Master QLMN Pascal Scholl
Experimental quantum computing pascal.scholl@pasqal.com

Lecture 5 - Quantum algorithms: focus on Shor algorithm

January 28th 2026

This lecture introduces the algorithms that can be implemented on a quantum computer and that enable an improvement in performances with respect to their classical counterparts.

Such algorithms can be separated into two subgroups:

- **Algorithms with exponential speedups:** These algorithms are often seen as the most promising at the mathematical level, as they allow to address NP-hard problems in polynomial time. However, they often require extremely powerful quantum computers, and there is a very small number of applications that belong to this class: algebraic and cryptography. We will address this category in this lecture, focusing on the most notorious example: Shor algorithm.
- **Algorithms with (super)polynomial speedups:** These algorithms might seem less interesting at the mathematical level, but have proven to be more near-term friendly for practical applications, and their range of potential application is much larger than the first category. In this class, we find algorithms such as Grover which allows to perform optimization tasks, quantum simulation, and machine learning / data treatment. These will be addressed in Lecture 6.

We note that quantum applications are a (strongly evolving) field of research: every year new applications, and optimization of their implementation on quantum computers (inducing a change in the speedup) are discovered.

We will first globally introduce the algorithms of the first group, then dive into the details of Shor's algorithm. We will perform the algorithm end-to-end, highlighting both the classical part of the algorithm, and the quantum part. We will discuss the quantum circuits which need to be implemented on a quantum computer in order to execute the algorithm. We will ground the explanations on real examples. We will in particular detail the iconic sub-routines of Shor's algorithm which are used in many algorithms: the Quantum Fourier Transform (QFT), and the modular exponentiation. We will finally detail the origin of the exponential gain of this algorithm.

1 Quantum algorithms with exponential speedups

The known algorithms which have an exponential speedup all share the same idea. If we are given a periodic function, even when the structure of the periodicity is quite complicated, we can often use a quantum algorithm to determine the period efficiently.

The general problem which defines a broad framework for these questions can be mathematically expressed in the language of group theory as follows:

- Let f be a function from a finitely generated group G to a finite set X such that f is constant on the cosets of a subgroup K, and distinct on each coset. Given a quantum black box for performing the unitary transform $U|g\rangle|h\rangle = |g\rangle|h \oplus f(g)\rangle$, for $g \in G$, $h \in X$, and $\oplus$ an appropriately chosen binary operation on X, find a generating set for K.

In (hopefully) simpler terms, given a black-box function f which we know have some kind of unknown periodicity (meaning that the function gives the same value to elements related by a hidden symmetry), the quantum computer can discover that hidden periodicity very efficiently.

A list of such algorithms is provided in Figure 1. In particular, we find the Deutsch algorithm which was introduced in Lecture 1. We observe that there are not many known algorithms belonging to this category, and finding others in the same spirit is an active field of research.

We next describe the most famous algorithm belonging to this category with the most impactful known application: finding prime numbers of integers N.

2 The classical part of Shor's algorithm

2.1 Algorithm description

As mentioned above, the aim of Shor's algorithm is to factorize a given number N. For the sake of simplicity and to connect the algorithm to cryptography, we will assume that $N = pq$ with p and q two prime numbers. The algorithm works as follows:

1. Select a random number $1 < a < N$.

2. Check whether the greatest common denominator is one: $\gcd(a, N) = 1$. If not, the problem is solved without the need of the rest of the algorithm (since N is a large number, it is highly unlikely).

3. **Period finding** Find the smallest number r for which $a^r \equiv 1\,[N]$. This is where the quantum computer is used (the *period finding algorithm* is present in Figure 1): it will find the value of r that satisfies this relationship.

4. **Classical post-processing** Having obtained $a^r \equiv 1\,[N]$ from the quantum processor, we can rewrite $(a^{r/2}-1)(a^{r/2}+1) \equiv 0\,[N]$, meaning that $(a^{r/2}-1)(a^{r/2}+1)$ is divisible by N: we're getting close to finding the prime factors. Further classical post-processing enables to get for example $(a^{r/2}-1) \to p$ and $(a^{r/2}+1) \to q$ (detailed below). However, there is a $\sim 1/2$ probability to get $(a^{r/2}-1) \to 1$ and $(a^{r/2}-1) \to N$, and thus we do not gain any information. In this case, the algorithm failed.

