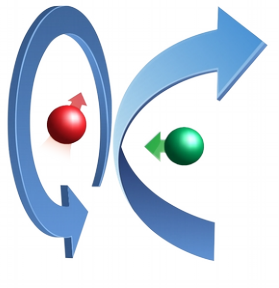
14e Journées Informatiques IN2P3/IRFU

14-17 novembre 2022, Le Croisic, domaine de Port au Rocs

Programmation des processeurs quantiques

Bogdan VULPESCU (Laboratoire de Physique de Clermont)

pour le MP QC2I (Quantum Computing pour les Deux Infinis)



L'abstraction RAM (Random Access Machine)

problème ⇒ algorithme ⇒ programme ⇒ exécution + I/O sur une machine

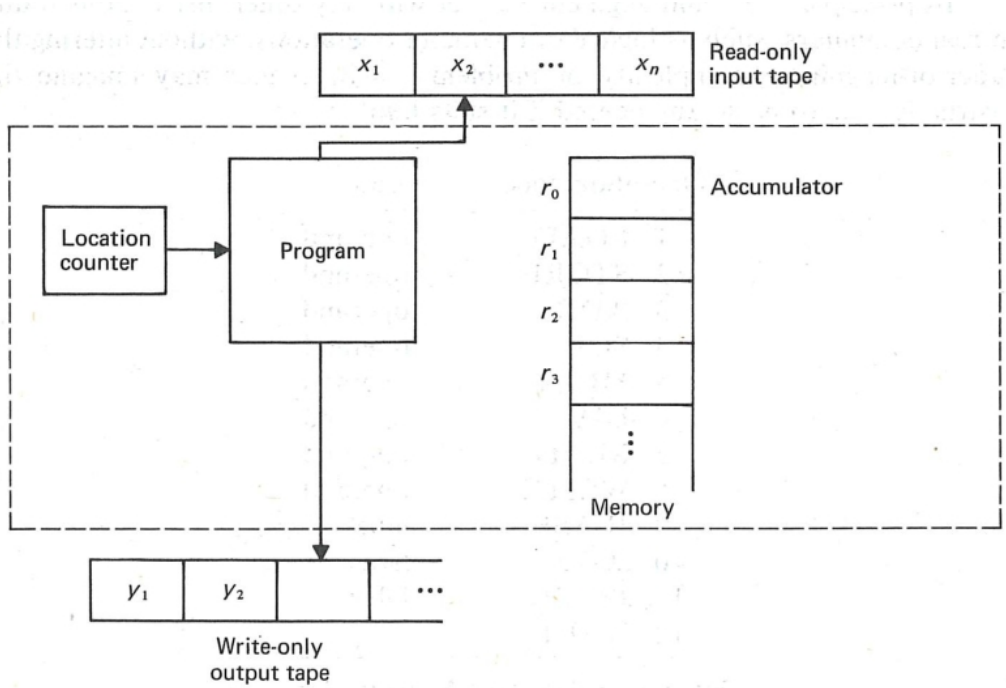


Fig. 1.3 A random access machine.

Instruction	Meaning
1. LOAD a	$c(0) \leftarrow v(a)$
2. STORE i	$c(i) \leftarrow c(0)$
STORE $*i$	$c(c(i)) \leftarrow c(0)$
3. ADD a	$c(0) \leftarrow c(0) + v(a)$
4. SUB a	$c(0) \leftarrow c(0) - v(a)$
5. MULT a	$c(0) \leftarrow c(0) \times v(a)$
6. DIV a	$c(0) \leftarrow \lfloor c(0)/v(a) \rfloor$†
7. READ i	$c(i) \leftarrow$ current input symbol.
READ $*i$	$c(c(i)) \leftarrow$ current input symbol. The input tape head moves one square right in either case.
8. WRITE a	$v(a)$ is printed on the square of the output tape currently under the output tape head. Then the tape head is moved one square right.
9. JUMP b	The location counter is set to the instruction labeled b.
10. JGTZ b	The location counter is set to the instruction labeled b if $c(0) > 0$; otherwise, the location counter is set to the next instruction.
11. JZERO b	The location counter is set to the instruction labeled b if $c(0) = 0$; otherwise, the location counter is set to the next instruction.
12. HALT	Execution ceases.

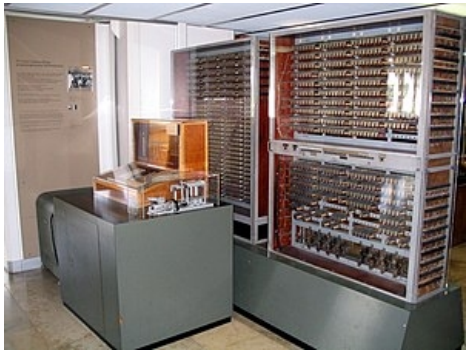
† Throughout this book, $\lceil x \rceil$ (*ceiling* of x) denotes the least integer equal to or greater than x, and $\lfloor x \rfloor$ (*floor*, or *integer part* of x) denotes the greatest integer equal to or less than x.

Fig. 1.5. Meaning of RAM instructions. The operand a is either $=i$, i, or $*i$.

