Informatique Quantique – Cours 3

Nicolas Boutry[1]

[1] Laboratoire de Recherche et Développement de l'EPITA (LRDE), France

Outline

1. Real Data
2. Quantum Search
3. Résoudre un problème logique par l'informatique quantique
4. Shor Factoring Algorithm
5. Quantum Machine Learning
   - Systèmes d'équations linéaires
   - Quantum Principal Component Analysis (QPCA)
   - Quantum Support Vector Machines (QSVM)
6. Annexes
   - Opérateurs simples
   - Décomposition en opérateurs simples
   - Algorithme de Deutsch-Josza
   - Algorithme de Simon
7. Conclusion

Outline

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning
- Systèmes d'équations linéaires
- Quantum Principal Component Analysis (QPCA)
- Quantum Support Vector Machines (QSVM)

6 Annexes
- Opérateurs simples
- Décomposition en opérateurs simples
- Algorithme de Deutsch-Josza
- Algorithme de Simon

7 Conclusion

Rappelons le fonctionnement de la RAM classique :

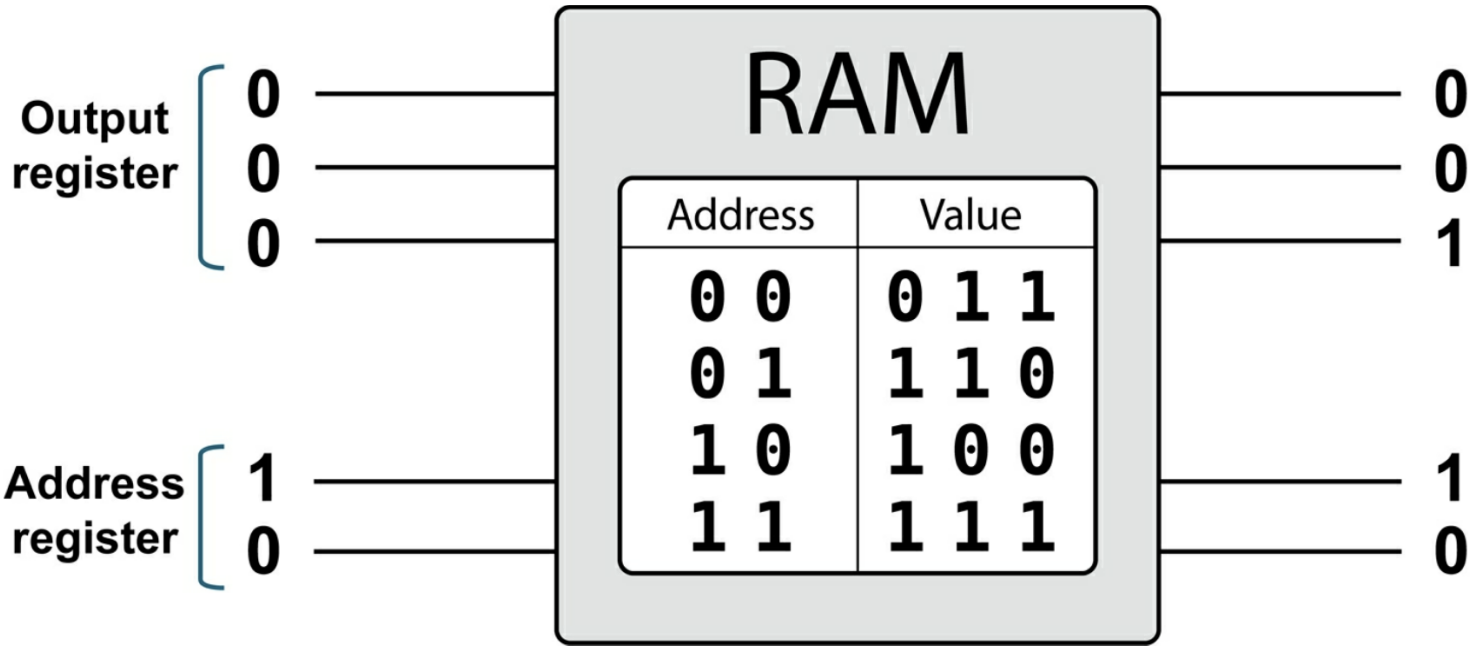


Figure 9-2. Conventional RAM — the table shows the values stored along with the interface used to access them

Et comparons cela à de la QRAM qui supporte la superposition d'états :

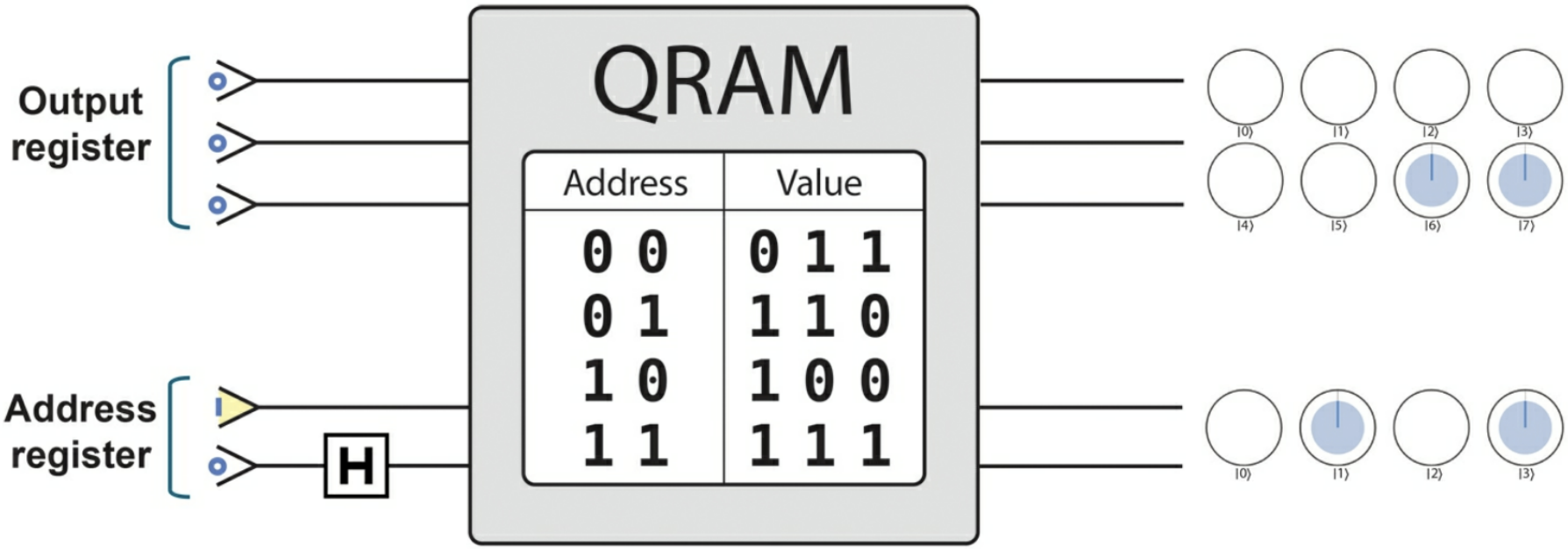


Figure 9-4. QRAM — using a HAD operation we prepare our address register in superposition, and we receive a similar superposition of stored values (shown in circle notation)

Ici, on demande d'écrire ce qu'il y a aux adresses 01 et 11 en même temps, on a donc en sortie 110 et 111, c'est-à-dire $|6\rangle$ et $|7\rangle$.

Imaginons maintenant que l'on veuille coder un vecteur $v = [0, 1, 2, 3]^T$, on peut coder ces informations au niveau des états:

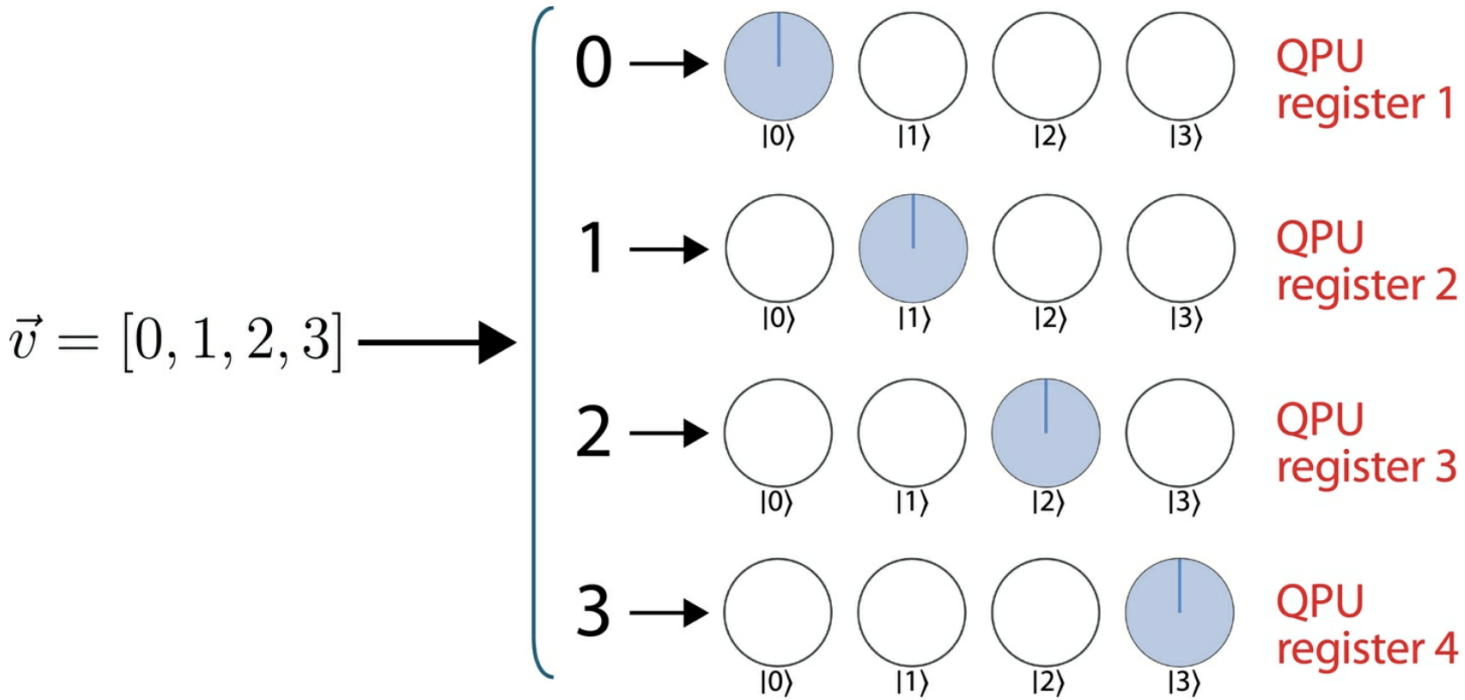


Figure 9-6. Using a state encoding for storing vector data in QPU registers

On peut aussi coder le vecteur au niveau des amplitudes :

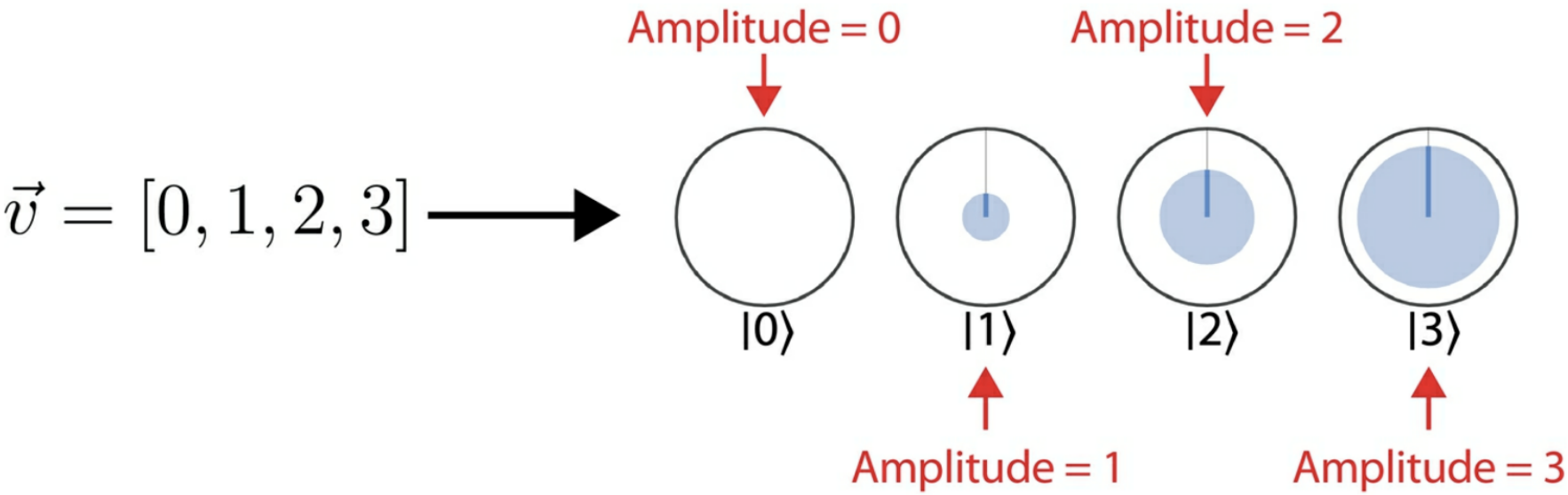
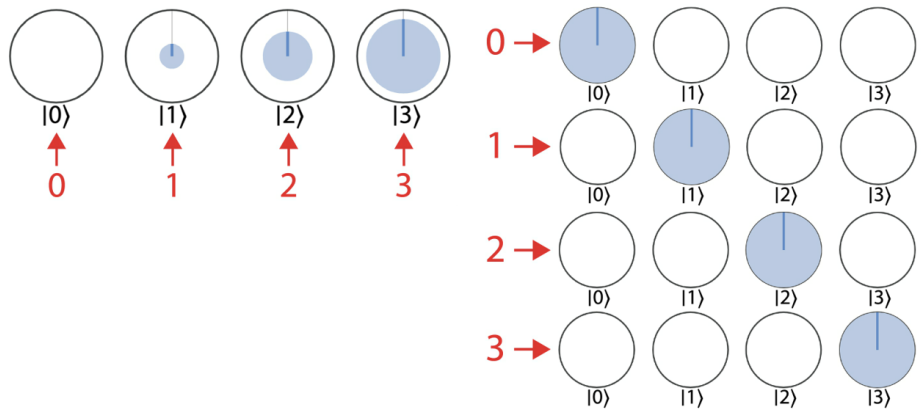


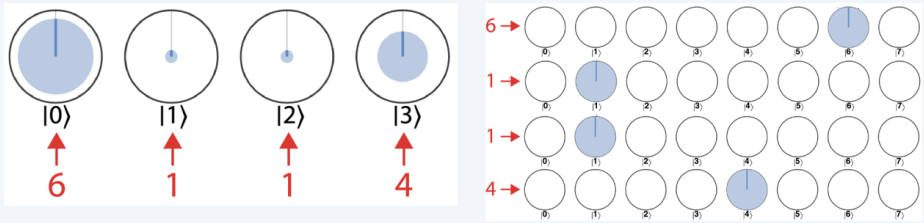
Figure 9-7. The basic idea of amplitude encoding for vectors

On pourra les restituer aisément ainsi :

```
[0, 1, 2, 3]
```




```
[6, 1, 1, 4]
```



Bien entendu, on encodera alors en relatif car les amplitudes sont des racines carrées de probabilités :

$$\vec{v} = [0, 1, 2, 3]$$

$$\longrightarrow [0.00, 0.27, 0.53, 0.80]$$

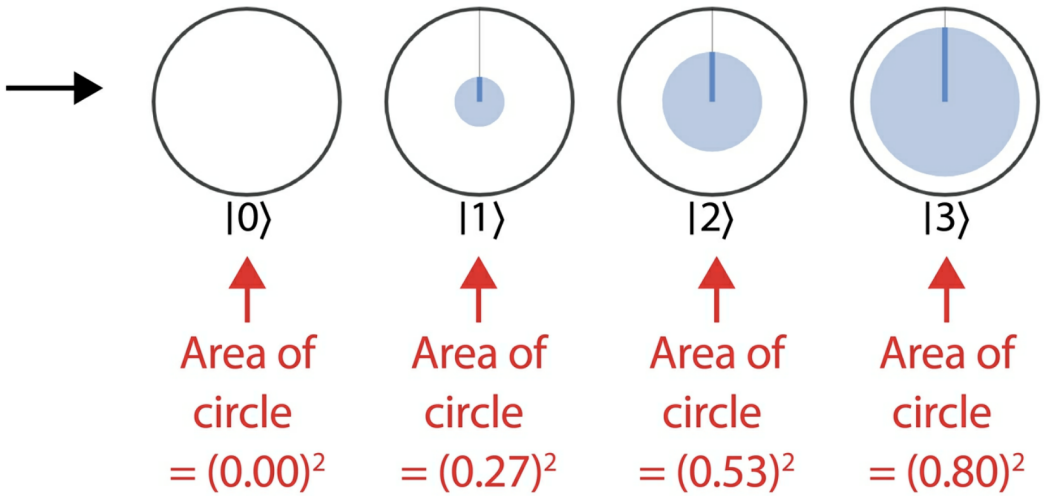


Figure 9-8. Correct amplitude encoding with a properly normalized vector

Outline

1. Real Data
2. Quantum Search
3. Résoudre un problème logique par l'informatique quantique
4. Shor Factoring Algorithm
5. Quantum Machine Learning
   - Systèmes d'équations linéaires
   - Quantum Principal Component Analysis (QPCA)
   - Quantum Support Vector Machines (QSVM)
6. Annexes
   - Opérateurs simples
   - Décomposition en opérateurs simples
   - Algorithme de Deutsch-Josza
   - Algorithme de Simon
7. Conclusion

TIP

The difference between binary logic and phase logic is important to understand and a potential point of confusion. Here's a summary:

Conventional binary logic

Applies logic gates to an input, producing an output

Quantum magnitude logic

Applies logic gates to a *superposition* of inputs, producing a *superposition* of outputs

Quantum phase logic

Flips the phase of every input value that would produce a 1 as output; this works when the register is in superposition too

It's perhaps easier to grasp the action of phase logic by seeing some examples in circle notation. Figure 10-1 illustrates how phase-logic versions of OR, NOR, XOR, AND, and NAND operations affect a uniform superposition of a two-qubit register.