Name	G	X	K	Function
Deutsch	$\{0,1\}, \oplus$	$\{0,1\}$	$\{0\}$ or $\{0,1\}$	$K = \{0,1\} : \begin{cases} f(x) = 0 \\ f(x) = 1 \end{cases}$ $K = \{0\} : \begin{cases} f(x) = x \\ f(x) = 1 - x \end{cases}$
Simon	$\{0,1\}^n, \oplus$	any finite set	$\{0, s\}$ $s \in \{0,1\}^n$	$f(x \oplus s) = f(x)$
Period-finding	$\mathbf{Z}, +$	any finite set	$\{0, r, 2r, \ldots\}$ $r \in G$	$f(x + r) = f(x)$
Order-finding	$\mathbf{Z}, +$	$\{a^j\}$ $j \in Z_r$ $a^r = 1$	$\{0, r, 2r, \ldots\}$ $r \in G$	$f(x) = a^x$ $f(x + r) = f(x)$
Discrete logarithm	$\mathbf{Z}_r \times \mathbf{Z}_r$ + (mod r)	$\{a^j\}$ $j \in Z_r$ $a^r = 1$	$(\ell, -\ell s)$ $\ell, s \in \mathbf{Z}_r$	$f(x_1, x_2) = a^{kx_1 + x_2}$ $f(x_1 + \ell, x_2 - \ell s) = f(x_1, x_2)$
Order of a permutation	$\mathbf{Z}_{2^m} \times \mathbf{Z}_{2^n}$ + (mod 2^m)	$\mathbf{Z}_{2^n}$	$\{0, r, 2r, \ldots\}$ $r \in X$	$f(x, y) = \pi^x(y)$ $f(x + r, y) = f(x, y)$ π = permutation on X
Hidden linear function	$\mathbf{Z} \times \mathbf{Z}, +$	$\mathbf{Z}_N$	$(\ell, -\ell s)$ $\ell, s \in X$	$f(x_1, x_2) =$ $\pi(sx_1 + x_2 \bmod N)$ π = permutation on X
Abelian stabilizer	(H, X) H = any Abelian group	any finite set	$\{s \in H \mid f(s, x) = x, \forall x \in X\}$	$f(gh, x) = f(g, f(h, x))$ $f(gs, x) = f(g, x)$

Figure 1: Adapted from [1]. List of typical algorithms with exponential speedups. All of them deal with *hidden subgroup problems*, where the function f maps from the group G to the finite set X, and is promised to be constant on cosets of the hidden subgroup K. Z_N represents the set $(0, 1, ..., N-1)$ in this table, and Z is the integers. The solved problem is to find K, given a black box function f.

5. If the algorithm failed, another a is chosen and the algorithm is restarted.

We detail below the two non-trivial steps: period finding and classical post-processing.

2.2 Period finding

We provide an intuition of why $a^r \equiv 1[N]$ is always true when $\gcd(a, N) = 1$. If $\gcd(a, N) = 1$, then $\gcd(a^j, N) = 1$ for any value of j. This means that $a^j \equiv l\,[N]$ with $l \neq 0$. In particular, the space defining all the possible a^j is infinite, whereas the space of l is finite (contains $N-2$ possible values). This means that there exist $i > j$ for which $a^i \equiv a^j\,[N]$. This can be rewritten as: $a^j(a^{i-j} - 1) \equiv 0\,[N]$. Since $\gcd(a^j, N) = 1$, the a^j term can be dropped from the congruence. We therefore obtain $(a^{i-j} - 1) \equiv 0\,[N]$, and by stating $i - j = r$, we get $a^r \equiv 1\,[N]$. This result is rigorously demonstrated in Euler's theorem.

Let's observe what period finding means in practice, and get an intuition as to why it helps finding the prime factors. Assuming $N = 15$ and $a = 2$, we compute the powers of a modulo N:

$$
\begin{aligned}
2^0 &\equiv 1\,[15] \\
2^1 &\equiv 2\,[15] \\
2^2 &\equiv 4\,[15] \\
2^3 &\equiv 8\,[15] \\
2^4 &\equiv 1\,[15] \\
2^5 &\equiv 2\,[15] \\
2^6 &\equiv 4\,[15] \\
2^7 &\equiv 8\,[15] \\
2^8 &\equiv 1\,[15]
\end{aligned}
$$

We observe that the result is *periodic*! The period is here $r = 4$, for which we have $2^4 \equiv 1\,[15]$. Since for any non trivial $[a, N]$, $a^0 = 1 \equiv 1\,[N]$, the first power of a for which one gets $a^r \equiv 1\,[N]$ sets the period. The fact that we obtain a periodic result can be understood by computing $a^{2r} = a^r \times a^r \equiv 1 \times 1\,[N] \equiv 1\,[N]$. Euler rigorously demonstrated that this phenomenon happens for any $[a, N]$ as long as $\gcd(a, N) = 1$.