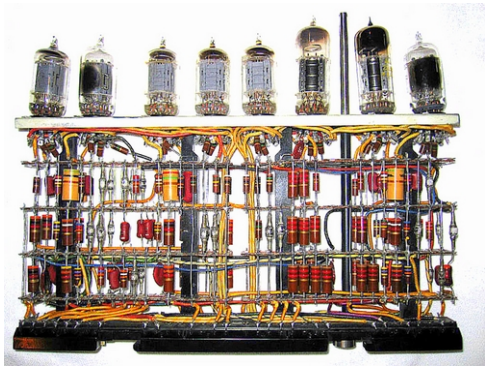
The design and analysis of computer algorithms

Aho, Hopcroft, Ullman, 1974

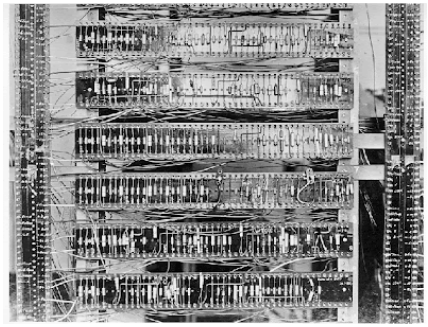
Le processeur classique



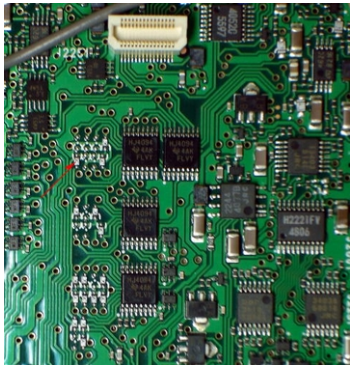
électro-mécanique



tubes



transistors



circuits intégrés

Set universel de portes : AND, OR, NOT, FANOUT (ou COPY)

$f:\{0,1\}^m \rightarrow \{0,1\}^n$ equivalent à $f_i:\{0,1\}^m \rightarrow \{0,1\}$ $(i=1,2,\ldots,n)$

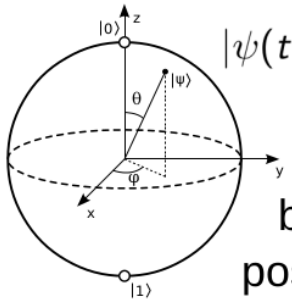
pour l'argument: $a=(a_{m-1},a_{m-2},\ldots,a_1,a_0)$ on calcule le minterms: $f_i^{(l)}(a)$

$f_i^{(l)}(a)=1$ si $a=a^{(l)}$ et 0 autrement, pour toute valeur de l'argument a du domaine de f

exemple: $a^{(l)}=110100\ldots001$ → $f_i^{(l)}(a)=a_{m-1}\wedge a_{m-2}\wedge\overline{a_{m-3}}\wedge a_{m-4}\wedge a_{m-5}\ldots\wedge\overline{a_2}\wedge\overline{a_1}\wedge a_0$

et finalement: $f_i(a)=f_i^{(1)}\vee f_i^{(2)}\vee\ldots\vee f_i^{(k)}$

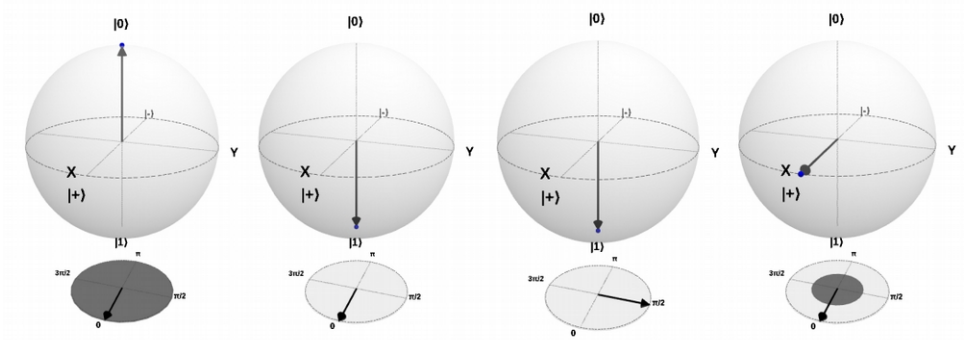
Les « opérations » quantiques



$|\psi(t)\rangle = \alpha(t)\,|0\rangle + \beta(t)\,|1\rangle$ Un qubit est décrit par une **fonction d'onde**, en général une combinaison linéaire des deux états possibles. La **sphère Bloch** offre une interprétation intuitive pour les **vecteurs d'état** situés sur sa surface

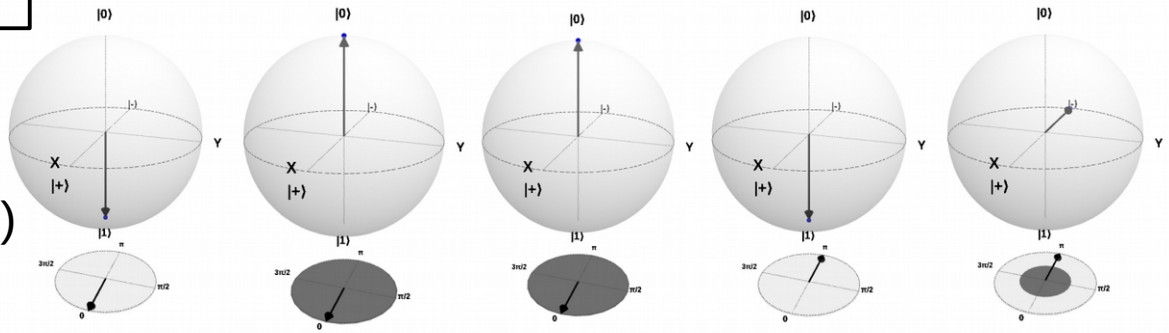
$$|\psi\rangle = \cos\tfrac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\tfrac{\theta}{2}\,|1\rangle = \begin{bmatrix}\cos\frac{\theta}{2}\\ e^{i\phi}\sin\frac{\theta}{2}\end{bmatrix}$$

l'espace des états d'un seul qubit étant un espace vectoriel à 2 dimensions.