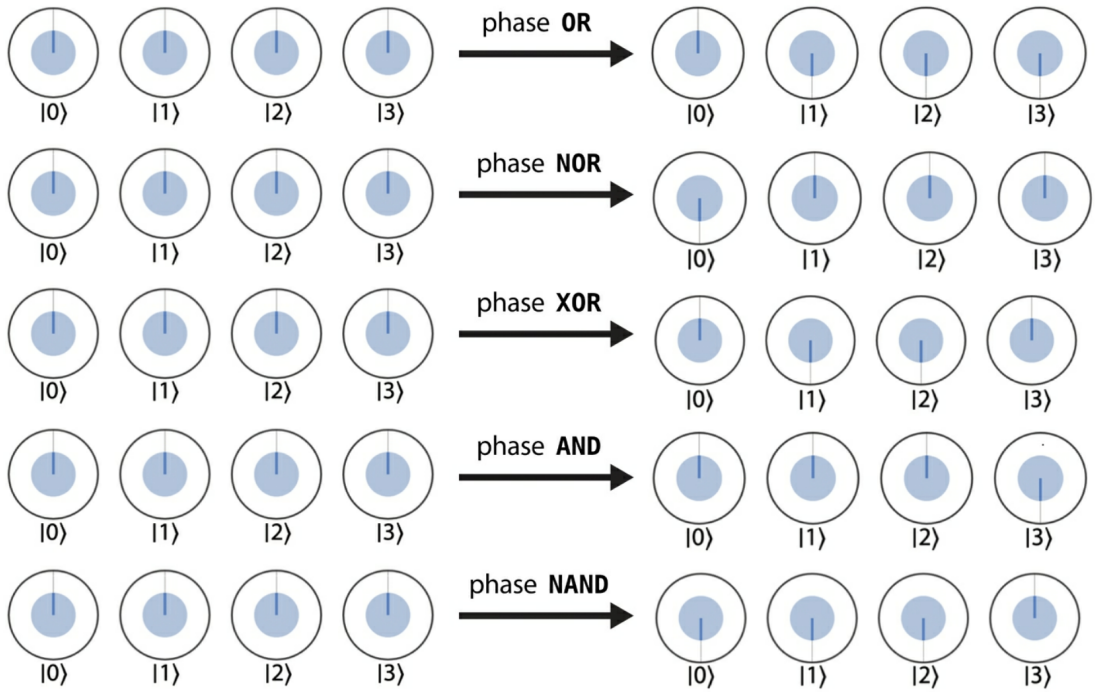


Figure 10-1. Phase-logic representations of some elementary logic gates on a two-qubit register

Nommons *a* le premier qubit, et *b* le deuxième qubit. On observe donc que les états $|1\rangle$, $|2\rangle$ et $|3\rangle$ sont retournés car *a* OR *b* est à 1 pour tous les états correspondant à "Soit *a* soit *b* est à 1".

Voici alors quelques portes quantiques de phase :

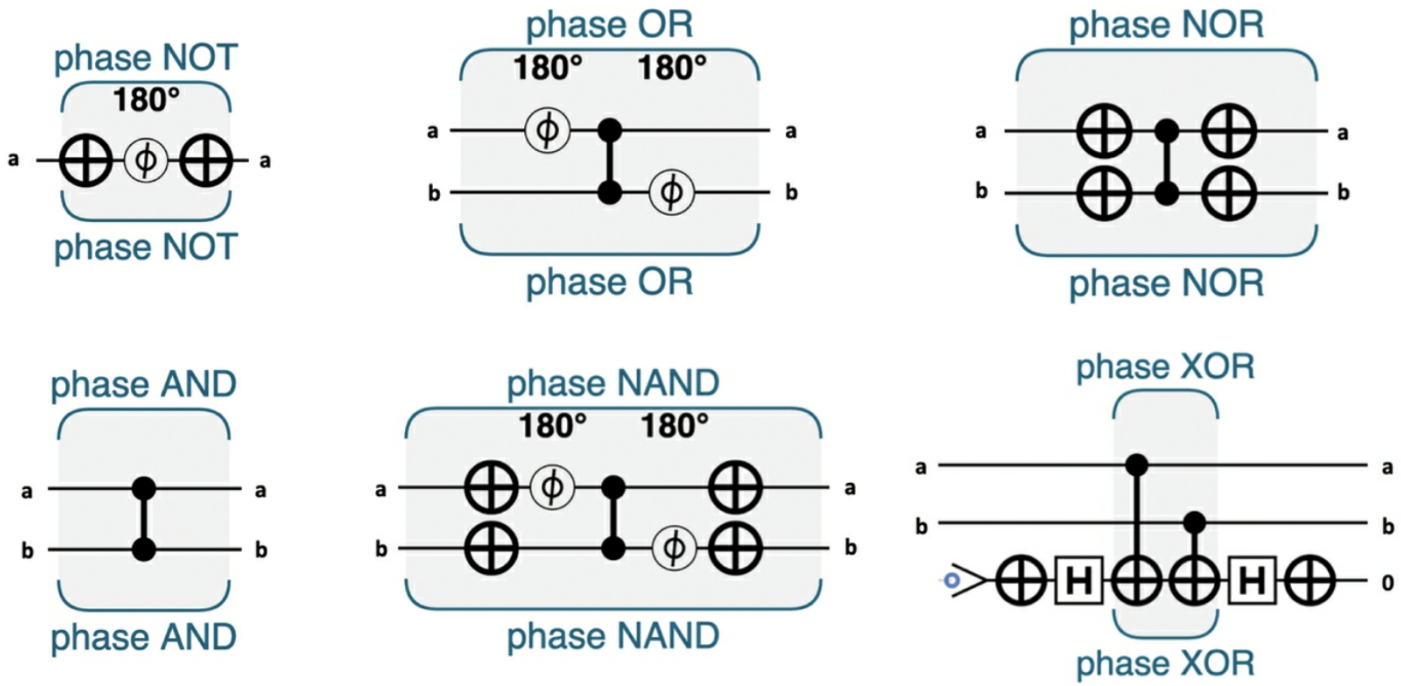


Figure 10-2. QPU operations implementing phase logic

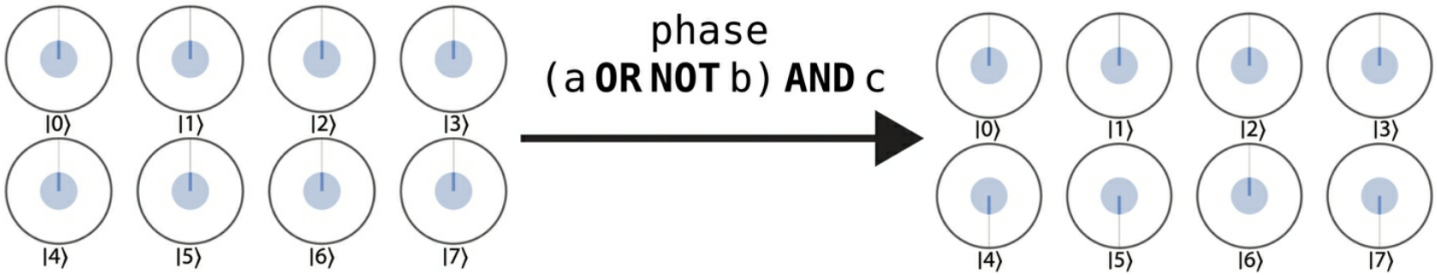


Figure 10-5. State transformation

Exercice : retrouvez que les états qui vont être retournés sont bien les états $|4\rangle$, $|5\rangle$ et $|7\rangle$ (indice : *a* qubit de poids faible et *c* qubit de poids fort).

Outline

1. Real Data
2. Quantum Search
3. Résoudre un problème logique par l'informatique quantique
4. Shor Factoring Algorithm
5. Quantum Machine Learning
   - Systèmes d'équations linéaires
   - Quantum Principal Component Analysis (QPCA)
   - Quantum Support Vector Machines (QSVM)
6. Annexes
   - Opérateurs simples
   - Décomposition en opérateurs simples
   - Algorithme de Deutsch-Josza
   - Algorithme de Simon
7. Conclusion

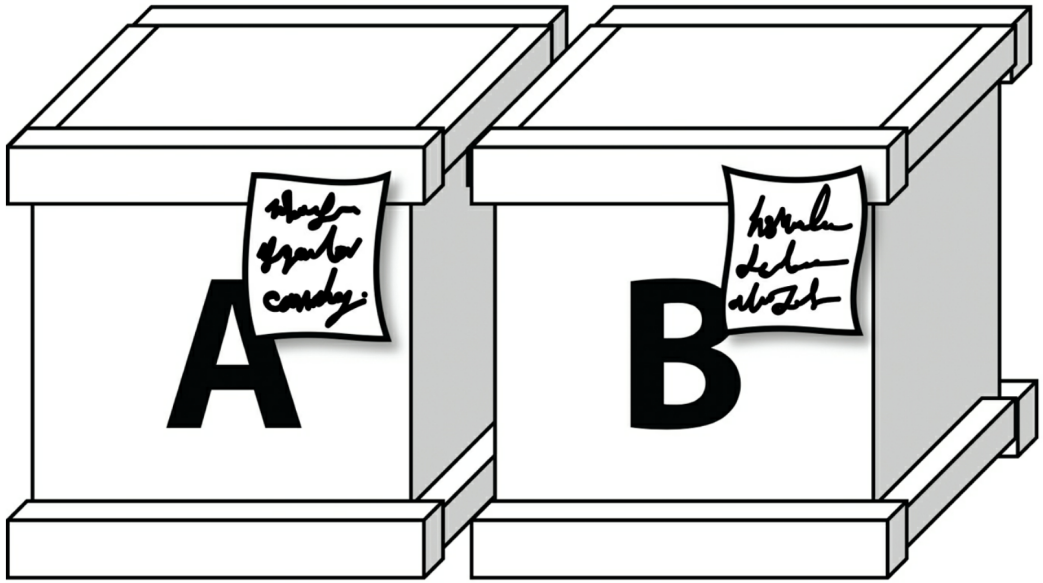


Figure 10-6. A birthday puzzle

On her big day, the princess received two boxes, and was allowed to open at most one. Each box might contain her coveted kitten, but might also contain a vicious tiger. Fortunately, the boxes came labeled as follows:

Label on box A

At least one of these boxes contains a kitten.

Label on box B

The other box contains a tiger.

De plus, on apprend que la règle du jeu est qu'il faut que soit A et B soient vraies, soit A et B soient fausses, trouver quelle boîte il faut ouvrir pour espérer avoir le chat.

Voilà comment résoudre à coup de schéma logique si cette solution peut être atteinte ou vérifiable :

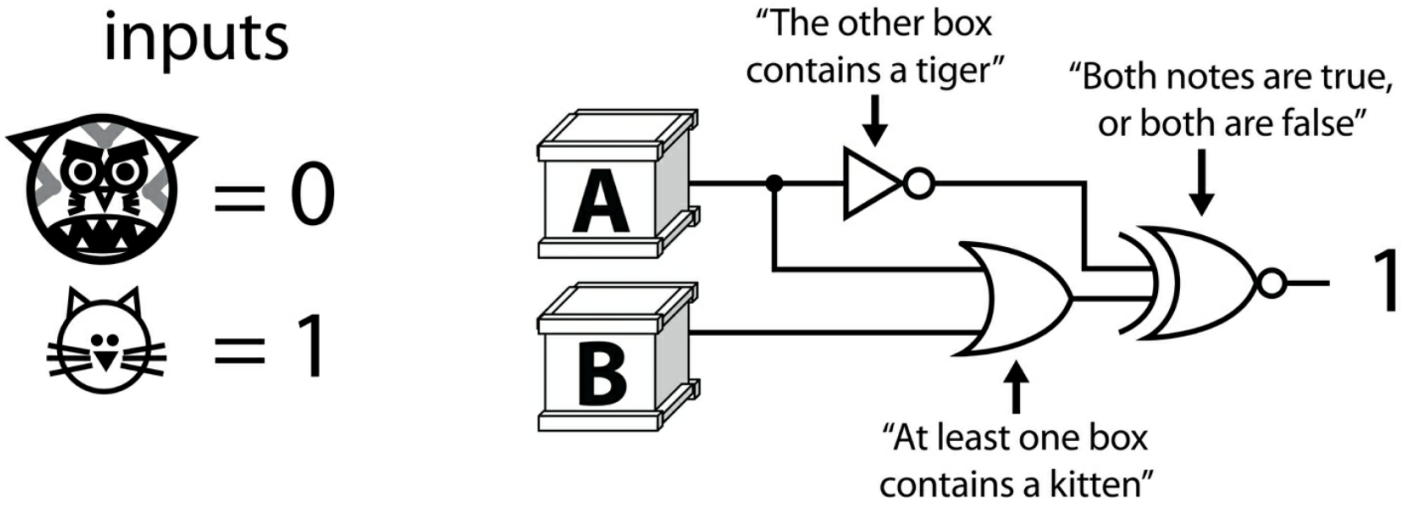


Figure 10-7. A digital solution to the problem

Une solution possible est donc un tigre dans A et un chat dans B.

Par l'informatique quantique, on peut tester toutes les solutions possibles en même temps grâce à la superposition d'états :

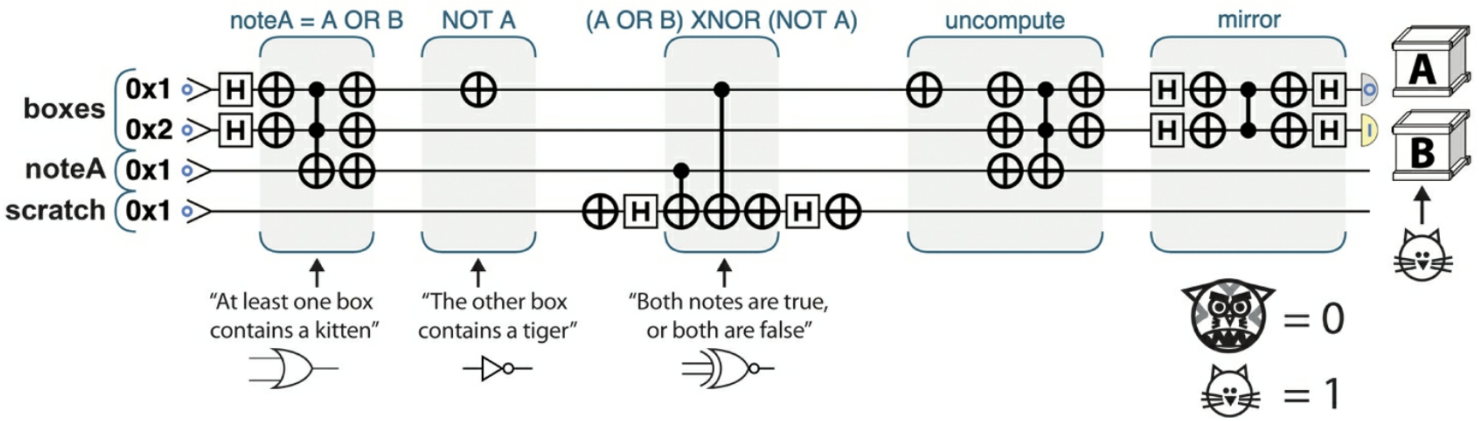


Figure 10-8. Logic puzzle: Kitten or tiger?

