OpenQASM[1,2,3] : vers un standard pour la programmation des processeurs quantiques

Bogdan VULPESCU (Laboratoire de Physique de Clermont)

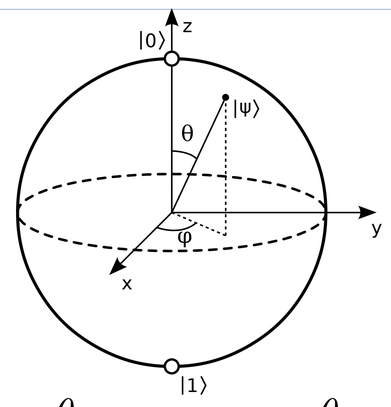
Projet QC2I
https://qc.pages.in2p3.fr/web/

Un qubit est représenté par son vecteur d'état, une combinaison linéaire (une superposition) des deux vecteurs de la base, à l'aide de deux coefficients complexes qui respectent une relation de normalisation. Les vecteurs appartiennent à un espace vectoriel à 2 dimensions muni d'un produit scalaire, un espace Hilbert. La sphère de Bloch permet la visualisation du qubit sur la surface d'une sphère unité, avec les états correspondant à la base canonique situés aux pôles de la sphère.
Les transformations du qubit sont décrites par des opérateurs du groupe unitaire d'ordre 2 : $U(2) = \{A \in \mathrm{M}_2(\mathbb{C}) | A^\dagger A = I\}$

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle \;\; , \;\; a, b \in \mathbb{C}$$

$$|a|^2 + |b|^2 = 1 \quad \text{(amplitudes de probabilité)}$$

$$p_{meas}(|0\rangle) = |a|^2 \quad , \quad p_{meas}(|1\rangle) = |b|^2$$



La sphère de Bloch

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

$|0\rangle$ et $|1\rangle$ sont les vecteurs propres de l'opérateur (Pauli) Z

Tout élément du groupe U(2) peut être spécifié – jusqu'à une phase globale – en utilisant les trois angles Euler θ, ϕ, λ. Les rotations correspondent à des opérateurs définis à l'aide des opérateurs Pauli Y et Z. Une autre décomposition est possible dans la base formée par l'opérateur Hadamard et l'opérateur de changement de phase.

$$U(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\phi+\lambda)}\cos(\theta/2) \end{bmatrix}$$

$$R_z(\alpha) = \exp(-i\alpha/2Z) \;\; , \;\; Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \;\; , \;\; R_y(\beta) = \exp(-i\beta/2Y) \;\; , \;\; Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$U(\theta, \phi, \lambda) = e^{i(\phi+\lambda)/2} R_z(\phi) R_y(\theta) R_z(\lambda)$$

$$U(\theta, \phi, \lambda) = e^{i\theta/2} P(\phi + \pi/2) H P(\theta) H \, P(\lambda - \pi/2)$$

Déclaration OpenQASM de la porte Hadamard:

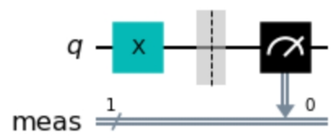
```
gate h q {U(π/2, 0, π) q;}
```

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \;\; , \;\; P(\alpha) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{bmatrix}$$

La porte d'inversion X:

```
gate x q {U(π, 0, π) q;}
```

$$\Rightarrow \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

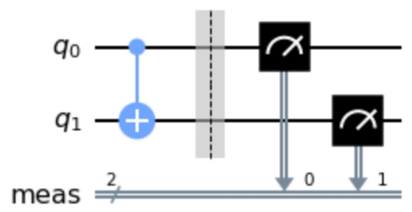


$\|q\rangle$	$meas$	%
$\|0\rangle$	1	100
$\|1\rangle$	0	100

La variante « contrôlée » CX de la porte X:

```
gate cx c, t {ctrl @ U(π, 0, π) c, t;}
cx q[0], q[1];
```

$$\Rightarrow \quad CX \equiv I \oplus X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$\|q_0\rangle$	$\|q_1\rangle$	$meas(q_1)$	%
$\|0\rangle$	$\|0\rangle$	0	100
$\|1\rangle$	$\|0\rangle$	1	100
$\|1\rangle$	$\|1\rangle$	0	100
$\|0\rangle$	$\|1\rangle$	1	100