$\theta = 0$
$\phi = 0$

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \qquad X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} \qquad Y = \begin{bmatrix}0 & -i\\i & 0\end{bmatrix} \qquad H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$



$\theta = \pi$
$\phi = 0$

$$|1\rangle = \begin{bmatrix}0\\1\end{bmatrix} \qquad X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} \qquad Y = \begin{bmatrix}0 & -i\\i & 0\end{bmatrix} \qquad Z = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix} \qquad H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

Un seul qubit :

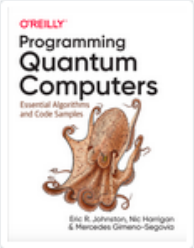
- espace de valeurs infini (**superposition**)
- transformations unitaires

Deux qubits :

- **interférence** (phase)
- **intrication** (communication)

Quel langage ?

Docs O'Reilly Buy Book Engines Errata Contact



Programming Quantum Computers

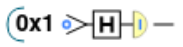
Code Samples

Run Program | Ex 2-1: Random bit | QCEngine

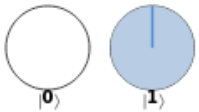
```
1  // Programming Quantum Computers
2  //   by Eric Johnston, Nic Harrigan and Mercedes Gimeno-Segovia
3  //   O'Reilly Media
4
5  // To run this online, go to http://oreilly-qc.github.io?p=2-1
6
7  // This sample generates a single random bit.
8
9  qc.reset(1);         // allocate one qubit
10 qc.write(0);         // write the value zero
11 qc.had();           // place it into superposition of 0 and 1
12 var result = qc.read();  // read the result as a digital bit
13
```

Tirage pile ou face avec une superposition égale de 0 et 1

Program circuit



Circle notation



https://oreilly-qc.github.io/

Qiskit https://qiskit.org/

```python
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, Aer, IBMQ, BasicAer
import math
## Uncomment the next line to see diagrams when running in a notebook
#%matplotlib inline

## Example 2-1: Random bit
# Set up the program
reg = QuantumRegister(1, name='reg')
reg_c = ClassicalRegister(1, name='regc')
qc = QuantumCircuit(reg, reg_c)

qc.reset(reg)          # write the value 0
qc.h(reg)              # put it into a superposition of 0 and 1
qc.measure(reg, reg_c) # read the result as a digital bit

backend = BasicAer.get_backend('statevector_simulator')
job = execute(qc, backend)
result = job.result()

counts = result.get_counts(qc)
print('counts:',counts)

outputstate = result.get_statevector(qc, decimals=3)
print(outputstate)
qc.draw()        # draw the circuit
```

Cirq https://quantumai.google/cirq

https://github.com/quantumlib/cirq

```python
def main():
    qc = QPU()
    qc.reset(1)
    qc.had()  # put it into a superposition of 0 and 1
    qc.read() # read the result as a digital bit

    qc.draw() # draw the circuit
    result = qc.run() # run the circuit
    print(result)


######################################################################
## The below class is a light interface, to convert the
## book's syntax into the syntax used by Cirq.
class QPU:
    def __init__(self):
        self.circuit = cirq.Circuit()
        self.simulator = cirq.Simulator()
        self.qubits = None

    def reset(self, num_qubits):
        self.qubits = [cirq.GridQubit(i, 0) for i in range(num_qubits)]

    def mask_to_list(self, mask):
        return [q for i,q in enumerate(self.qubits) if (1 << i) & mask]

    def had(self, target_mask=~0):
        target = self.mask_to_list(target_mask)
        self.circuit.append(cirq.H.on_each(*target))

    def read(self, target_mask=~0, key=None):
        if key is None:
```

Q#

```
namespace QSharp.Chapter2
{
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Intrinsic;

    // Example 2-1: Random bit

    operation RandomBit () : Unit {
        // allocate one qubit
        use q = Qubit();
        // put it into superposition of 0 and 1
        H(q);

        // measure the qubit and store the result
        let bit = M(q);

        // make sure the qubit is back to the 0 state
        Reset(q);

        Message($"{bit}");
    }
}
```

+ Amazon Braket SDK

+ etc.

https://learn.microsoft.com/en-us/azure/quantum/install-overview-qdk

OpenQASM : le standard ?

https://github.com/openqasm/openqasm

OpenQASM 3.x Live Specification

- Introduction
  - Design Goals
  - Scope
  - Implementation Details
  - Contributors
- Language
  - Comments
  - Version string
  - Included files
  - Types and Casting
    - Identifiers
    - Variables
    - Quantum types
    - Classical scalar types
    - Compile-time constants
    - Literals
    - Arrays
    - Types related to timing
    - Aliasing
    - Index sets and slicing
    - Register concatenation and slicing
    - Classical value bit slicing
    - Array concatenation and slicing
    - Casting specifics
  - Gates
    - Built-in gates
    - Hierarchically defined unitary gates