Sachant que les phases retournées seront celles correspondant aux bonnes solutions, à quoi sert le mirror ? (Réponse au slide suivant).

Les phases retournées seront transformées en “plus grandes amplitudes”, ainsi elles seront lisibles en sortie de l'algorithme dans la distribution de probabilités.

Imaginons un problème de satisfaisabilité (3-SAT) plus complexe que précédemment :

Hands-on: A Satisfiable 3-SAT Problem

Consider the following 3-SAT problem:

```
(a OR b) AND (NOT a OR c) AND (NOT b OR NOT c) AND (a OR c)
```

On aura alors besoin du schéma quantique suivant pour savoir s'il est vérifiable :

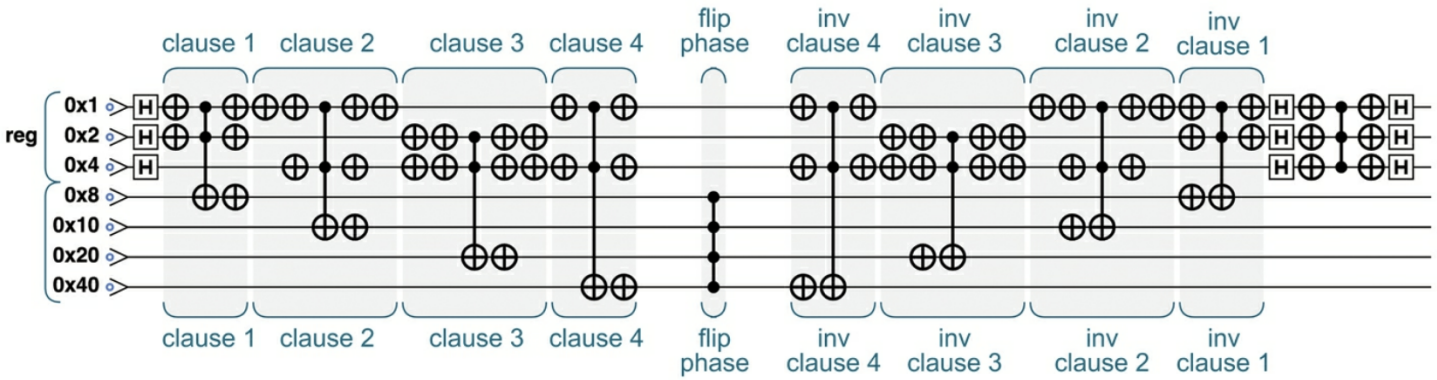


Figure 10-9. A satisfiable 3-SAT problem

En règle générale, ce type de problème sera de la forme suivante :

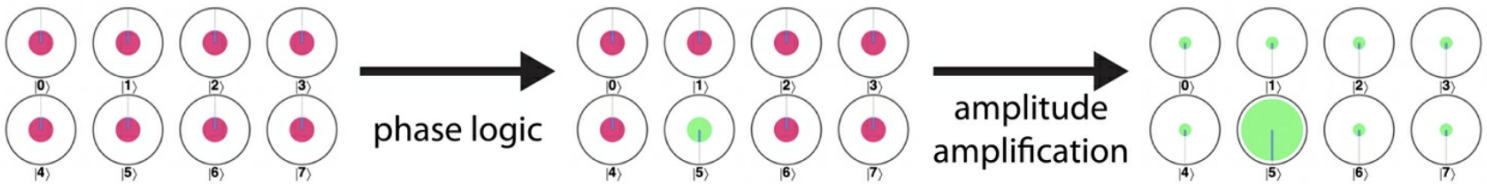


Figure 10-10. Circle notation for a satisfiable 3-SAT problem after one iteration

This tells us that the boolean statement can be satisfied by values `a=1`, `b=0`, and `c=1`.

On part des états en superposition, on applique notre schéma quantique pour voir la vérifiabilité du problème posé (avec une logique de phase), on a donc retourné certaines phases, puis on fait de l'AA pour obtenir les sorties correspondant à des solutions possibles avec des probabilités correspondantes non nulles.

Imaginons un problème dont on sait qu'il n'admet pas de solution, que se passera-t-il ?

Hands-on: An Unsatisfiable 3-SAT Problem

Now let's look at an unsatisfiable 3-SAT problem:

```
(a OR b) AND (NOT a OR c) AND (NOT b OR NOT c) AND (a OR c) AND b
```

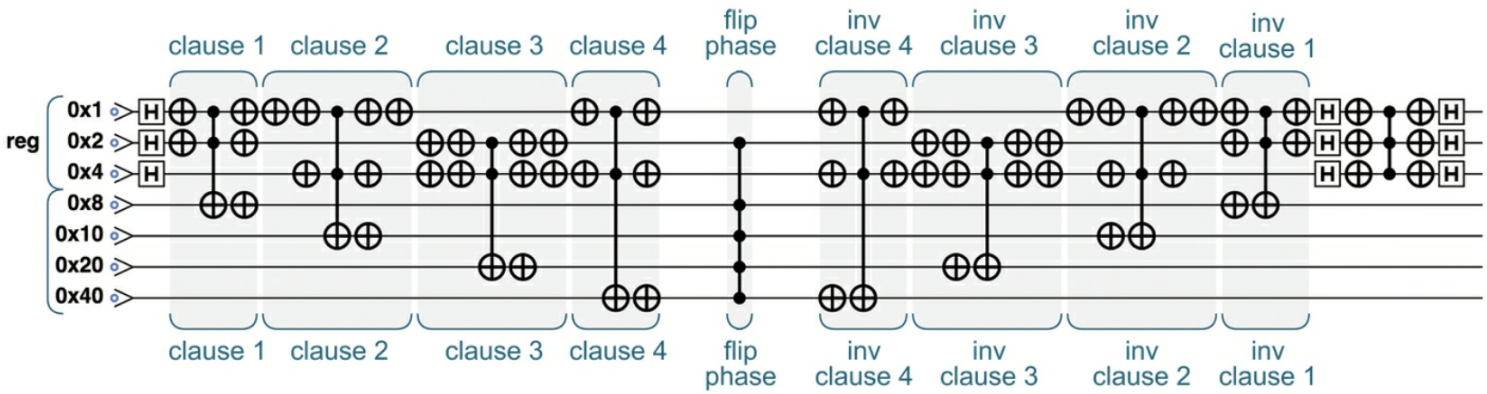


Figure 10-11. An unsatisfiable 3-SAT problem

On observera une distribution uniforme en sortie.

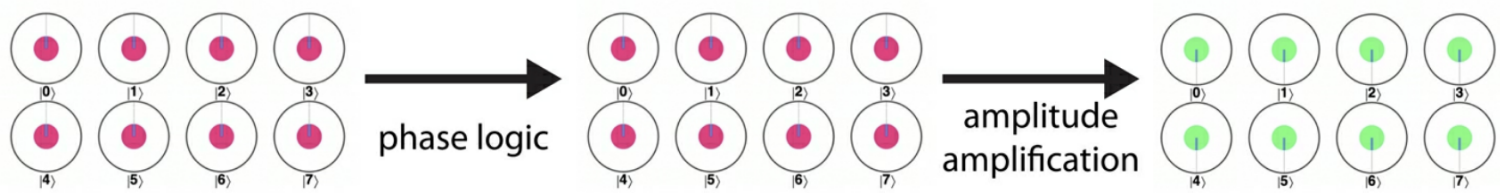


Figure 10-12. Circle notation for an unsatisfiable 3-SAT problem

Une question pourra alors se poser, les solutions marchent-elles toutes ou est-ce le contraire ? Pour cela, comment procéderiez-vous ?

Outline

1. Real Data
2. Quantum Search
3. Résoudre un problème logique par l'informatique quantique
4. Shor Factoring Algorithm
5. Quantum Machine Learning
   - Systèmes d'équations linéaires
   - Quantum Principal Component Analysis (QPCA)
   - Quantum Support Vector Machines (QSVM)
6. Annexes
   - Opérateurs simples
   - Décomposition en opérateurs simples
   - Algorithme de Deutsch-Josza
   - Algorithme de Simon
7. Conclusion

What Shor's Algorithm Does

Let's start with the `ShorQPU()` function. We're going to assert without proof a useful fact from number theory.[2] Namely, it's possible to solve the problem of finding prime factors p and q of a number $N = pq$ if we are able to solve the seemingly unrelated problem of finding the repeat period of the function $a^x \bmod(N)$, as the integer variable x is varied. Here N is still the number we want to factor; a is called the *coprime*. The value of the coprime can be chosen to be any prime number that we like.

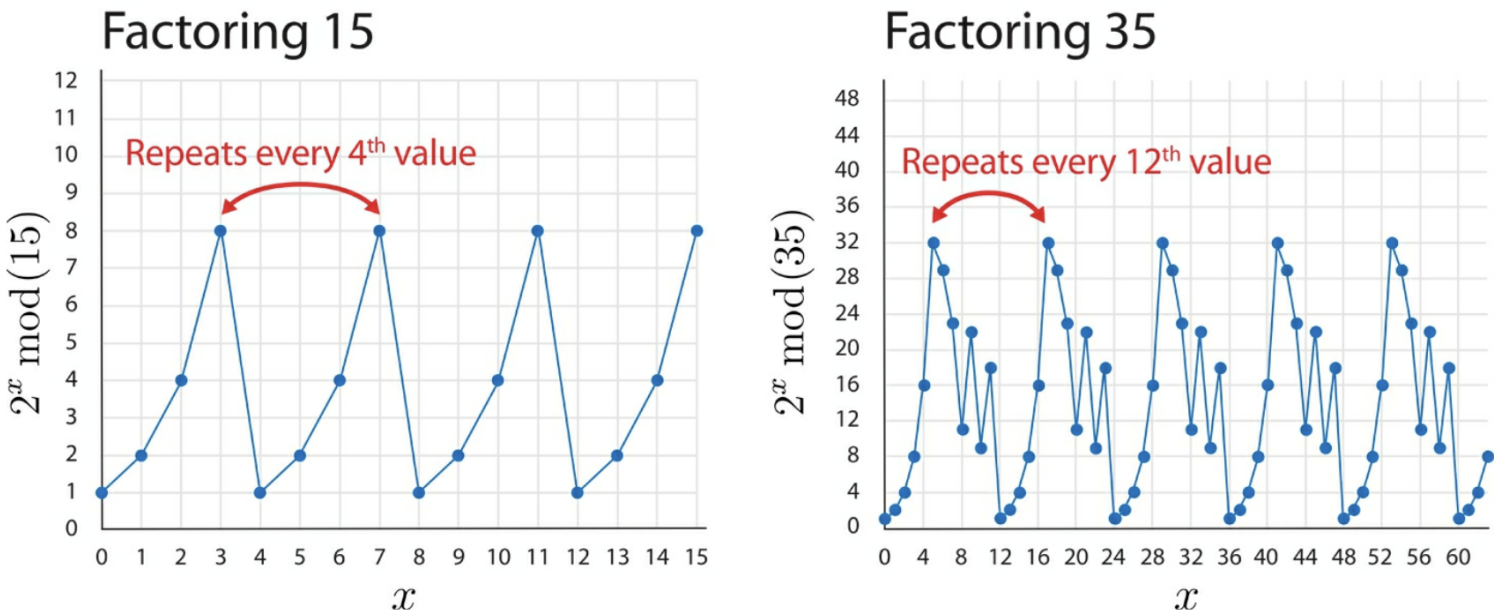


Figure 12-1. The repeat periods for two different values of N

Prenons $a = 2$, alors $a^0 \% N = 1$, $a^1 \% N = 2$, etc. On observe alors une période de 4 pour le cas $N = 15$ et de 12 pour le cas $N = 35$. Regardons comment en déduire p et q pour chaque situation.

Une fois que l'on connaît la période de répétition P, alors un des facteurs premiers de N pourrait être donné par $gcd(N, a^{P/2} + 1)$ et l'autre pourrait être donné par $gcd(N, a^{P/2} - 1)$.

Dans le cas $N = 15$, on obtient une période de 4, donc $p = gcd(15, 2^2 + 1) = 5$, et l'autre est donc $p = gcd(15, 2^2 - 1) = 3$ (rappel : gcd signifie "greatest common divisor", c'est-à-dire le plus grand diviseur commun).

Complexité de l'algorithme de Shor en fonction de N :

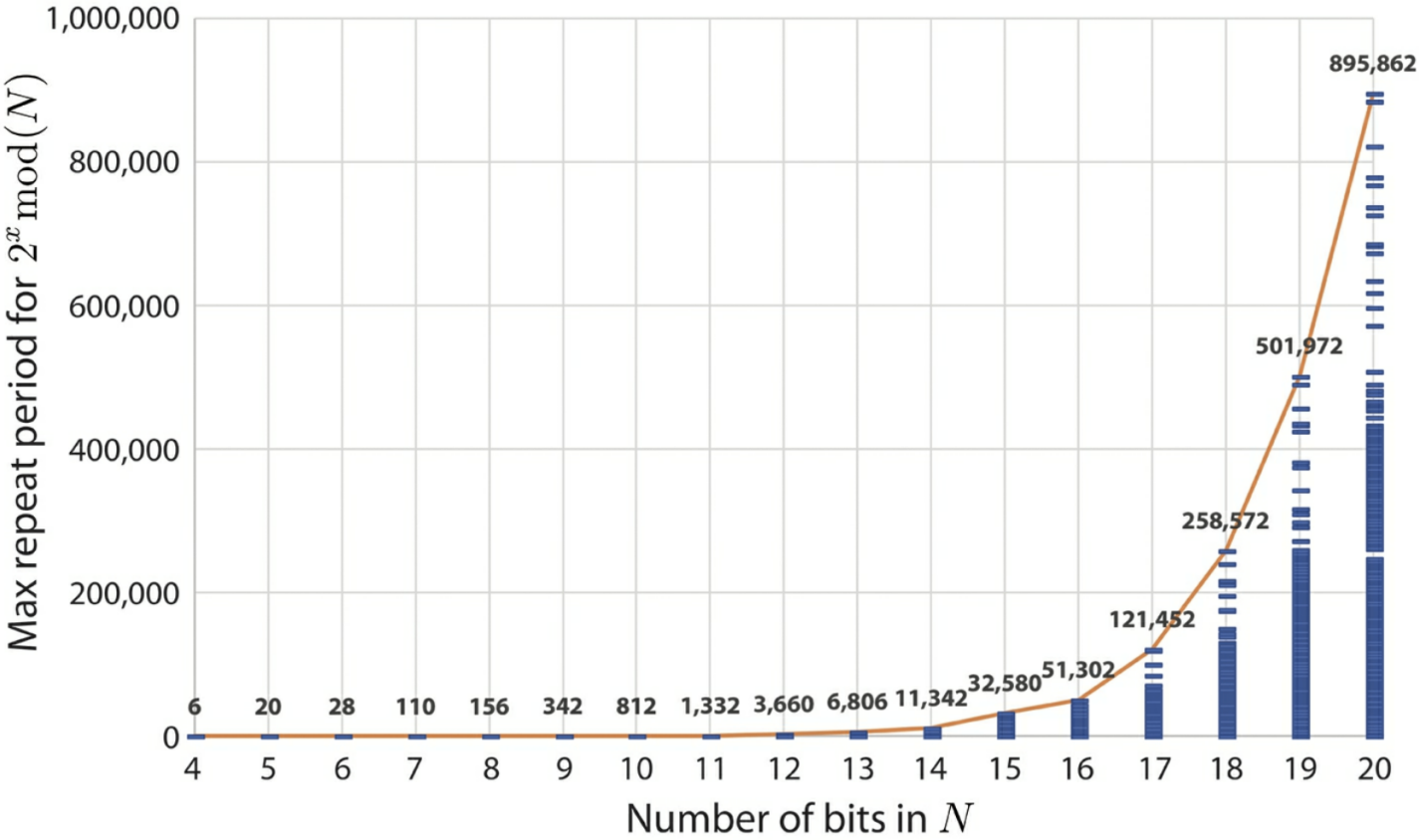


Figure 12-2. The maximum number of loops required to find the repeat period of an integer represented by a bitstring of length N. Each bar also displays a histogram showing the distribution of repeat periods for integers represented by a bitstring of length N.

