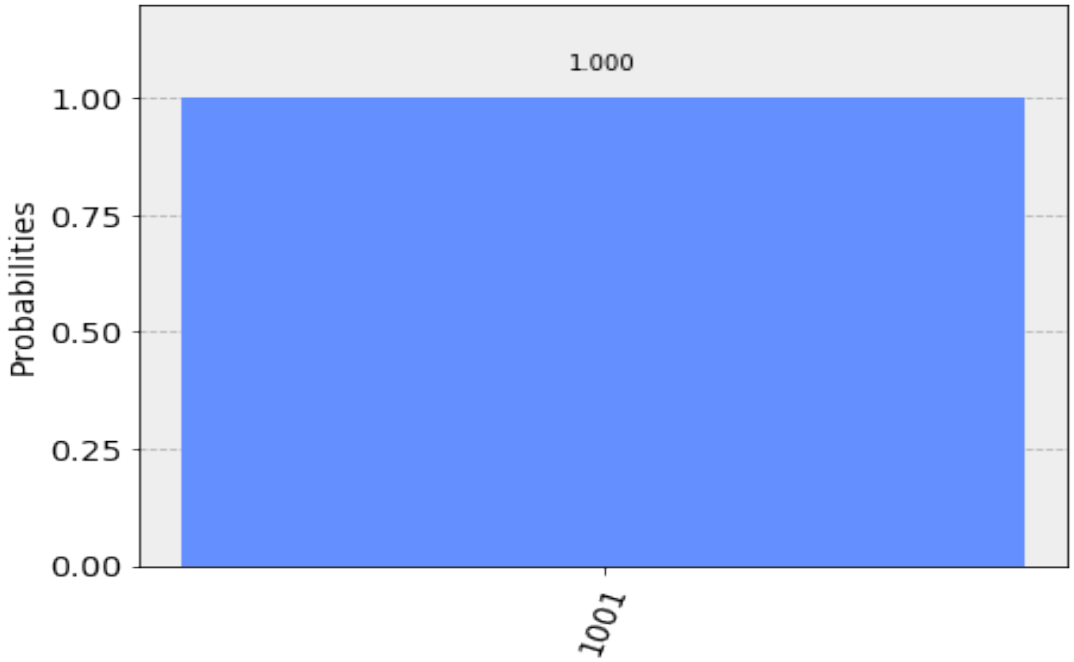


[3]:
```
#classical ideal simulation
backend = Aer.get_backend('qasm_simulator')
job_sim = execute(circuit, backend, shots=1)
sim_result = job_sim.result()
```

[4]:
```
#print and visualaise measurement results (an application of the Born rule)
print(sim_result.get_counts(circuit))
plot_histogram(sim_result.get_counts(circuit))
```

```
{'1001': 1}
```

[4]:



[5]:
```
#find available devices for the experiment
print("Available backends:")
#previously we did `IBMbackends()` this is dicsouraged now
IBMQ.get_provider()
provider.backends()
```

```
Available backends:
```

[5]:
```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_london') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_burlington') from IBMQ(hub='ibm-q', group='open',
project='main')>,
 <IBMQBackend('ibmq_essex') from IBMQ(hub='ibm-q', group='open',
```

```
project='main')>]
```

[6]:
```
#trick to select a device so as to minimize queuing
from qiskit.providers.ibmq import least_busy

large_enough_devices = provider.backends(filters=lambda x: x.configuration().
 ↪n_qubits > 2 and
                                                    not x.configuration().
 ↪simulator)
backend = least_busy(large_enough_devices)
print("The best backend is " + backend.name())
```