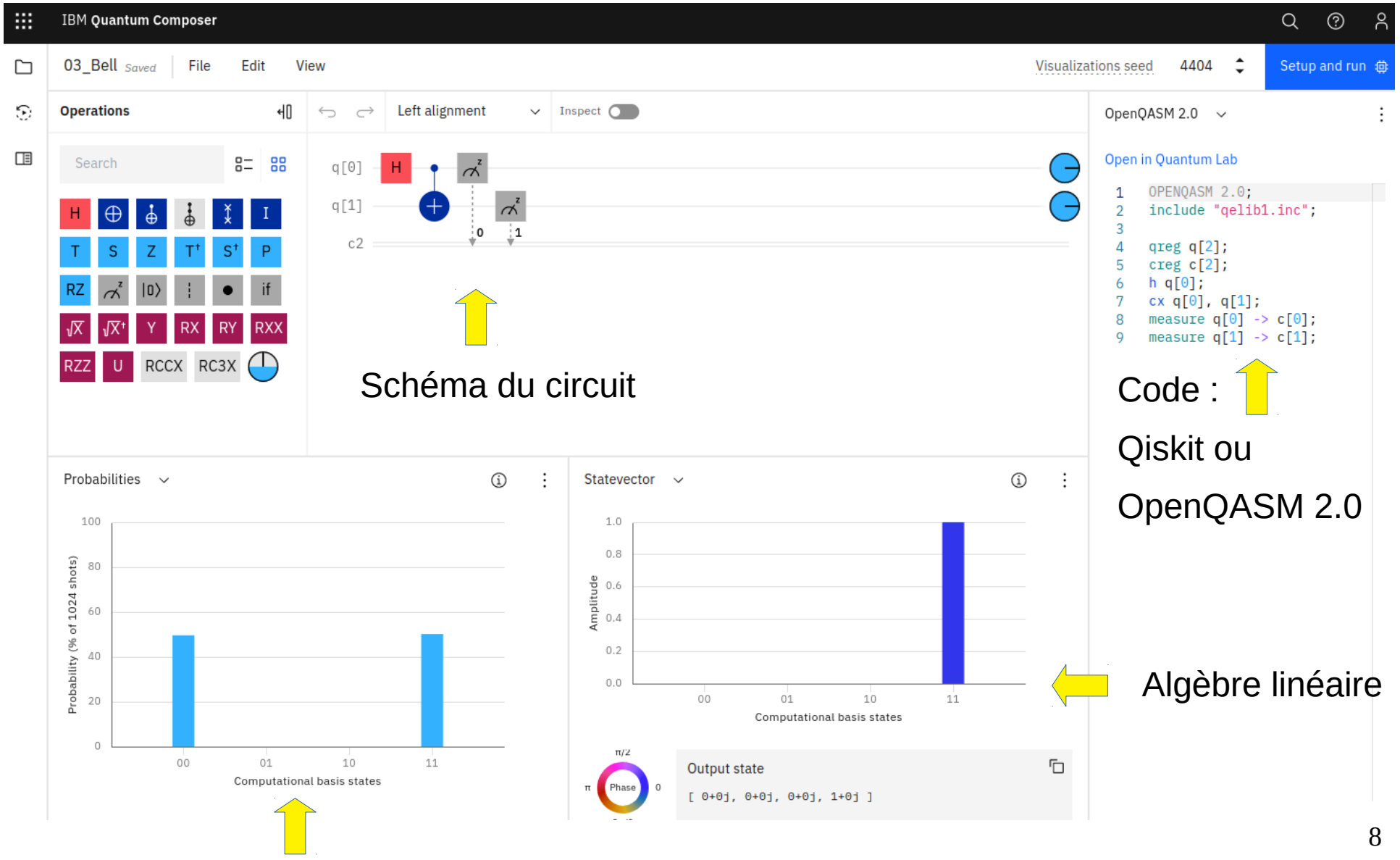
https://openqasm.com/

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

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

h q[0];
measure q[0] -> c[0];
```

Un langage de type IR (Intermediate Representation)

- déclaration / définition de portes
- contrôle d'exécution
- types classiques / quantiques
- procédures, fonctions externes
- programmation au niveau physique

Un programme est un circuit



IBM Quantum Experience

ibmq_quito

Details

5	Qubits	Status:	Online	Avg. CNOT Error:	1.053e-2
16	QV	Total pending jobs:	172 jobs	Avg. Readout Error:	4.374e-2
2.5K	CLOPS	Processor type ⓘ:	Falcon r4T	Avg. T1:	87.41 us
		Version:	1.1.34	Avg. T2:	82.22 us
		Basis gates:	CX, ID, RZ, SX, X	Providers with access:	1 Providers ↓
		Your usage:	4 jobs		

Your upcoming reservations 0

Calibration data Last calibrated: about 7 hours ago

Map view | Graph view | Table view

Qubit:

Frequency (GHz)

Avg 5.184

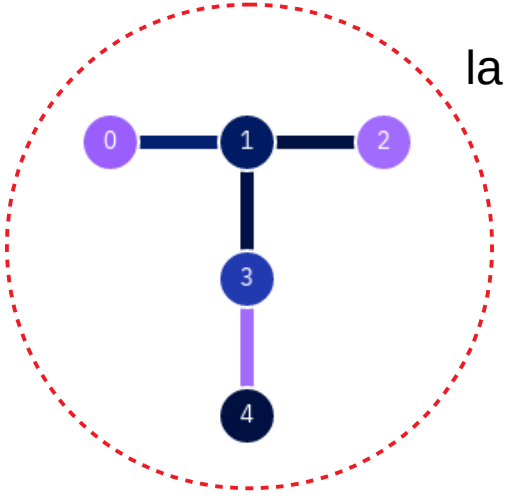
min 5.052 max 5.322

Connection:

CNOT error

Avg 1.053e-2

min 8.380e-3 max 1.581e-2



la topologie des registres de qubits (hardware), ici la configuration Falcon de IBM Quantum

Transpiler = compiler

(exemple avec le circuit pour créer l'état Bell)

$$CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad , \quad \text{fait partie des portes natives du processeur de type Falcon}$$

$$H = \frac{1}{\sqrt{(2)}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad , \quad \text{(porte d'Hadamard) ne fait pas partie ...}$$

doit être obtenu à partir des portes CX, ID, RZ, SX, X:

$$U = RZ(\pi/2) \cdot SX \cdot RZ(\pi/2)$$

$$RZ(\theta) = \exp(-i\,\theta/2\,Z) \quad , \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad , \quad SX = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} \quad ,$$

$$RZ(\pi/2) = \begin{bmatrix} \exp(-i\,\pi/4) & 0 \\ 0 & \exp(+i\,\pi/4) \end{bmatrix} \Rightarrow U = \exp(-i\,\pi/4) \cdot H$$