On parle de périodicité, que se passe-t-il donc avec une DFT ?

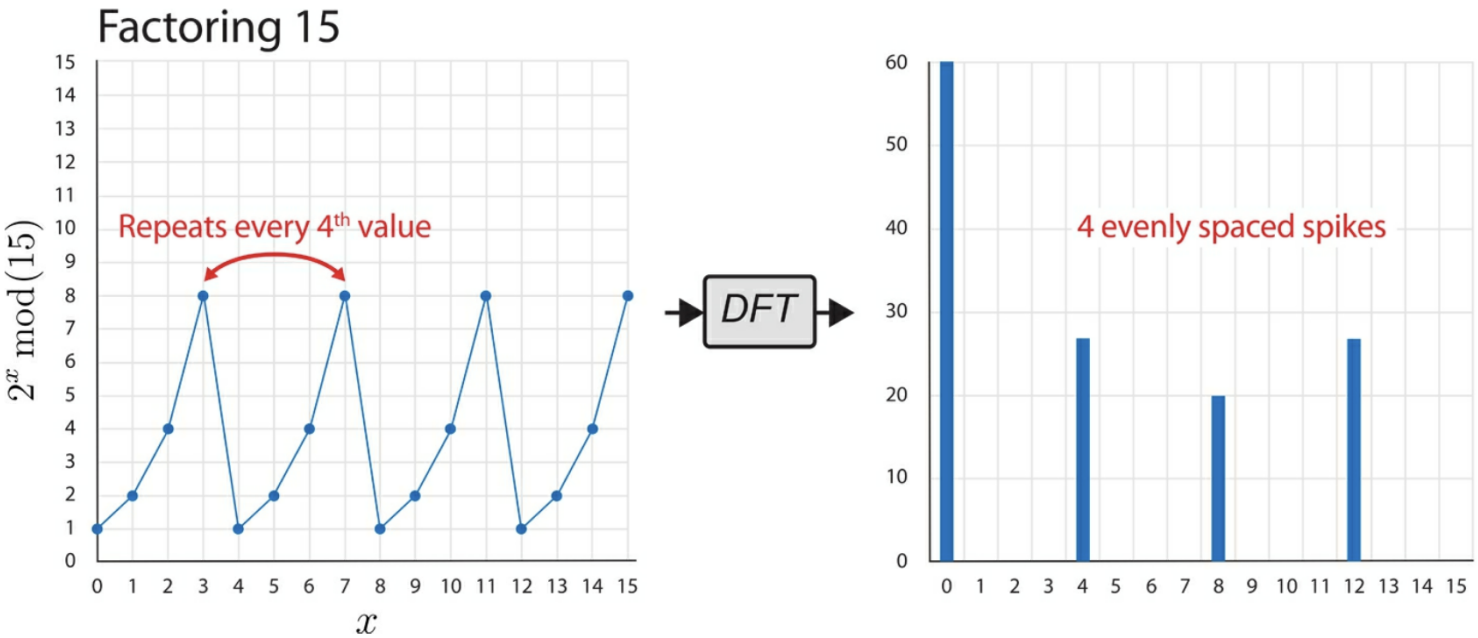


Figure 12-3. Computation performed while factoring 15

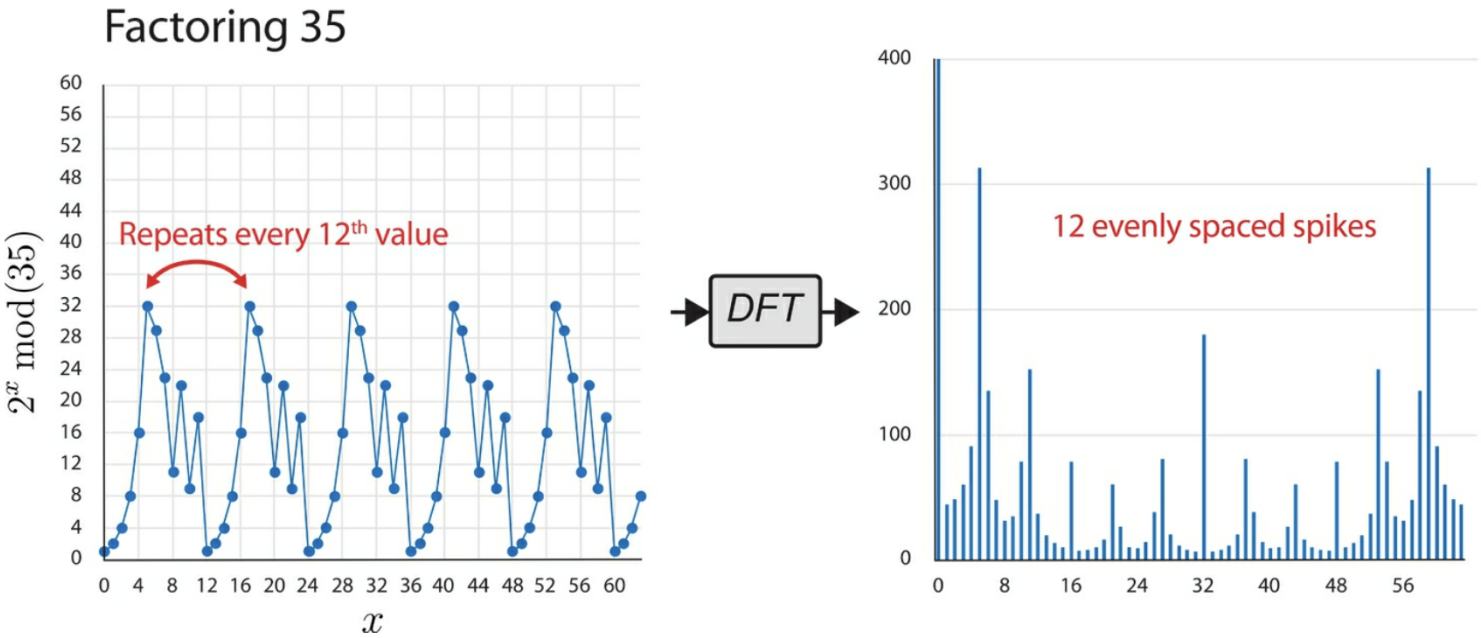


Figure 12-4. Computation performed while factoring 35

On voit que l'on peut déduire la périodicité en comptant le nombre de pics dans la DFT.

Voici donc un algorithme quantique à base de QFT (vous reconnaîtrez le pattern ci-dessous) :

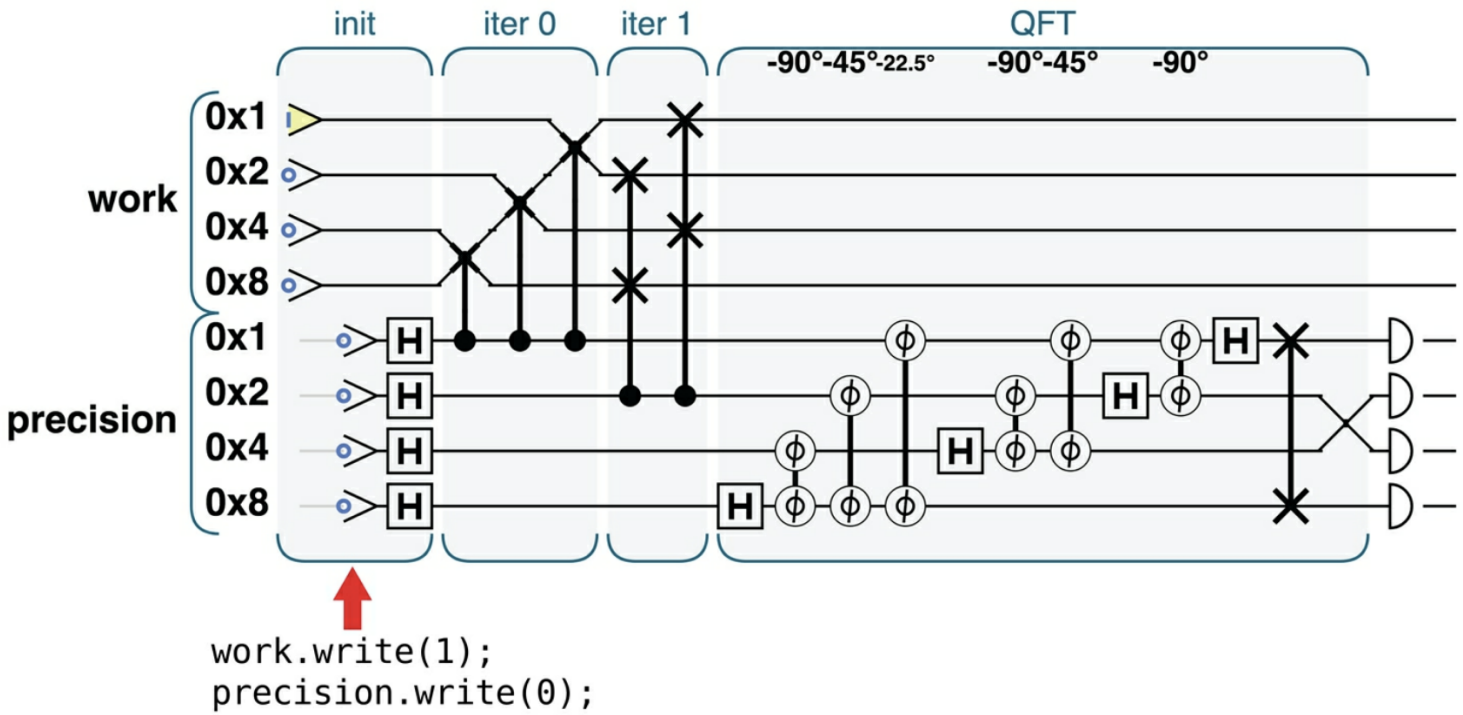


Figure 12-5. QPU instructions for step 1

On commence avec $\texttt{precision} = \texttt{0}$ et $\texttt{work} = 1$:

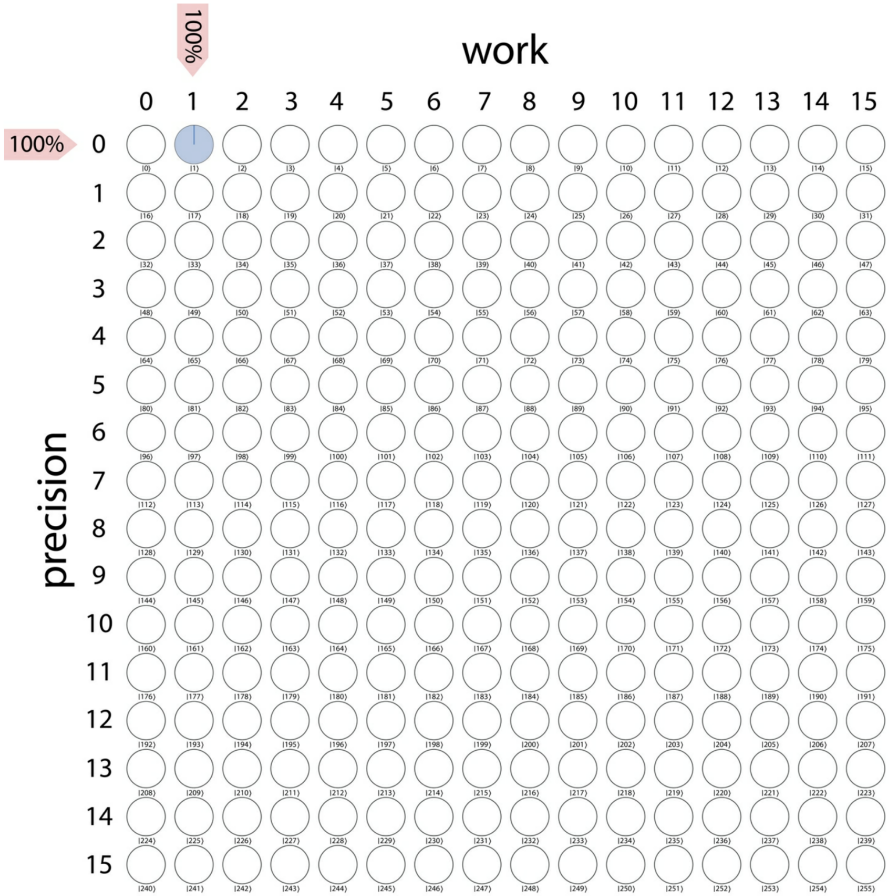


Figure 12-6. Step 1: Work and precision registers are initialized to values of 1 and 0, respectively

On applique des HAD pour avoir toutes les valeurs possibles dans `precision` :

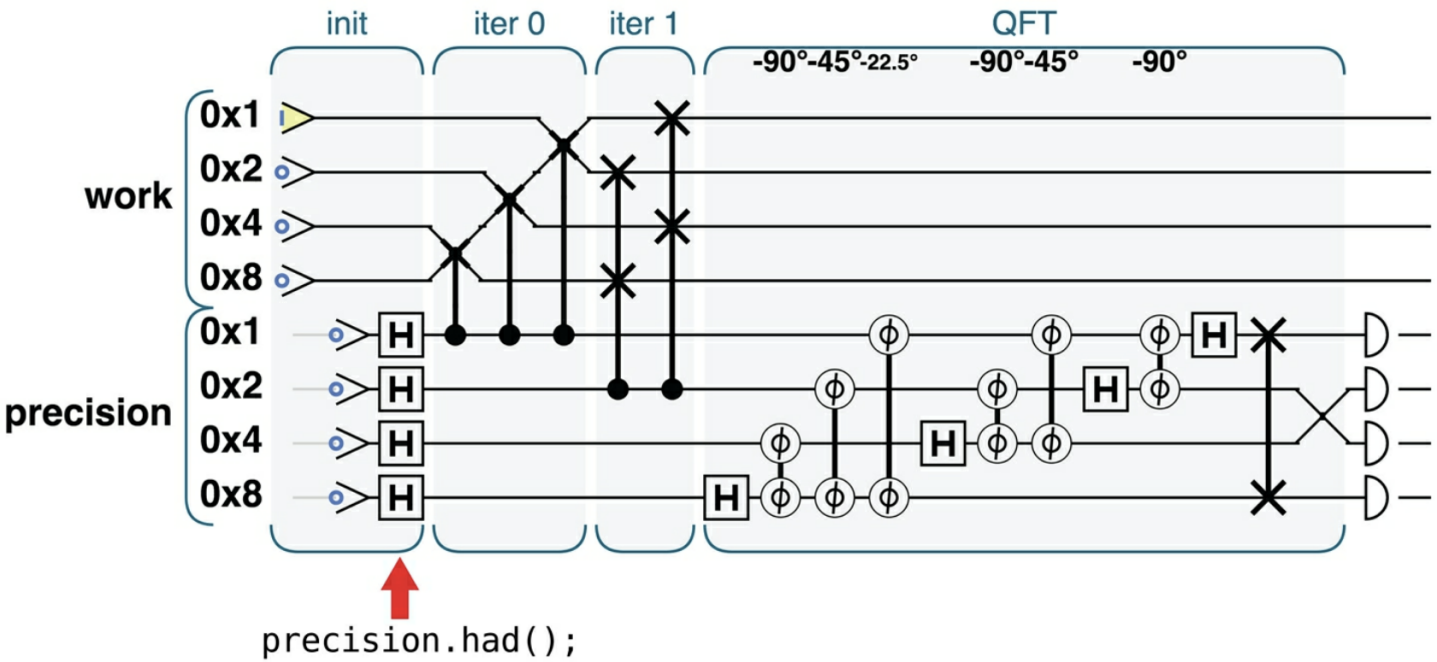
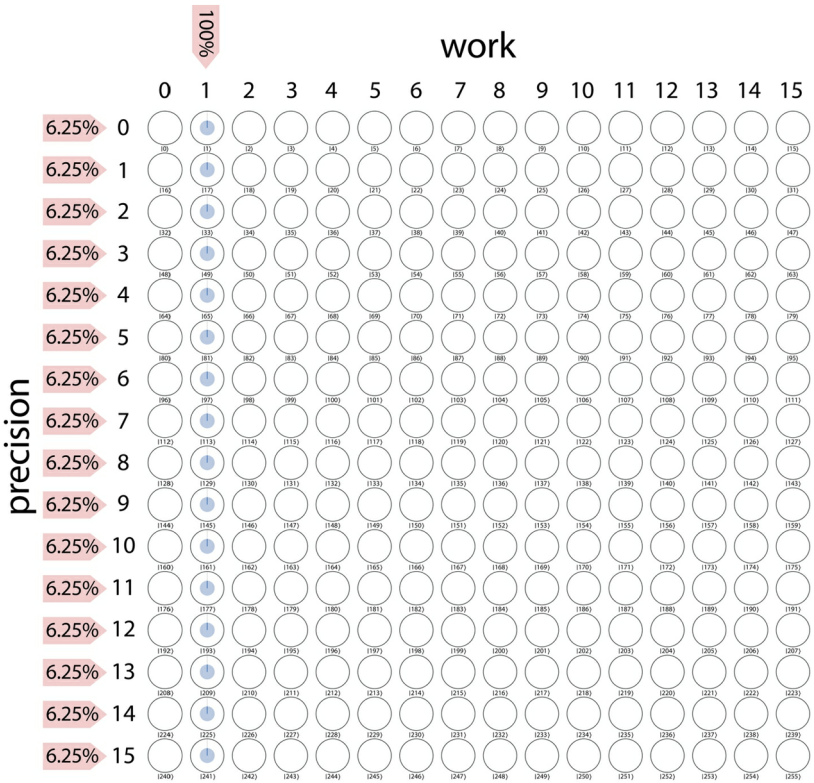