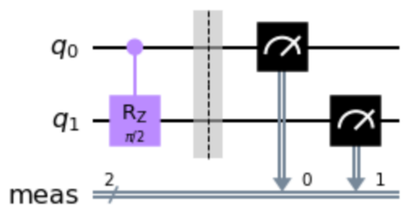
Pour les rotations, il est nécessaire d'inclure une phase globale :

```
gate rz(a) q {gphase(-a/2); U(0, 0, a) q;}
```

Pour appliquer la porte dans sa variante contrôlée:

```
ctrl @ rz(a) q[0], q[1];
```

$$\Rightarrow \quad I \oplus R_z(\pi/2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i\pi/4} & 0 \\ 0 & 0 & 0 & e^{i\pi/4} \end{bmatrix}$$



En OpenQASM il y a trois modificateurs de porte: contrôle (**`ctrl @ g`**), inversion (**`inv @ g`**), qui donne l'opérateur adjoint et puissance (**`pow(r) @ g`**).
À noter que pour chaque opérateur unitaire U il existe un opérateur H avec valeurs propres réelles (opérateur Hermitique) tel que: $U = \exp(iH), U^r = \exp(irH)$.

Le contrôle de l'exécution du code : **`if`** **`else`** **`for`** **`while`**

```
qreg q[4];
creg c[4];
reset q;
h q[0];
measure q[0] -> c[0];
if (c[0] == 1) p(-pi/2) q[1];
qubit in;
```

Types classiques: **`int, uint, int[n], float, bool, bit, angle, angle[n]`**

$$2\pi \times 0.c_{n-1}c_{n-2}\ldots c_0$$

avec des valeurs dans $[0, 2\pi)$

et deux modificateurs de variables :

```
input int basis;
output bit result;
```

Procédures:

```
def func(bit[2] a) -> bit[2] {
  if (...) return b"11"; ...
flags = func(a);
```

Fonctions externes:

```
extern func(bit a) -> bit;
```

Python, C, x86, compilé à l'extérieur (GCC, LLVM)

+ structure de contrôle minimal pour le *run-time*

La programmation pour le **niveau physique**, dépendent de plateforme.

Délai, durée:

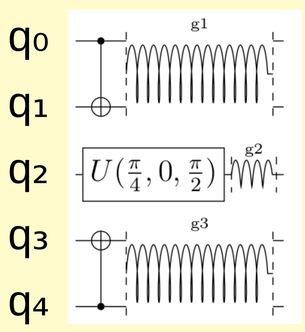
```
duration stride = 1us;
reset q[0];
x q[0];
delay[stride] q[0];
barrier q[0];
c0 = measure q[0];
```

Périmètre, barrière:

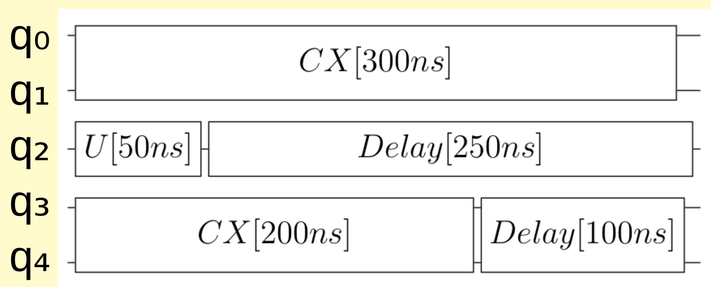
```
box {
  delay[ddt] q;
  x q;
  delay[ddt] q;
  x q;
  delay[ddt] q;
} // découplage dynamique
```

Alignement des portes:

```
// à gauche
stretch g1, g2, g3;
barrier q;
cx q[0], q[1];
```




```
delay[g1] q[0], q[1];
u(π/4, 0, π/2) q[2];
delay[g2] q[2];
cx q[4], q[3];
delay[g3] q[3], q[4];
barrier q;
```



[1] https://arxiv.org/abs/1707.03429, https://arxiv.org/abs/2104.14722
[2] https://openqasm.com/
[3] https://github.com/openqasm/openqasm