Amazon Braket (sur AWS)

Devices (12)

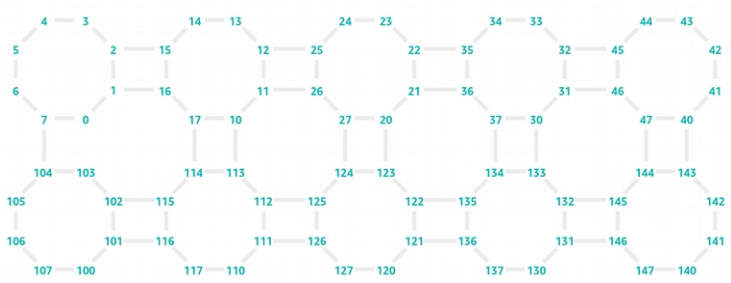
View device

Filter devices

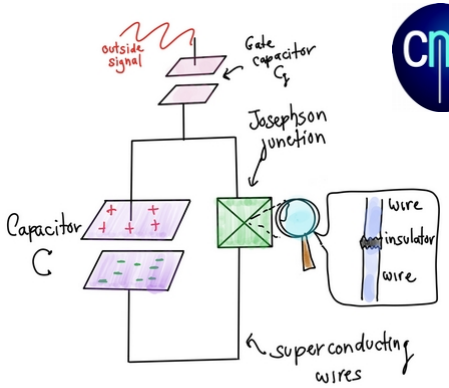
Show retired devices

Hardware provider	Device	Availability	Description
Amazon Web Services	SV1	AVAILABLE NOW	Amazon Braket state vector simulator
Amazon Web Services	TN1	AVAILABLE NOW	Amazon Braket tensor network simulator
Amazon Web Services	DM1	AVAILABLE NOW	Amazon Braket density matrix simulator
D-Wave	Advantage_system6.1	AVAILABLE NOW	Quantum Annealer based on superconducting qubits
D-Wave	Advantage_system4.1	AVAILABLE NOW	Quantum Annealer based on superconducting qubits
D-Wave	DW_2000Q_6	AVAILABLE NOW	Quantum Annealer based on superconducting qubits
IonQ	IonQ Device	AVAILABLE NOW	Universal gate-model QPU based on trapped ions
Oxford Quantum Circuits	Lucy	13:38:27	Universal gate-model QPU based on superconducting qubits
QuEra	Aquila	19:38:26	Analog quantum processor based on neutral atom arrays
Rigetti	Aspen-11	OFFLINE	Universal gate-model QPU based on superconducting qubits
Rigetti	Aspen-M-2	07:38:27	Universal gate-model QPU based on superconducting qubits
Xanadu	Borealis	18:38:26	Gaussian Boson Sampling on a programmable photonic processor

- supraconducteur, "discrète" : Rigetti, (80q, programmation avec portes), OQC
- supraconducteur, "analogue" : D-Wave (2048, 5760q, quantum annealing = recuit, variation continue)
- atomes neutres, "analogue" : QuEra (256q, 87Rb, lasers pour refroidir et manipuler)
- photons, "discrète" : Xanadu (216 modes)
- ions piégés, "discrète" : IonQ (11q, 171Yt+)



x80



Circuit with a Josephson junction and a gate capacitor

https://pennylane.ai

Rigetti Aspen-M-2

```
1 device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-2")
2 supported_gates = device.properties.action['braket.ir.jaqcd.program'].supportedOperatio
3 print('Gate set supported by the Rigetti device:\n', supported_gates)
4 print('\n')
```