Figure 12-7. QPU instructions for step 2


100%
work
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
precision
6.25% 0
6.25% 1
6.25% 2
6.25% 3
6.25% 4
6.25% 5
6.25% 6
6.25% 7
6.25% 8
6.25% 9
6.25% 10
6.25% 11
6.25% 12
6.25% 13
6.25% 14
6.25% 15

On crée le signal avec `iter0` et `iter1` :

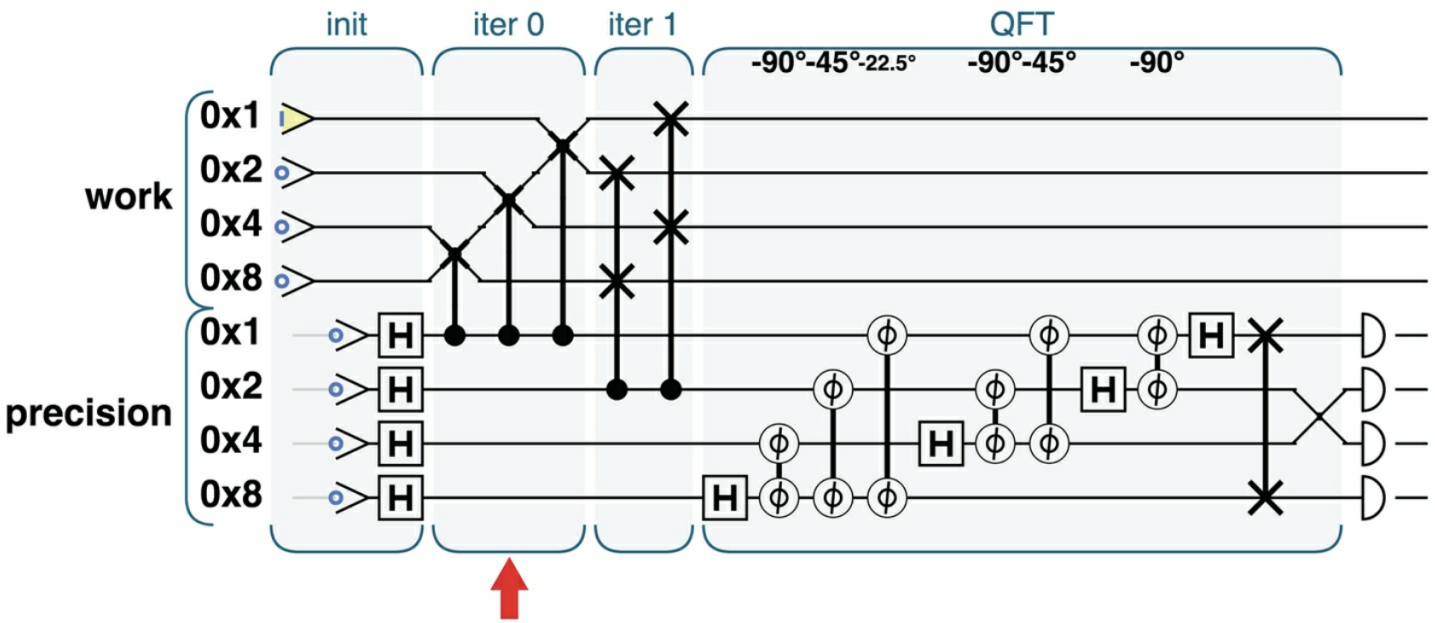
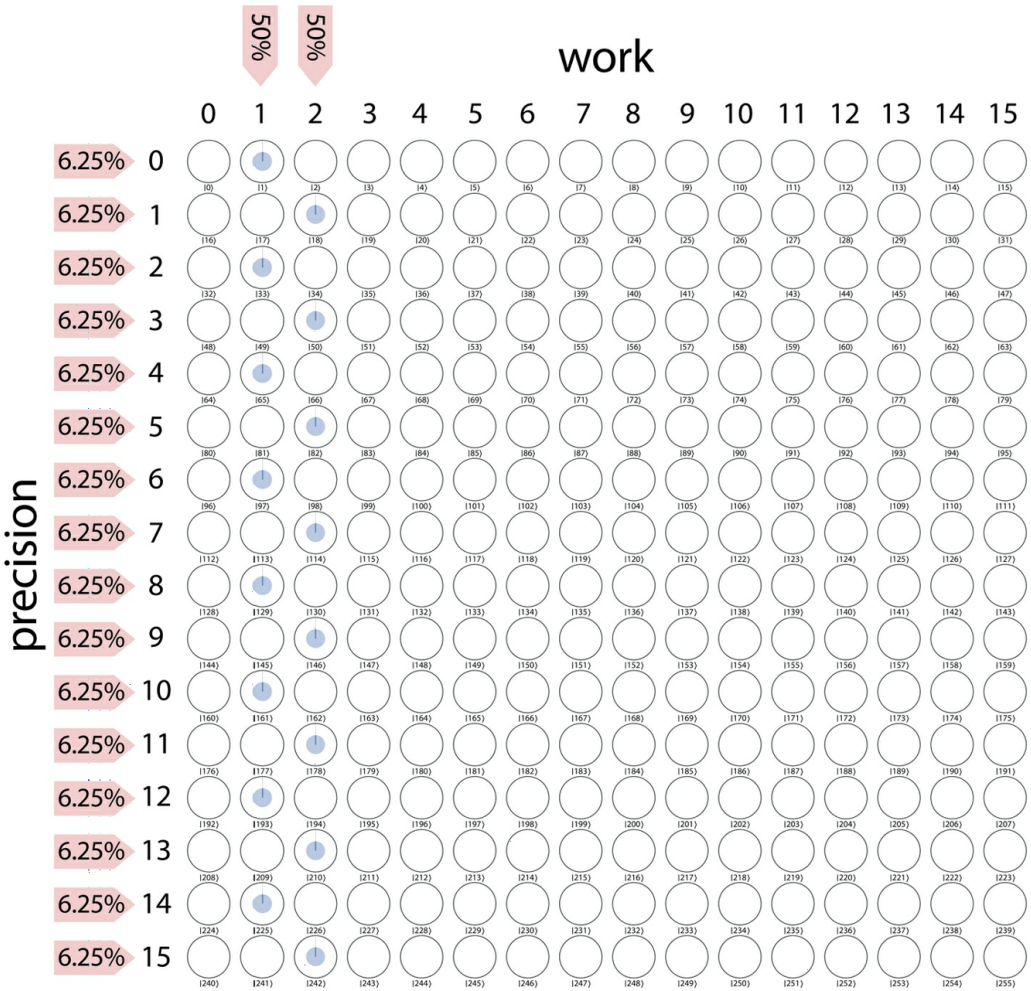


Figure 12-9. QPU instructions for step 3

The result is shown in Figure 12-10.



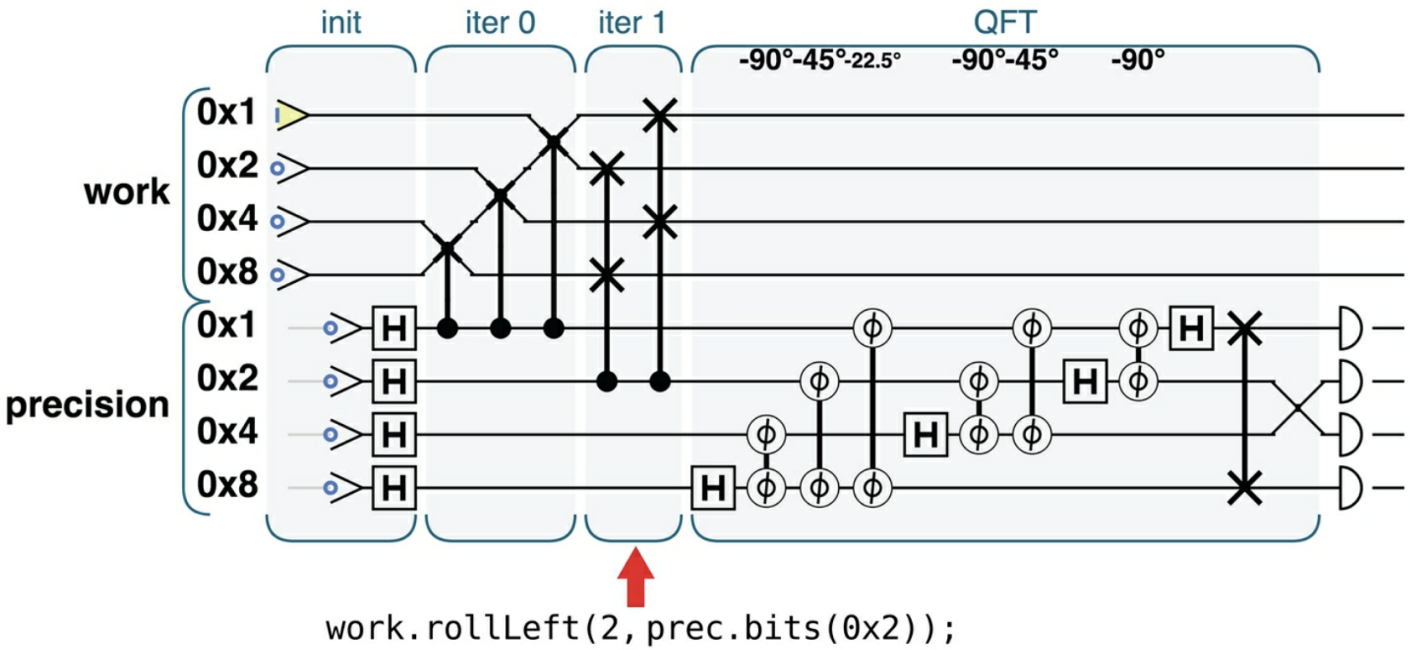


Figure 12-11. QPU instructions for step 4

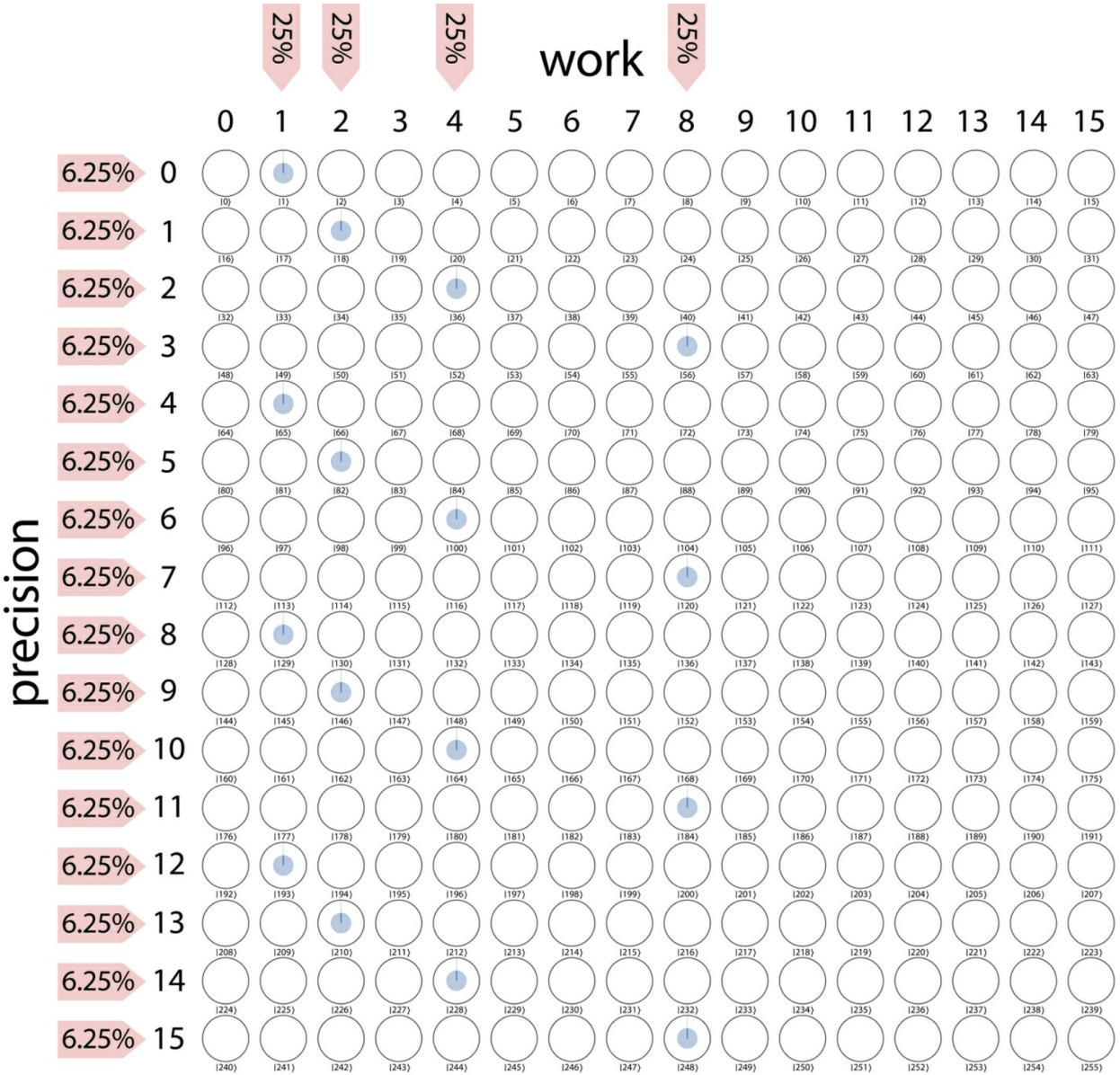


Figure 12-12. Step 4: The work register now holds a superposition of 2^x mod(15) for every possible value of x in the precision register