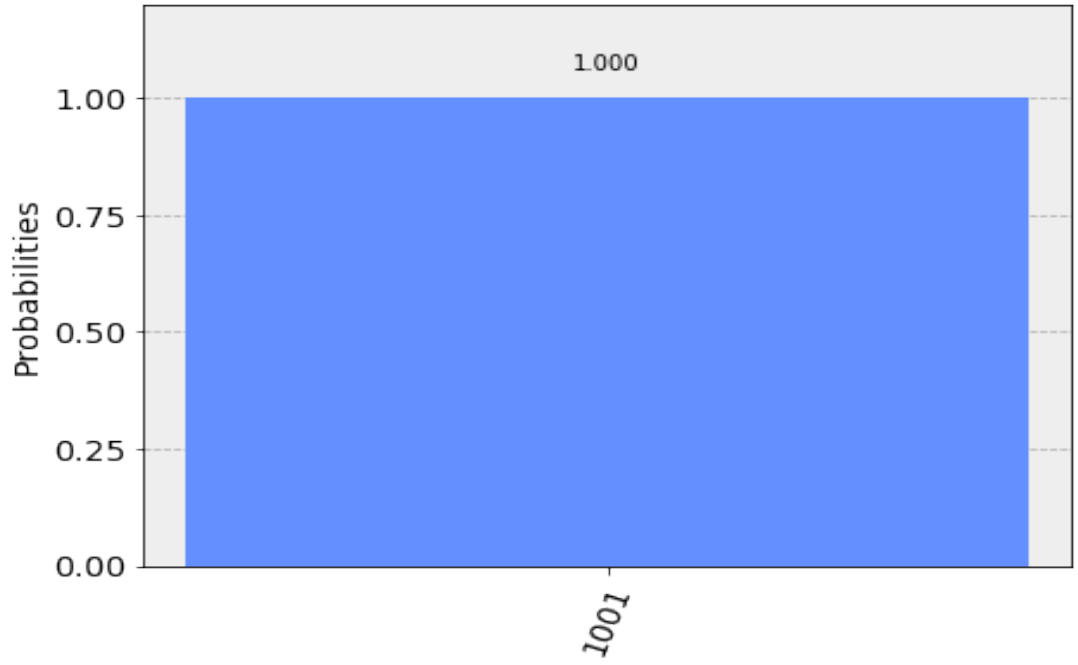
```
The best backend is ibmq_ourense
```

[7]:
```
#Run the job
from qiskit.tools.monitor import job_monitor
shots = 1         # Number of shots to run the program (experiment); maximum is␣
 ↪8192 shots.
max_credits = 3         # Maximum number of credits to spend on executions.

job_exp = execute(circuit, backend=backend, shots=shots,␣
 ↪max_credits=max_credits)
job_monitor(job_exp)
```

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

[8]:
```
#print experimental result
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(circuit)

plot_histogram([counts_exp])
```

[8]:

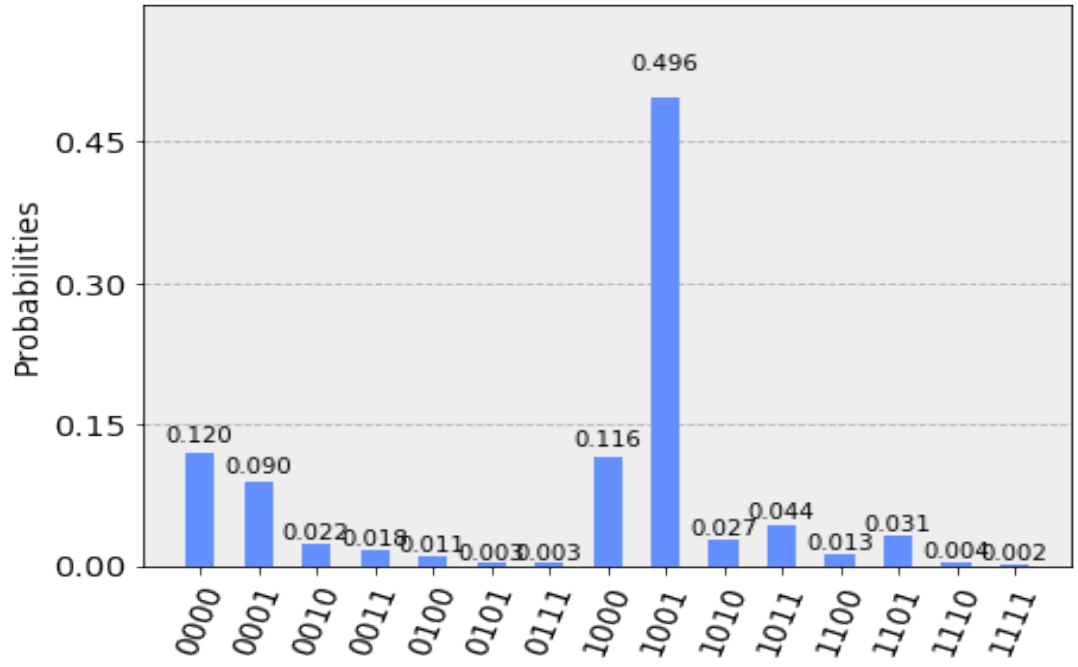


[9]:
```
#Run the job
from qiskit.tools.monitor import job_monitor
shots = 1024        # Number of shots to run the program (experiment); maximum␣
 ↪is 8192 shots.
max_credits = 3         # Maximum number of credits to spend on executions.

job_exp = execute(circuit, backend=backend, shots=shots,␣
 ↪max_credits=max_credits)
job_monitor(job_exp)
```

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

[10]:
```
#print experimental result
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(circuit)

plot_histogram([counts_exp])
```

[10]:


Probabilities
0.00
0.15
0.30
0.45
0.120
0.090
0.022
0.018
0.011
0.003
0.003
0.116
0.496
0.027
0.044
0.013
0.031
0.004
0.002
0000
0001
0010
0011
0100
0101
0111
1000
1001
1010
1011
1100
1101
1110
1111

[ ]:

5. Perspectives

1. Applications révées des ordinateurs quantiques (déja réalisées à l'échelle des NISQ)

   - Calcul, algorithmique (p.ex Factorisation,...)
   - Optimisation
   - Simulation de "l'équation de Schroedinger" (chimie quantique, physique nucléaire,...)
   - Echantillonnage (finance, générateurs de nombres aléatoires, machine learning,...)

2. NISQ sont un petit laboratoire pour

   - Mieux comprendre et modéliser la décohérence (i.e., le bruit, l'influence de l'environnement, la transition vers la physique classique)
   - Mettre en évidence la "suprématie quantique" ou plus modestement "l'avantage quantique"

3. NISQ sont aussi un outil fantastique pour l'enseignement de l'information et du calcul quantique

   - Les étudiants participent et contribuent à un écosystème et une communauté de "Qiskiters"
   - Projets interdisciplinaires: physique, maths, ingénieurs et informaticiens

     Exemple 1: traduire un circuit théorique pour tenir compte des contraintes géométriques de la machine

     Exemple 2: Optimiser la complexité tout en tenant compte des contraintes

   - Nous avons mené des projets d'équipe d'étudiants (physique et informatique et communication) en 2018 et 2019 (avec Dr M. A. Dupertuis)
   - Nos étudiants ont participé aux Qiskit Camps et Hackathons (Vermont 2018, Piz Gloria 2019, CERN)

4. l'EPFL est membre du IBM Q network depuis juillet 2019.

References

Livres recents

"Learn Quantum Computation using Qiskit" A university quantum algorithms/computation course supplement based on Qiskit, https://community.qiskit.org/textbook/

"Mastering Quantum Computing with IBM QX" Explore the world of quantum computing using the Quantum Composer and Qiskit, par Dr Christine Corbett Moran, Pakt Ed. January 2019

Articles sur l'informatique quantique

"Open Quantum Assembly Language" Andrew W. Cross, Lev S. Bishop, John A. Smolin, Jay M. Gambetta (2017) https://arxiv.org/abs/1707.03429

"Open source software in quantum computing" Mark Fingerhuth, Tomas Babej, Peter Wittek (2018) https://arxiv.org/abs/1812.09167v1 PLoS ONE 13(12): e0208561

La technologie des qubits supraconducteurs

"A Quantum Engineer's Guide to Superconducting Qubits" Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P. Orlando, Simon Gustavsson, William D. Oliver (2019) https://arxiv.org/abs/1904.06560 Applied Physics Reviews 6, 021318 (2019)