```
Gate set supported by the Rigetti device:
 ['cz', 'xy', 'ccnot', 'cnot', 'cphaseshift', 'cphaseshift00', 'cphaseshift01', 'cphaseshift10', 'cswap', 'h', 'i',
'iswap', 'phaseshift', 'pswap', 'rx', 'ry', 'rz', 's', 'si', 'swap', 't', 'ti', 'x', 'y', 'z', 'start_verbatim_box',
'end_verbatim_box']
```

Calibration

Qubit specs | Edge specs | JSON

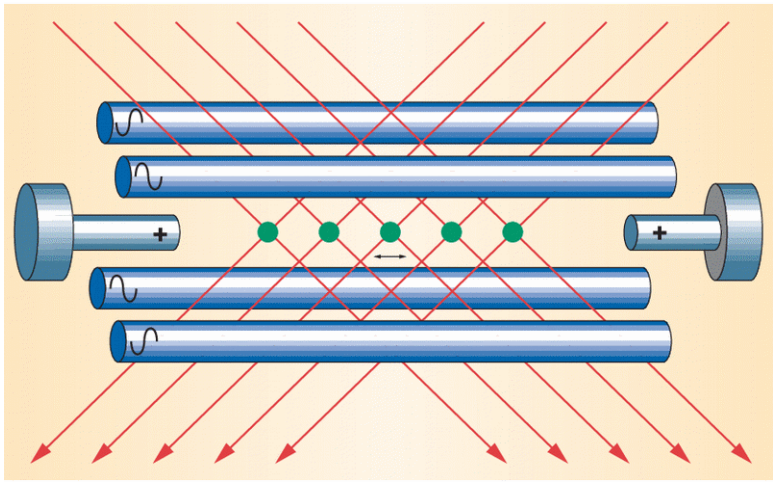
Last updated: Oct 04, 2022 15:01 (UTC) Info

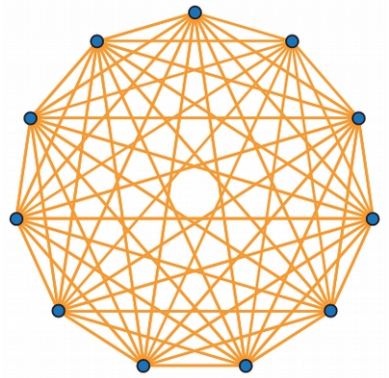
Qubit

Qubit	T1 (µs) Info	T2 (µs) Info	Fidelity (RB) (%) Info	Fidelity (simultaneous RB) (%) Info	Readout fidelity (%) Info	Active reset fidelity (%) Info
0	4.174	26.639	97.489 ± 0.363	98.366 ± 0.177	87.400	92.650
1	40.553	11.582	99.697 ± 0.018	99.599 ± 0.027	82.700	97.700
2	28.995	18.911	99.899 ± 0.007	99.792 ± 0.006	98.200	99.400
3	62.428	49.743	99.916 ± 0.020	99.855 ± 0.020	98.700	98.100
4	23.797	38.484	99.916 ± 0.011	99.780 ± 0.010	98.800	99.800
5	30.658	22.566	99.734 ± 0.034	99.788 ± 0.127	96.100	97.800
6	39.530	6.143	99.859 ± 0.027	99.684 ± 0.011	95.200	98.600
7	56.242	113.004	99.909 ± 0.011	99.854 ± 0.019	93.300	99.700

La technologie des ions piégés

Beyond Bits: The Future of Quantum Information Processing.
Andrew Steane, Eleanor G. Rieffel





11 qubits

IonQ

Calibration

Last updated: Oct 04, 2022 11:00 (UTC)


```
{
  "braketSchemaHeader": {
    "name": "braket.device_schema.ionq.ionq_provider_properties",
    "version": "1"
  },
  "fidelity": {
    "1Q": {
      "mean": 0.9954
    },
    "2Q": {
      "mean": 0.992
    },
    "spam": {
      "mean": 0.99752
    }
  },
  "timing": {
    "T1": 10000,
    "T2": 0.2,
    "1Q": 0.00001,
    "2Q": 0.0002,
    "readout": 0.00013,
    "reset": 0.00002
  }
}
```


```
device = AwsDevice("arn:aws:braket:::device/qpu/ionq/ionQdevice")
supported_gates = device.properties.action['braket.ir.jaqcd.program'].supportedOperations
print('Gate set supported by the IonQ device:\n', supported_gates)
print('\n')
```