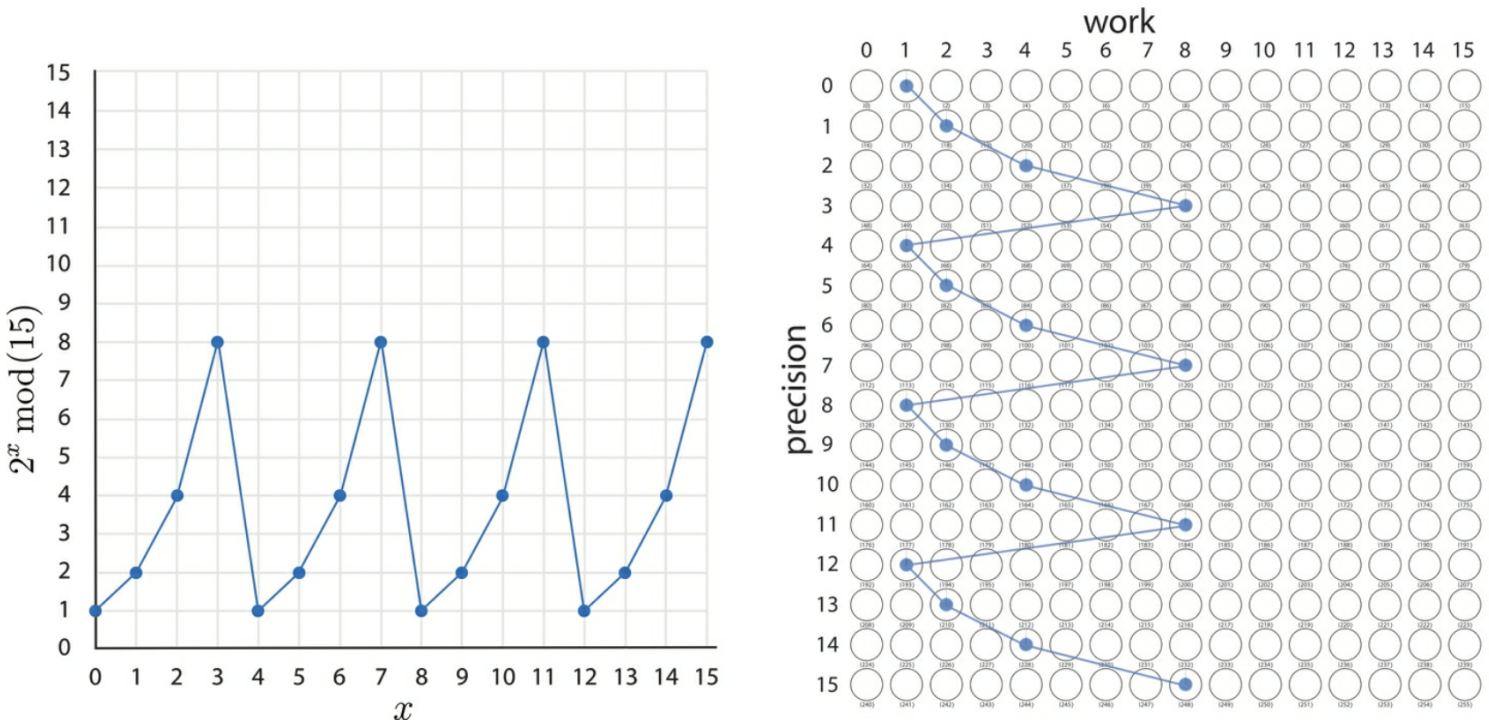


Figure 12-13. Hey, this looks familiar!

On a codé astucieusement un signal 1D temporel dans notre “matrice d’états” !

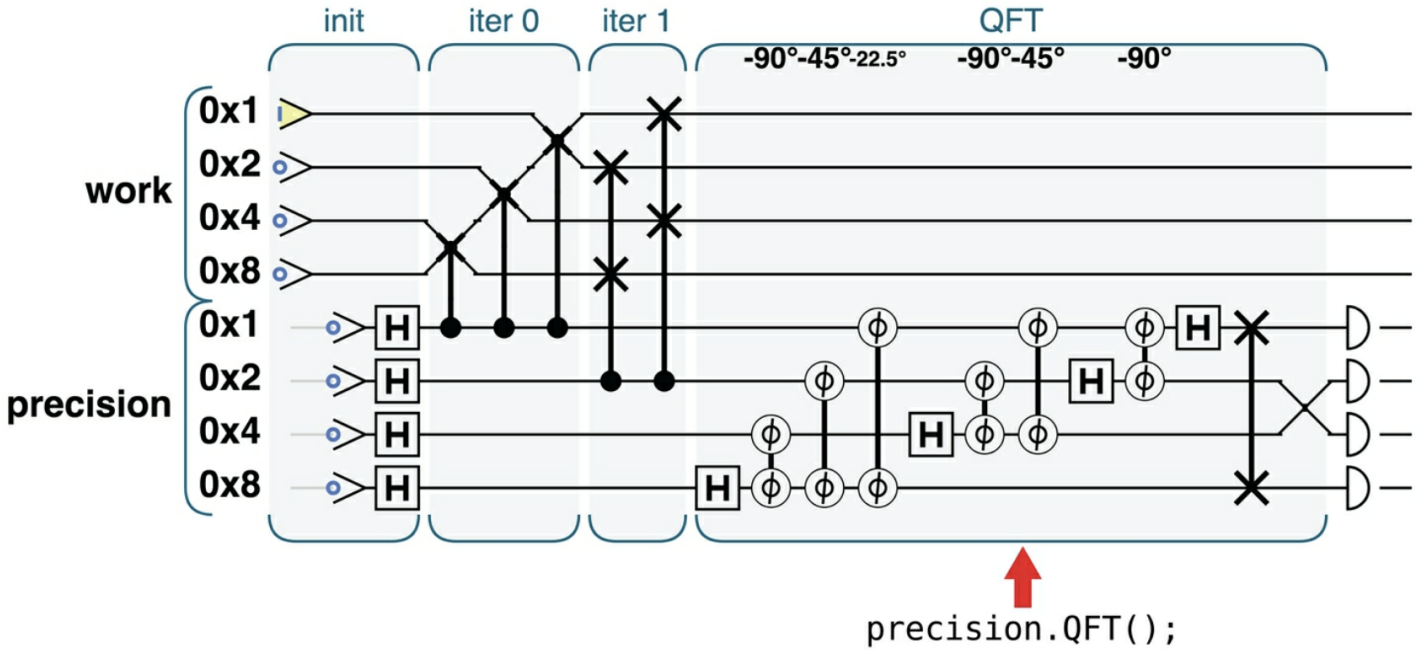


Figure 12-14. QPU instructions for step 5

On applique enfin Fourier.

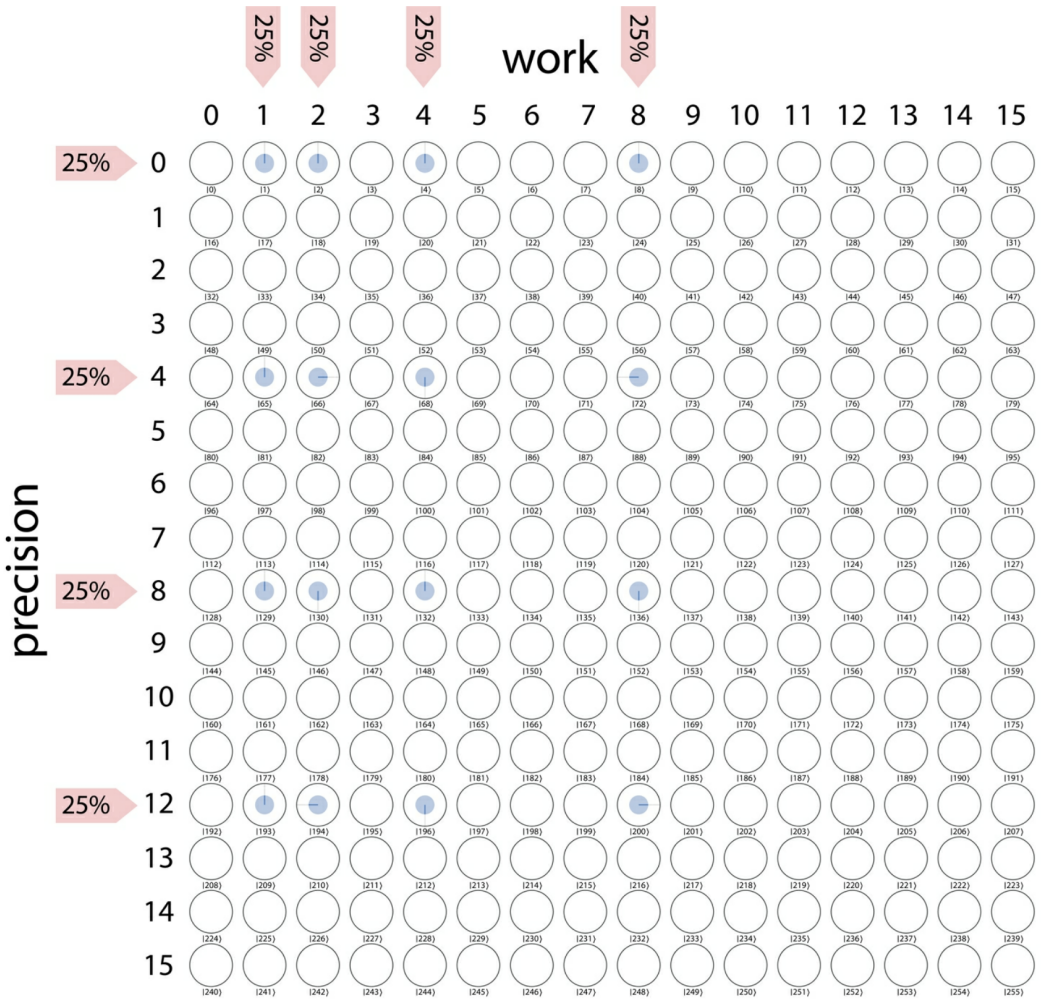


Figure 12-15. Step 5: Frequency spikes along the precision register following application of the QFT

On obtient donc nos “pics/spikes” dans cette matrice.

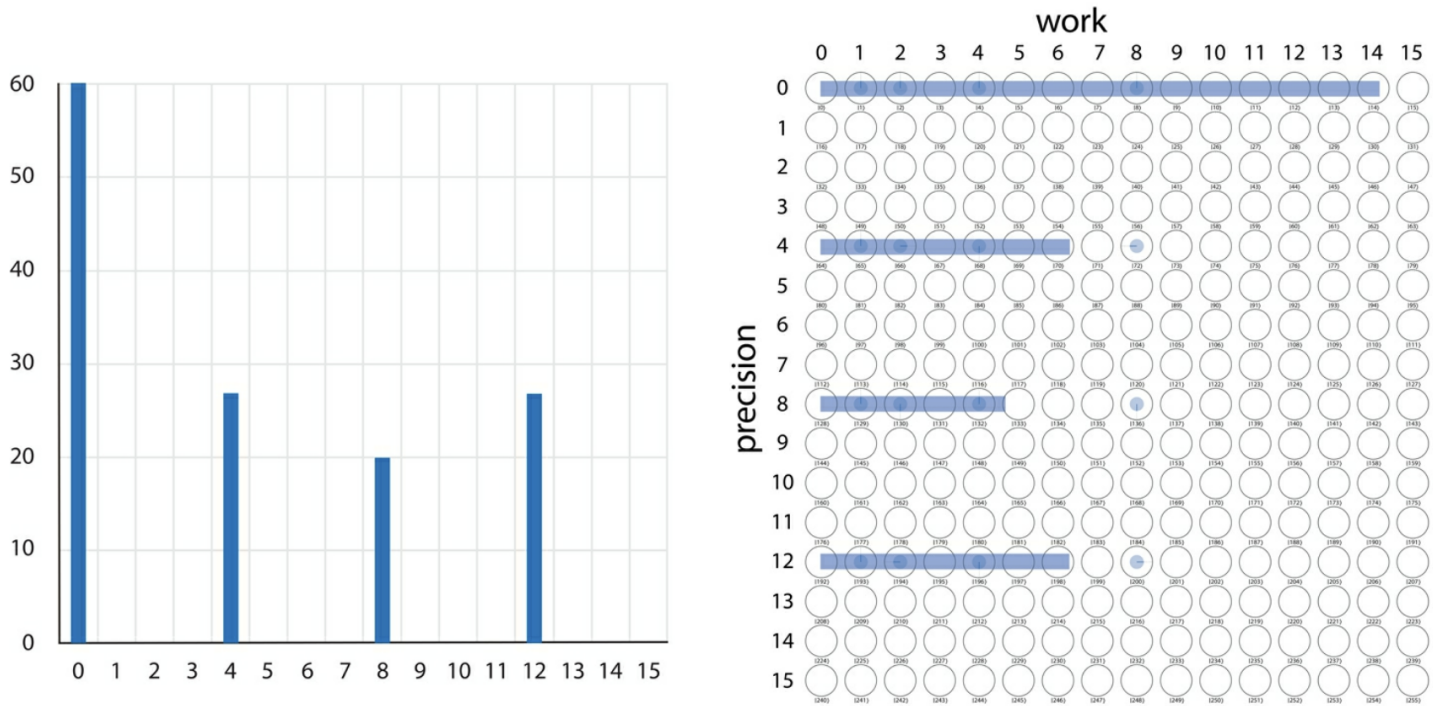


Figure 12-16. Once again, this resembles something we've seen

Il ne reste plus qu'à lire le registre `precision`.

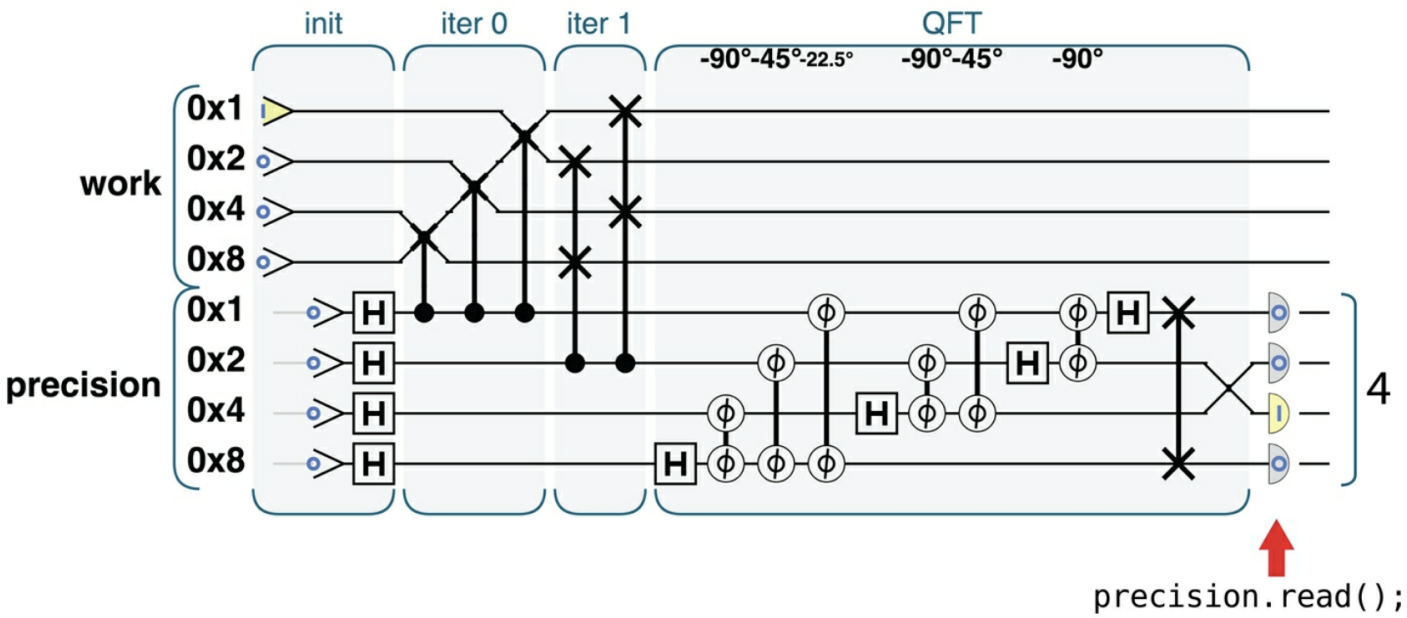


Figure 12-17. QPU instructions for step 6

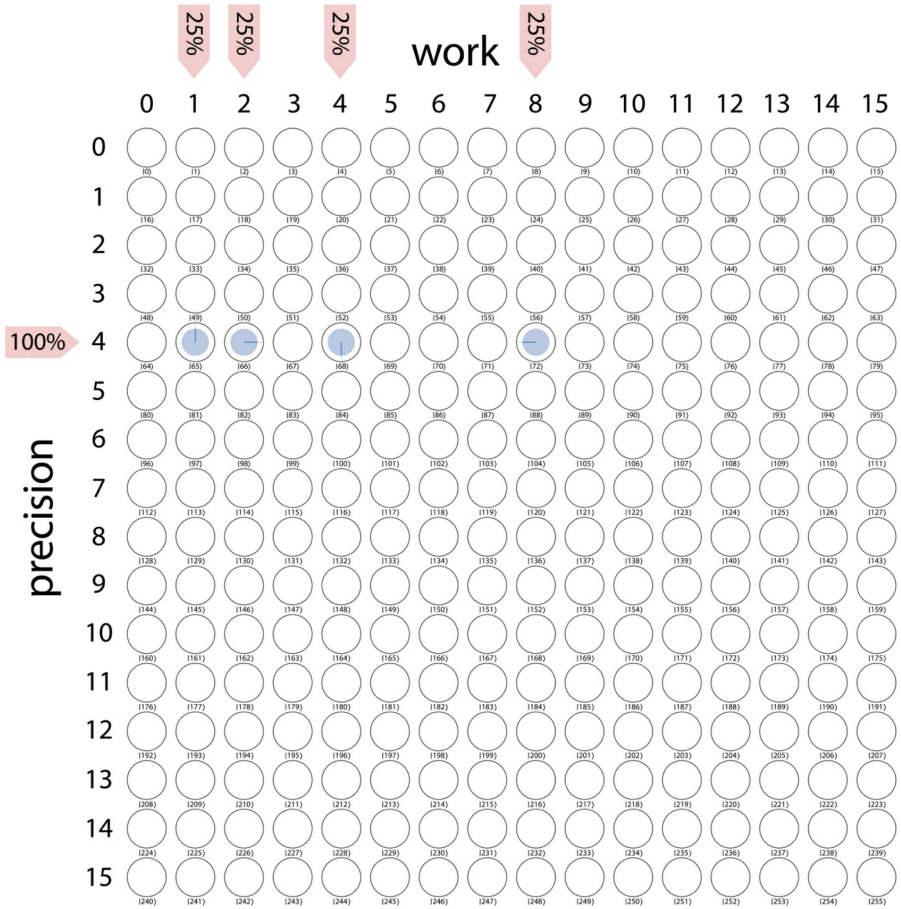


Figure 12-18. Step 6: After a READ on the precision register

The factors of a number can be difficult to find, but once found the answer is simple to verify. We can easily verify that 3 and 5 are both prime and factors of 15 (so there's no need to even try checking the second value of 8 in `ShorLogic()`).