This order finding task is in the end pretty simple, and is guaranteed to converge thanks to Euler's theorem: one just needs to compute every successive powers of a until finding $a^r \equiv 1\,[N]$. However, the number of steps is *proportional* to N. This can be understood by the fact that, in order to find r, one needs to compute a^r with $r > 0$ exactly r times. The exact value of r depends on a, but it can be demonstrated that r increases linearly with N.

2.3 Classical post-processing

Assuming that the quantum processor found r for which $a^r \equiv 1\,[N]$, we can rewrite $(a^{r/2}-1)(a^{r/2}+1) \equiv 0\,[N]$, meaning that $(a^{r/2}-1)(a^{r/2}+1)$ is divisible by N, which means that $(a^{r/2}-1)(a^{r/2}+1)$ is divisible by p and q. Because p and q are prime numbers, p and q are divisible either by $a^{r/2}-1$, or $a^{r/2}+1$ (known as Euclid's lemma). This last point is true only because p and q are prime numbers and $N = pq$. For generic N Shor's algorithm still works, even though more steps are required.

This means that:

$$a^{r/2} \pm 1 \equiv 0\,[p] \tag{1}$$

$$a^{r/2} \pm 1 \equiv 0\,[q] \tag{2}$$

where the $\pm$ here accounts for the situation described above: p is divisible by $a^{r/2}-1$ or $a^{r/2}+1$. There are then four possibilities, out of which only two allow to find the prime numbers:

$$a^{r/2}+1 \equiv 0\,[p] \text{ and } a^{r/2}+1 \equiv 0\,[q] \rightarrow \text{failure} \tag{3}$$

$$a^{r/2}+1 \equiv 0\,[p] \text{ and } a^{r/2}-1 \equiv 0\,[q] \rightarrow \text{success} \tag{4}$$

$$a^{r/2}-1 \equiv 0\,[p] \text{ and } a^{r/2}+1 \equiv 0\,[q] \rightarrow \text{success} \tag{5}$$

$$a^{r/2}-1 \equiv 0\,[p] \text{ and } a^{r/2}-1 \equiv 0\,[q] \rightarrow \text{failure} \tag{6}$$

$$\tag{7}$$

We describe these two situations:

- **Success situation** Assuming we are in the second situation described above, $a^{r/2}+1$ divides p but not q. This means that $\gcd(a^{r/2}+1, N) = p$, and $\gcd(a^{r/2}-1, N) = q$.
- **Failure situation** In the failure situations, $a^{r/2} \pm 1$ divides both p and q, thus divides N, and therefore $\gcd(a^{r/2} \pm 1, N) = 1$ or N: we did not find the prime factors. The failure situation happens when $a^{r/2} \equiv \pm 1\,[N]$.

The strength of Shor's algorithm relies on finding the correct r for which $a^{r/2} \pm 1$ divides only one of the prime factors, which happens when:

1. $r/2$ is an integer (r is even)
2. $a^{r/2} \not\equiv \pm 1\,[N]$

These two conditions only rely on the initial choice of a.

We can wonder what is the probability for these two conditions to be valid for a given value of a. It can be demonstrated that $a^{r/2} \not\equiv \pm 1\,[N]$ happens with probability $\geq 1/2$, and r being even happens with probability $\geq 1/2$. Combined, we obtain a typical probability for Shor's algorithm to be successful $\geq 1/4$. Crucially, the success probability does not depend on N, and therefore on the complexity theory point of view, these failures cases have a negligible impact. In practice, this means that the quantum processor will need to compute $a^r \equiv 1\,[N]$ a few times before the algorithm finds the prime numbers.

2.4 Classical part of Shor algorithm on a simple example

We take an example to illustrate the key concepts described above. We try to factorize into prime numbers $N = 21$, where $p = 3$ and $q = 7$.

We first illustrate a success situation. We choose $a = 2$, and find that $2^6 = 64 \equiv 1\,[21]$. Thus $r = 6$ and:

$$a^{r/2} = 8 \equiv 3\,[21] \tag{8}$$

meaning that we are in a success situation. We compute the two quantities which divide p and q:

$$a^{r/2} - 1 = 7 \tag{9}$$

$$a^{r/2} + 1 = 9 \tag{10}$$

In this situation, $a^{r/2} - 1$ only divides q and not p: this is why the algorithm works. We then find the prime numbers:

$$\gcd(a^{r/2} + 1, N) = 3 \tag{11}$$

$$\gcd(a^{r/2} - 1, N) = 7 \tag{12}$$

We now illustrate a failure situation. We choose $a = 20$, and try to find $a^r \equiv 1[21]$:

$$20^1 = 20 \equiv -1\,[21] \tag{13}$$

$$20^2 = 400 \equiv 1\,[21] \tag{14}$$

So the order is $r = 2$, which is even. However, $20^1 \equiv -1\,[21]$, meaning that we expect to be in a failure situation. We confirm this by computing:

$$a^{r/2} - 1 = 19 \tag{15}$$

$$a^{r/2} + 1 = 21 \tag{16}$$

and find:

$$\gcd(a^{r/2} + 1, N) = 1 \tag{17}$$

$$\gcd(a^{r/2} - 1, N) = 21 \tag{18}$$

as expected.