```
Gate set supported by the IonQ device:
 ['x', 'y', 'z', 'rx', 'ry', 'rz', 'h', 'cnot', 's', 'si', 't', 'ti', 'v', 'vi', 'xx', 'yy', 'zz', 'swap', 'i']
```

Calcul mathématique versus représentation sous forme de qubits + série de circuits (évolution)

https://arxiv.org/abs/2104.08181 (Ruiz Guzman, Lacroix 2021)

are labeled as $i = 0, 1, \ldots, M-1$. This Hamiltonian is written as $H = H_J + H_U$, where H_J and H_U are the hopping and interaction terms respectively given by:

$$H_J = -J \sum_{i,\sigma} (a^\dagger_{i+1,\sigma} a_{i,\sigma} + a^\dagger_{i,\sigma} a_{i+1,\sigma}),$$
$$H_U = +U \sum_i \hat{n}_{i,\uparrow} \hat{n}_{i,\downarrow},$$

with $n_{i,\sigma} = a^\dagger_{i,\sigma} a_{i,\sigma}$ and $\sigma = \{\uparrow, \downarrow\}$. In order to apply the JWT mapping, it is convenient to organize the qubits as follows. Spin-up single-particle states indexed as $i = 0, \ldots, M-1$ are associated with qubits labeled with $\alpha = 0, \ldots, M-1$. Particles with spin-down indexed as $i = 0, \ldots, M-1$ are associated to qubits $\alpha = M, \ldots, 2M-1$. With this, we obtain the mapping (with proper account for the boundary conditions):

$$H_J = J \sum_{\alpha=0, \alpha \neq M-1}^{2M-2} \left[Q^+_{\alpha+1} Q_\alpha + \text{h.c.} \right],$$

together with

$$H_U = \frac{U}{4} \sum_{\alpha=0, M-1} [I_\alpha - Z_\alpha] [I_{\alpha+M} - Z_{\alpha+M}]. \quad (6)$$

The generating function evaluation with the circuits presented in Fig. 1 requires to perform the time-evolution operator. For its implementation, we simply use the Trotter-Suzuki method [25, 49]. The time interval $[0, t]$ is divided into small intervals Δt. For small enough time interval, we have:

$$U(\Delta t) = e^{-i\Delta t H} \simeq e^{-i\Delta t H_J} e^{-i\Delta t H_U} \equiv U_J(\Delta t) U_U(\Delta t).$$

The propagators U_J can be further decomposed as:

$$U_J(\Delta t) = \prod_\alpha e^{-iJ\Delta t \left[Q^+_{\alpha+1} Q_\alpha + \text{h.c.} \right]},$$
$$= \prod_\alpha \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\lambda) & -i\sin(\lambda) & 0 \\ 0 & -i\sin(\lambda) & \cos(\lambda) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}_{\alpha, \alpha+1} \quad (7)$$

with $\lambda = \Delta t J$. To obtain the matrix form, standard manipulation of Pauli matrices is used. Note that the index on the matrix indicates that the matrix acts on the two qubits α and $\alpha + 1$.

For the interaction propagator we have

$$U_U(\Delta t) = \prod_\alpha e^{-iU\Delta t [I_\alpha - Z_\alpha][I_{\alpha+M} - Z_{\alpha+M}]},$$
$$= \prod_\alpha \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-i\Delta t U} \end{pmatrix}_{\alpha, \alpha+M}. \quad (8)$$

We recognize in the last expression the controlled phase-shift gate with phase $\phi = -\Delta t U$. The two circuits that simulate U_U and U_J are displayed in panels (a) and (b) of Fig. 2.

2. *Pairing Hamiltonian*

As a second illustration, we will also consider the pairing Hamiltonian [50–53] that is standardly used in the context of nuclear physics or small superconducting systems. This Hamiltonian has already been used on QC in Refs. [37, 38] and more recently in Refs. [31, 54]. We write this Hamiltonian as:

$$H = \sum_p \varepsilon_p N_p + g \sum_{pq} P^\dagger_p P_q \equiv H_\varepsilon + H_g. \quad (9)$$

Introducing the notation $(a^\dagger_p, a^\dagger_{\bar{p}})$ as the creation operators of time-reversed single-particle states. The different operators are defined as:

$$\hat{N}_p = a^\dagger_p a_p + a^\dagger_{\bar{p}} a_{\bar{p}},$$
$$\hat{P}^\dagger_p = a^\dagger_p a^\dagger_{\bar{p}}.$$

These operators correspond respectively to the pair occupation, and to the pair creation operators. In this model, time-reversed single-particle states are degenerated with

Problème de physique

↓

formulation

↓

modèle de calcul

↓

programmation

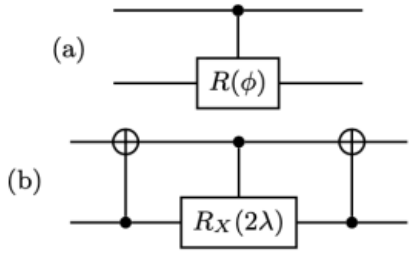
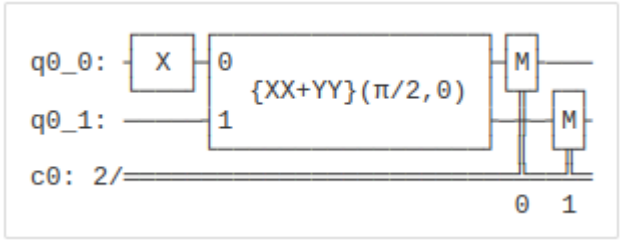


FIG. 2: Circuits used to simulate the Hubbard model. The circuit (a) simulates the interaction term H_U where $R(\phi)$ is the unitary phase operator with $\phi = -\Delta t U$. Circuit (b) simulates a short time-step evolution of the hopping term H_J where $R_X(2\lambda) = e^{-i\lambda X}$ and where $\lambda = J\Delta t$.

Simulateurs / émulateurs

Qiskit

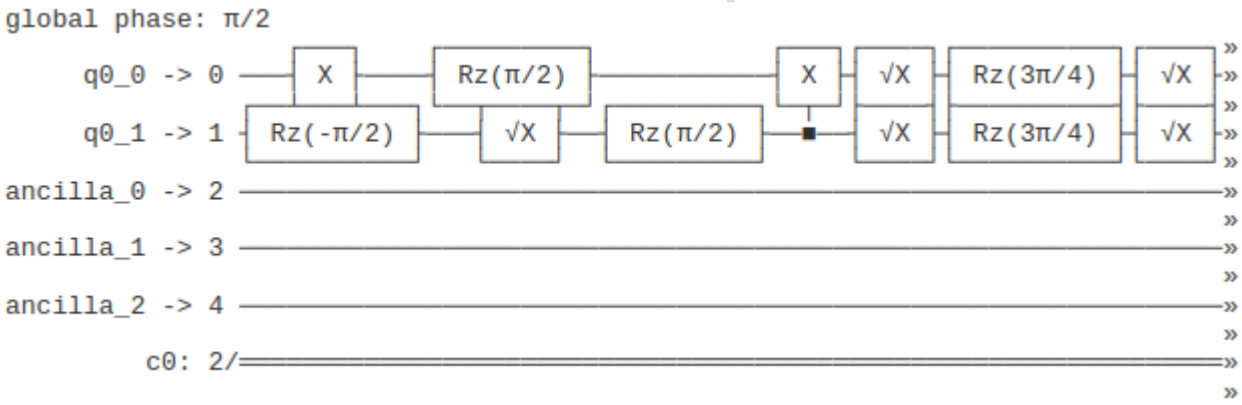


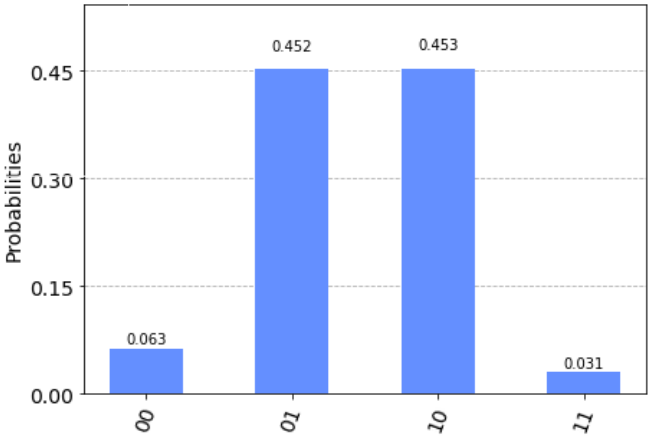
Pourquoi simuler ?

- pour comprendre la modélisation et l'optimiser
- pour adapter à une topologie spécifique de QPU
- pour essayer de mitiger les effets du bruit quantique