Success!

Outline

1. Real Data
2. Quantum Search
3. Résoudre un problème logique par l'informatique quantique
4. Shor Factoring Algorithm
5. Quantum Machine Learning
   - Systèmes d'équations linéaires
   - Quantum Principal Component Analysis (QPCA)
   - Quantum Support Vector Machines (QSVM)
6. Annexes
   - Opérateurs simples
   - Décomposition en opérateurs simples
   - Algorithme de Deutsch-Josza
   - Algorithme de Simon
7. Conclusion

Systèmes d'équations linéaires

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning
- Systèmes d'équations linéaires
- Quantum Principal Component Analysis (QPCA)
- Quantum Support Vector Machines (QSVM)

6 Annexes

7 Conclusion

Algorithme de HHL (Harrow, Hassidim, et Lloyd, 2009) pour la résolution de systèmes d'équations linéaires:

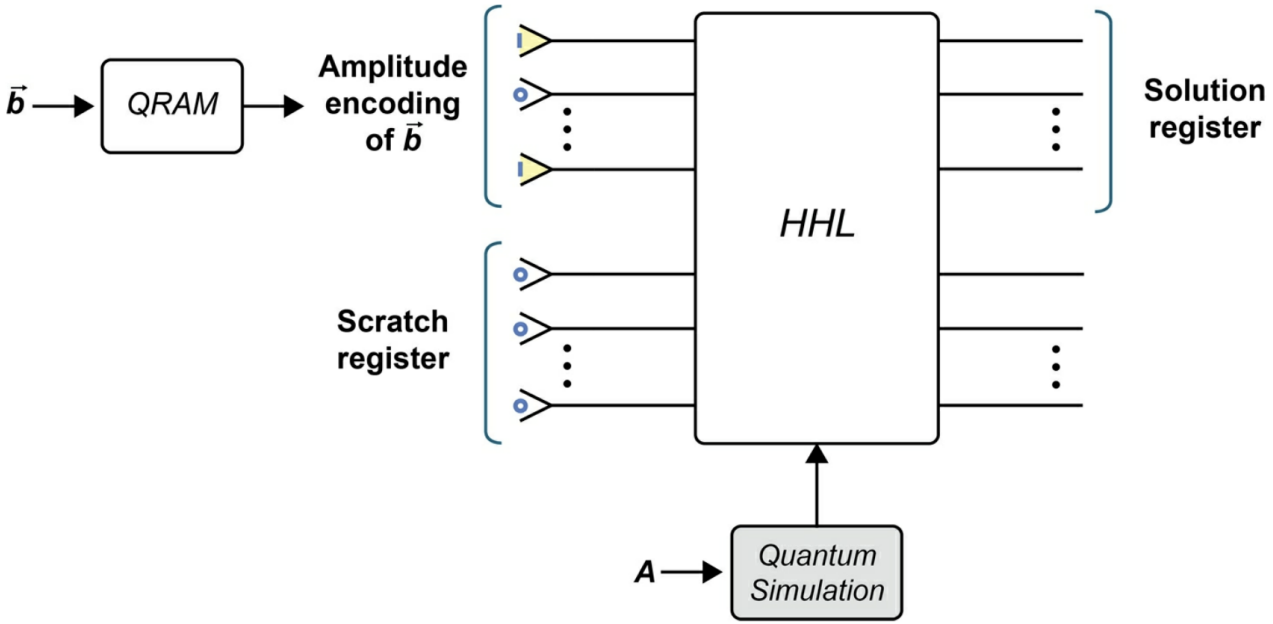


Figure 13-1. A high-level view of the inputs and outputs utilized by the HHL algorithm to solve systems of linear equations.

Speed and fine print

The HHL algorithm has a runtime[5] of $O(\kappa^2 s^2 \epsilon^{-1} \log n)$.

In comparison with the conventional method of conjugate gradient descent and its runtime of $O(ns\kappa \log(1/\epsilon))$, HHL clearly offers an exponential improvement in the dependence on the size of the problem (n).

Si on veut voir cet algorithme plus en détail :

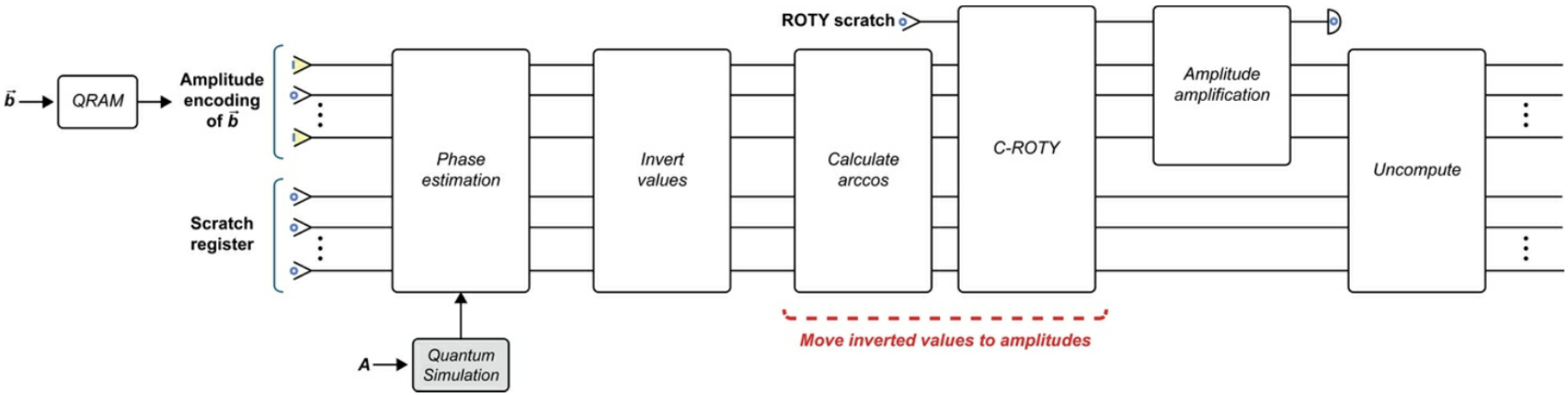


Figure 13-2. Schematic outline of the primitives contained in the HHL algorithm

Quantum Principal Component Analysis (QPCA)

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

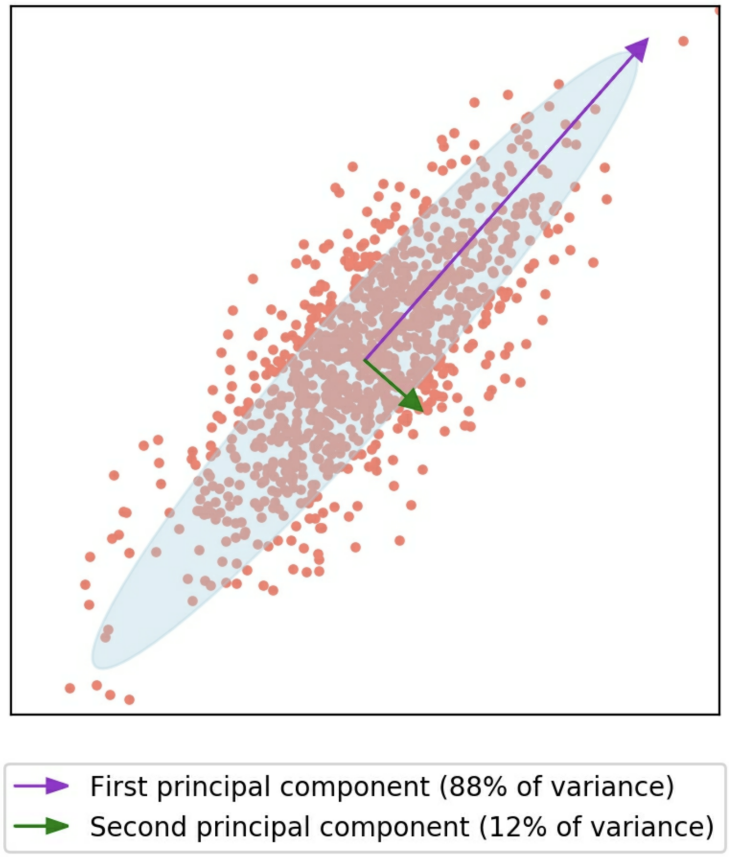
4 Shor Factoring Algorithm

5 Quantum Machine Learning
- Systèmes d'équations linéaires
- Quantum Principal Component Analysis (QPCA)
- Quantum Support Vector Machines (QSVM)

6 Annexes

7 Conclusion

On part d'une distribution de points et on veut estimer sa matrice de covariance, ses valeurs et ses vecteurs propres :



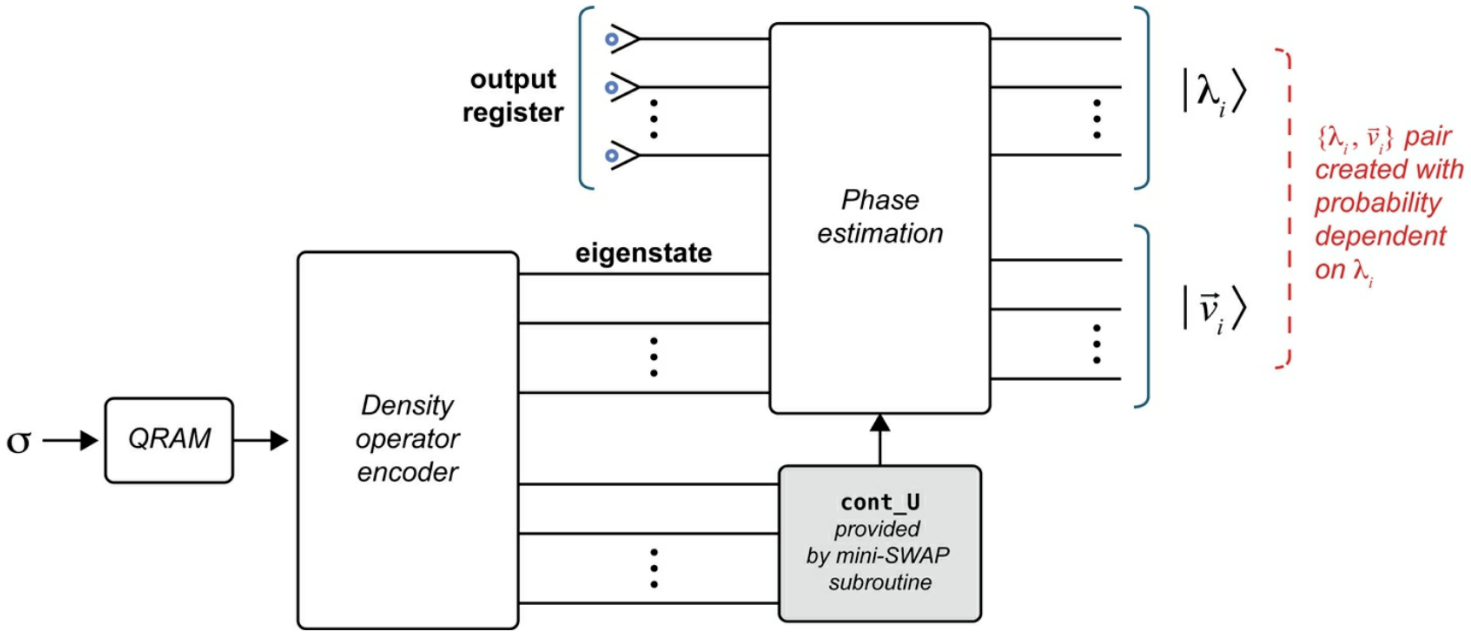


Figure 13-5. Schematic for QPCA

Quantum Support Vector Machines (QSVM)

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning
- Systèmes d'équations linéaires
- Quantum Principal Component Analysis (QPCA)
- Quantum Support Vector Machines (QSVM)

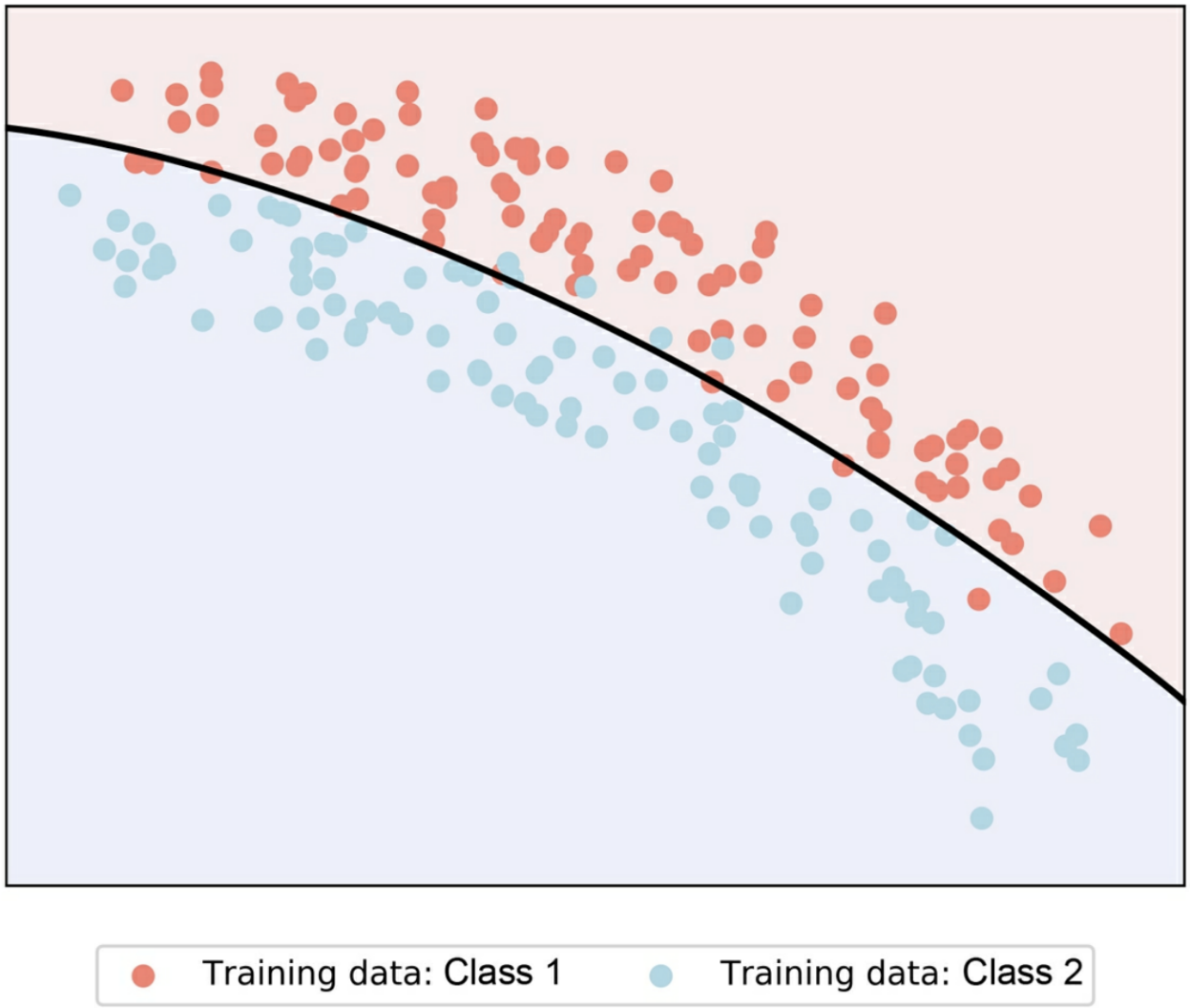
6 Annexes

7 Conclusion

On a deux classes de points et on cherche la séparation (linéaire ou non selon la SVM choisie) qui maximise la marge.