Exercise 1. Apply Shor's algorithm to $N = 15$, using $a = 2$, following the same structure as what is done above.

3 The quantum part of Shor's algorithm

Having described the classical part of Shor's algorithm, we now discuss the part where the quantum processor is used. As previously discussed, for a given $1 < a < N$, the quantum processor is able to find r such that: $a^r \equiv 1[N]$.

In the following, we will perform a very important basis change to describe the quantum states which are used throughout Shor's algorithm, introduced in Lecture 1, Section 4. We introduce the notation $|x\rangle$, where $x = \sum_{j=0}^{2^n-1} x_j 2^j$, $x_j \in 0, 1$ is defined as being an integer that is described in the binary basis. For example, for $x = 5$, $|5\rangle \equiv |101\rangle$.

3.1 Quantum algorithm description

We first give an overview of what the quantum processor does, and next dive into explaining each key parts. The algorithm uses two quantum registers:

- **The control register**, which is composed of n qubits. It is in this register that the information about r will be extracted.
- **The work register**, which is composed of m qubits. It is used to enable r to be extracted from the control register.

The quantum operations are performed as follows:

1. **Initialization.**, Starting from the initial state $|0\rangle^{\otimes n}|0\rangle^{\otimes m}$, apply $H^{\otimes n}$ on the control register, and one X gate on the m^{th} qubit in the work register:

$$|0\rangle^{\otimes n}|0\rangle^{\otimes m} \rightarrow \frac{1}{\sqrt{2^n}}[(|0\rangle+|1\rangle)\otimes \ldots \otimes(|0\rangle+|1\rangle)]\otimes|00\ldots1\rangle = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\otimes|1\rangle \quad (19)$$

   where we now use the binary basis to express the states of the two registers. We here use the superposition to "test" all the possible numbers at the same time.

2. **Modular exponentiation function (MEF).** Conditionally apply the unitary operation U that implements the modular exponentiation function $a^x[N]$ on the work register whenever the control register is in state $|x\rangle$:

$$\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\otimes|1\rangle \rightarrow \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle|a^x[N]\rangle \quad (20)$$

   This steps generates entanglement between the two registers. As $a^x[N] < N$, this number can be well represented if $2^m \geq N$, and therefore the number of qubits that are required in the work register are $m \geq \log_2(N)$.

3. **Projective measurement on the work register.** We note that this step is not necessary in practice, but eases a lot the understanding of the algorithm. Assuming that the work register is projected onto $|a^x \equiv b[N]\rangle$, we obtain:

$$\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle|a^x[N]\rangle \rightarrow \frac{1}{\sqrt{M}}\sum_{k=0}^{M-1}|x_0+kr\rangle \quad (21)$$

   The sum runs over all x values for which $|a^x \equiv b[N]\rangle$. Since $a^{x+r} \equiv a^x[N]$, we obtain that the sum runs over the states which are periodic in r, and that can be written in the form $|x_0+kr\rangle$ with k an integer. The sum runs over M which represents all the possible states within the control register (2^n states) that satisfy $x = x_0 + kr$. As $2^n \gg r$ and $x_0 < r$, the number of possible states is $M \simeq 2^n/r$. Thanks to the entanglement between the two registers, the states in the control register are now encoding r! This means that if we would measure the control register many times, we would obtain various x values, where the spacing between

these obtained x values would tell us the value of r. However, we would need to sample N times the wavefunction, and therefore we would not get any speedup as compared to classical period finding. An important remark that we can make here is that r is encoded in the *periodicity* of states. In order to make the algorithm efficient, we need to find an operation that converts this periodicity in the state domain into frequency peaks in an equivalent "state-frequency" domain. Thankfully, such operation exists, is unique to quantum mechanics (as it requires the ability to make the states interfere), and is called the quantum Fourier transform.