```
from qiskit.circuit.library import XXPlusYYGate
qr = QuantumRegister(2)
cr = ClassicalRegister(2)
qc = QuantumCircuit(qr, cr)
qc.x(qr[0])
theta = pi/2
XY = XXPlusYYGate(theta, beta)
qc.append(XY, qr)
qc.measure(qr, cr)
qc.draw()
```

```
from qiskit.providers.aer import AerSimulator
from qiskit.providers.fake_provider import FakeQuitoV2
backend = FakeQuitoV2()
backend_sim = AerSimulator.from_backend(backend)
transpiled_circuit = transpile(qc, backend_sim)
job = backend_sim.run(transpiled_circuit)
counts = job.result().get_counts()
plot_histogram(counts)
```





Simuler la théorie (pas seulement la mesure)

Amazon Braket

```
from braket.circuits import Circuit, Observable
from braket.devices import LocalSimulator
circ = Circuit().rx(0, 0.15).ry(1, 0.2).rz(2, 0.25).h(3).cnot(control=0,
       target=2).cnot(1, 3).x(4)
obs = Observable.X() @ Observable.Y()
target_qubits = [0, 1]
circ.expectation(obs, target=target_qubi
circ.variance(obs, target=target_qubits)
circ.sample(obs, target=target_qubits)
device = LocalSimulator()
task = device.run(circ, shots=100)
result = task.result()
print("Expectation value for <X0*Y1>:", result.values[0])
...
circ.state_vector()
task = device.run(circ, shots=0)
result = task.result()
print("Final state vector:\n",
result.values[0])
```

```
T  : |   0    | 1 |              Result Types                |

q0 : -Rx(0.15)-C---Expectation(X@Y)-Variance(X@Y)-Sample(X@Y)-
                |   |                |              |
q1 : -Ry(0.20)-|-C-Expectation(X@Y)-Variance(X@Y)-Sample(X@Y)-
                | |
q2 : -Rz(0.25)-X-|-----------------------------------------
                  |
q3 : -H----------X-----------------------------------------

q4 : -X-----------------------------------------------------

T  : |   0    | 1 |              Result Types                |
```

$(2^5 = 32$ nombres complexes$)$

```
Final state vector:
 [-0.41050209-0.56896874j  0.        +0.j         -0.22402587-0.66486826j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j         -0.02902509-0.06413206j
  0.        +0.j         -0.00877641-0.0698452j   0.        +0.j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j         -0.00457713+0.00265117j
  0.        +0.j         -0.00515618+0.00118012j  0.        +0.j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.        +0.j         -0.04995883+0.01683351j  0.        +0.j
 -0.05270213+0.00131783j  0.        +0.j        ]
```

Hybride : CPU + QPU (tasks et jobs)

Amazon Braket

Script Python : `script.py`

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.jobs import save_job_result
device = AwsDevice(os.environ["DEVICE_ARN"])
counts_list = []
angle_list = []
for _ in range(5):
    angle = np.pi * np.random.randn()
    random_circuit = Circuit().rx(0, angle)
    task = device.run(random_circuit, shots=100)
    counts = task.result().measurement_counts
    angle_list.append(angle)
    counts_list.append(counts)
    print(counts)
save_job_result({"counts": counts_list, "angle": angle_list})
print("Test job completed!")
```

```
from braket.aws import AwsQuantumJob
job = AwsQuantumJob.create(device=
"arn:aws:braket:::device/quantum-
simulator/amazon/sv1", source_module=
"script.py", entry_point= "",
wait_until_complete= True)
```

synchrone ou asynchrone

```
results = job.result()
print("counts: ", results["counts"])
print("angles: ", results["angles"])
job.download_result()
```

S'exécute à l'intérieur d'un container Docker dans le Cloud de Calcul Élastique Amazon (EC2).

Conclusions

La programmation des processeurs quantiques aujourd'hui :

- des outils pour décrire la nature quantique des *éléments de calcul*
- des outils pour simuler (algèbre linéaire, statistique quantique)
- des commandes pour manipuler expérimentalement les qubits
- des élément *classiques* de programmation
- dépendent de l'interface vers le QPU (le fabricant) : Qiskit, Cirq, Q#, ...
- OpenQASM se profile comme un langage intermédiaire (IR) sur deux niveaux : logique et physique (voir poster)
  - initialement par IBM, récemment adopté par Amazon Braket pour les QPU mis à disposition par ses partenaires: Rigetti, IonQ, OQC
  - IBM OpenQASM 2.0 (bientôt 3.0), Amazon Braket OpenQASM 3.0

- ceci n'est pas une revue des langages existant aujourd'hui...

Merci de votre attention !

Connie Zhou for IBM (dans QuantaMagazine)