Figure 13-6. Example of the classification hyperplane produced by an SVM for a two-class classification problem with two features

Ici on a le cas linéaire.



Et là un cas non linéaire.

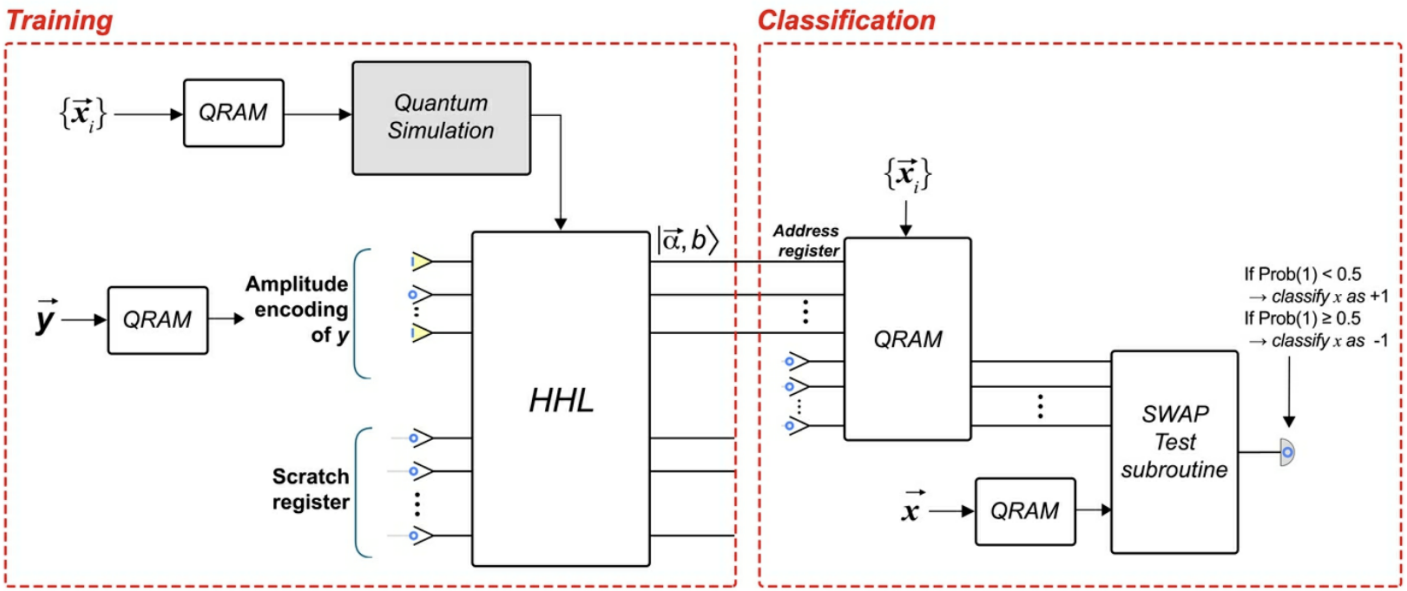


Figure 13-8. Schematic showing the stages in training and classifying with a QSVM

Outline

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning
- Systèmes d'équations linéaires
- Quantum Principal Component Analysis (QPCA)
- Quantum Support Vector Machines (QSVM)

6 Annexes
- Opérateurs simples
- Décomposition en opérateurs simples
- Algorithme de Deutsch-Josza
- Algorithme de Simon

7 Conclusion

Opérateurs simples

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning

6 Annexes
- Opérateurs simples
- Décomposition en opérateurs simples
- Algorithme de Deutsch-Josza
- Algorithme de Simon

7 Conclusion

En réalité, les opérateurs simples sont identifiables à des matrices :

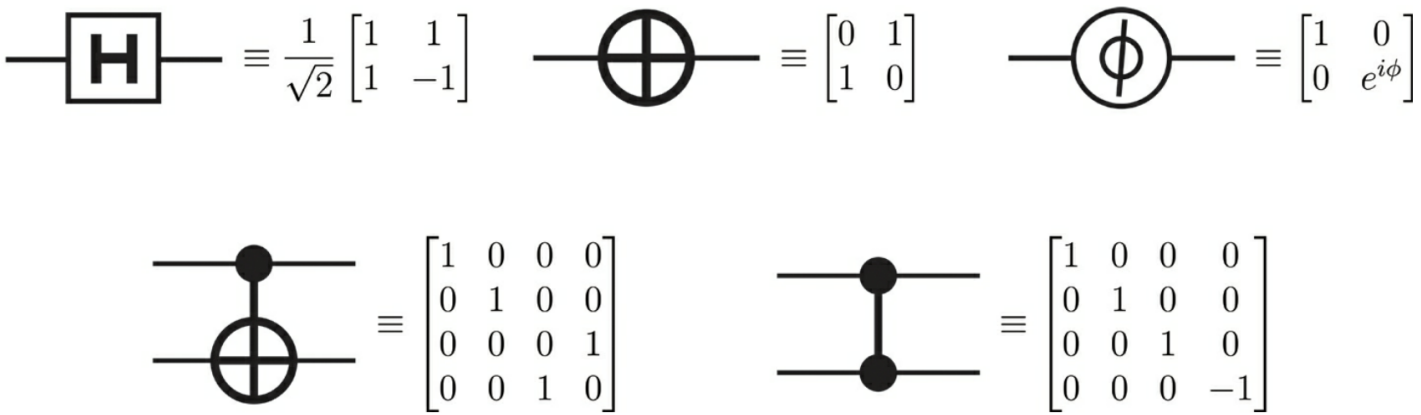


Figure 14-1. Matrix representations of the most basic single- and two-qubit gates

Décomposition en opérateurs simples

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning

6 Annexes
- Opérateurs simples
- Décomposition en opérateurs simples
- Algorithme de Deutsch-Josza
- Algorithme de Simon

7 Conclusion

Dans le cas d'une matrice unitaire :

> Fortunately, more complex conditional operations can be written as series of single-qubit and two-qubit operations. In Figure 14-2 we show a general decomposition of a controlled QPU operation (corresponding to a general unitary matrix in the mathematics of quantum computing). The constituent operations in this decomposition need to be chosen such that A, B, and C satisfy $U = e^{i\alpha} AXBXC$ (where X is the NOT gate) and acting all three operations directly one after the other, $A \cdot B \cdot C$, has no overall effect on our QPU register state.

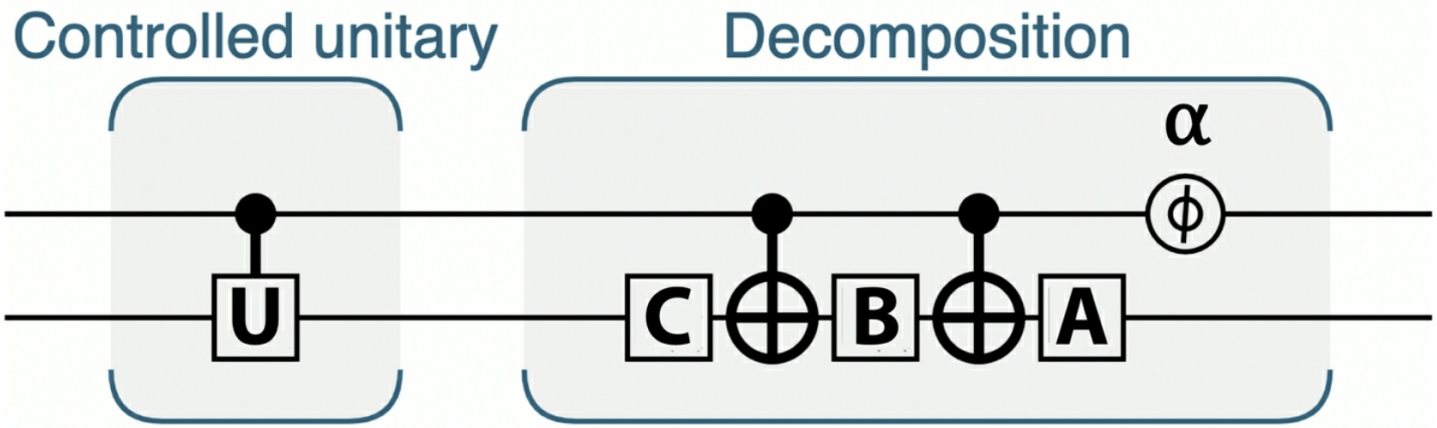


Figure 14-2. General decomposition of a controlled unitary

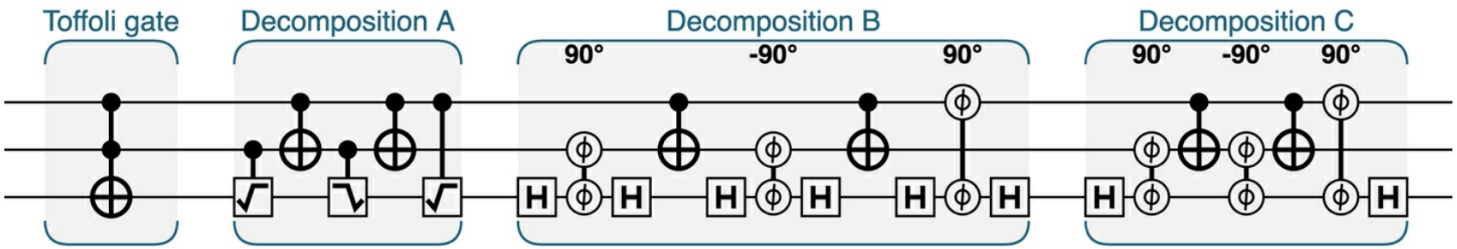


Figure 14-3. The familiar CCNOT gate can be decomposed into more basic operations

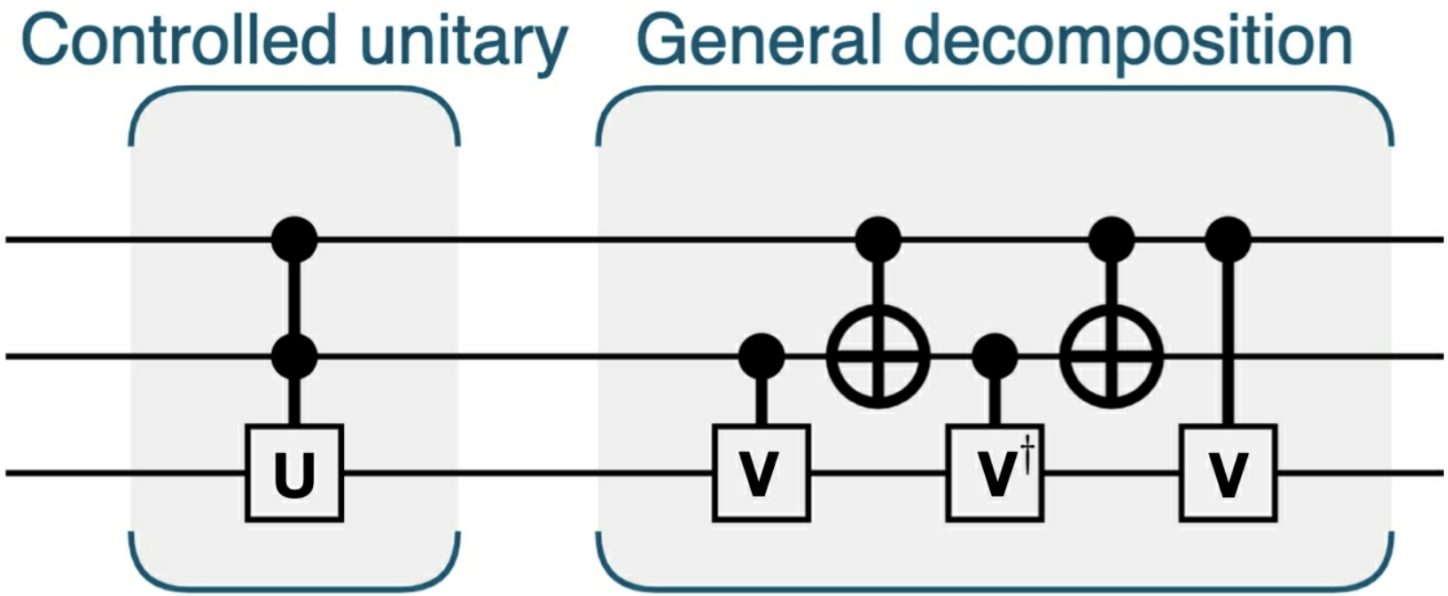


Figure 14-4. A general controlled-controlled unitary decomposition

Algorithme de Deutsch-Josza

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning

6 Annexes
- Opérateurs simples
- Décomposition en opérateurs simples
- Algorithme de Deutsch-Josza
- Algorithme de Simon

7 Conclusion

Deutsch-Jozsa

Oracle

Takes a binary string of n bits and outputs a single bit that is the result of applying a function f to the binary string. We are promised that the function f is either constant, in which case the output bit will always be the same, or balanced, in which case there will be the same number of 0 and 1 outputs.

Problem

Decide with absolute certainty whether f is constant or balanced in as few queries to the oracle as possible.

Query complexity

Conventionally, we will need to make $2^{n-1} + 1$ queries to the oracle to be absolutely sure about the nature of the function. With a QPU, we can solve the problem with zero probability of error *in a single quantum query!*

This algorithm can be run online at http://oreilly-qc.github.io?p=14-DJ.

Algorithme de Simon

1 Real Data

2 Quantum Search

3 Résoudre un problème logique par l'informatique quantique

4 Shor Factoring Algorithm

5 Quantum Machine Learning

6 Annexes
- Opérateurs simples
- Décomposition en opérateurs simples
- Algorithme de Deutsch-Josza
- Algorithme de Simon

7 Conclusion

Simon

Oracle

Takes an n-bit binary string, x, and outputs a single integer. All the possible input strings are paired through a secret string s, such that two strings (x,y) will result in the same output if and only if $y = x \oplus s$ (where $\oplus$ denotes bitwise addition modulo 2).

Problem

Find the secret string s.

Query complexity

A conventional deterministic algorithm will require at least $2^{n-1} + 1$ oracle queries. By using Simon's quantum algorithm, we can find the solution in a number of calls that scales linearly in n, rather than exponentially.

This algorithm can be run online at http://oreilly-qc.github.io?p=14-S.

Outline

1. Real Data
2. Quantum Search
3. Résoudre un problème logique par l'informatique quantique
4. Shor Factoring Algorithm
5. Quantum Machine Learning
   - Systèmes d'équations linéaires
   - Quantum Principal Component Analysis (QPCA)
   - Quantum Support Vector Machines (QSVM)
6. Annexes
   - Opérateurs simples
   - Décomposition en opérateurs simples
   - Algorithme de Deutsch-Josza
   - Algorithme de Simon
7. Conclusion

Pensez-bien à pratiquer sur QCEngine les exercices fournis, et bon courage :)