4. **Quantum Fourier Transform (QFT).** As for classical signals, the QFT enables to convert periodicity of states into "state-frequency" peaks:

$$\begin{aligned}\frac{1}{\sqrt{M}}\sum_{k=0}^{M-1}|x_0+kr\rangle \rightarrow &\frac{1}{\sqrt{2^nM}}\sum_{k=0}^{M-1}\sum_{y=0}^{2^n-1}e^{2i\pi(x_0+kr)y/2^n}|y\rangle \\ =&\frac{1}{\sqrt{2^nM}}\sum_{y=0}^{2^n-1}e^{2i\pi x_0y/2^n}\sum_{k=0}^{M-1}e^{2i\pi kry/2^n}|y\rangle\end{aligned}$$

After QFT, the 2^n states are present in the wavefunction again (as after the initialization). While previously the $|x\rangle$ states were encoding all the possible integers from 0 to N, the $|y\rangle$ states now encode the frequencies that were stored into the coherent superposition of the $|x\rangle$ states. In particular, as in classical Fourier transform, the pre-factor in front of the $|y\rangle$ states is representative of the strength of the periodicity y within the superposition before QFT. Since the state before QFT is a superposition of states $|x_0+kr\rangle$, we can qualitatively understand that the frequency r should get out of the QFT. This can be seen in the above expression: the inner sum running over k is a finite geometric series, and is therefore equal to M only if $s=ry/2^n$ is an integer. This means that the states $|y\rangle$ with high amplitude will be the ones for which $y=s2^n/r$, with s an integer. Since y is bounded by 2^n-1, the maximum value of s is $r-1$. We can therefore re-write the output wavefunction considering only these states:

$$\begin{aligned}\frac{1}{\sqrt{2^nM}}\sum_{y=0}^{2^n-1}e^{2i\pi x_0y/2^n}\sum_{k=0}^{M-1}e^{2i\pi kry/2^n}|y\rangle \simeq &\frac{M}{\sqrt{2^nM}}\sum_{s=0}^{r-1}e^{2i\pi x_0s/r}|s2^n/r\rangle \\ \simeq &\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}e^{2i\pi x_0s/r}|s2^n/r\rangle\end{aligned}$$

where we used $M\simeq 2^n/r$. We have here assumed that $s2^n/r$ is always an integer, which simplifies the calculation of the wavefunction. This is in practice not true (and therefore the geometric sum does not exactly equal to M). The selected states by the QFT are therefore $|y\rangle=|\text{round}(s2^n/r)\rangle$, and the closer $s2^n/r$ is from an integer, the larger the amplitude is.

5. **Measurement of the control register and continued fraction expansion.** The quantum processor therefore outputs a state $|y\rangle=|\text{round}(s2^n/r)\rangle$ with $0\leq$

$s \leq r-1$. This means that we obtain a result close to the ratio $y/2^n = s/r$ in which both s and r are unknown: a last step is required to find r... meaning that we do not get r directly from the quantum processor. First, the output state is not exactly $s2^n/r$. However, as $|y - s2^n/r| \leq 1/2$, we obtain that $|y/2^n - s/r| \leq 1/2^{n+1}$. By choosing a sufficiently large n, the rounding has a negligible impact on finding r. Second, we need to compute r from knowing $y/2^n \simeq s/r$ with an unknown (but bounded) s. In order to tackle this issue, the key concept is *not* to look for the value of s, but rather to observe that we have the ratio of two integers (y and 2^n) which is (almost) equal to the ratio of two other integers (s and r). Setting $a = y/2^n$, we perform the *continued fraction expansion* of a, which takes the form:

$$a = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}} \tag{22}$$

where a_i are the expansion terms of x. It can be demonstrated that if (1) a satisfies $|a - s/r| < 1/(2r^2)$ and (2) $\gcd(s, r) = 1$, then it is guaranteed that the continuous fraction expansion of a will at some point be equal to s/r. As discussed above, we know that $|a - s/r| \leq 1/2^{n+1}$. This means that we need $2^n > r^2$, r is unknown but $r < N$, and therefore we need $2^n > N^2$. This last point sets the number of qubits that are required in the control register: we need $n > 2\log_2(N)$ qubits. The condition $\gcd(s, r) = 1$ has typically a probability of $1/2$ to be true, and therefore the quantum part of Shor's algorithm can also fail (the probability does not depend on N, and in practice even if $\gcd(s, r) \neq 1$ there is still a chance to find r). By successively checking the obtained fraction each time we add a term and verifying if $a^r \equiv 1\,[N]$, we are sure to obtain the correct r.

From the above results, we can bound the number of qubits that are required in order for Shor to function: $n > 2\log_2(N)$ in the control register and $m \geq \log_2(N)$ in the work register.

At the quantum circuit level, the algorithm has only 3 steps, described in Figure 2:

1. A global Hadamard gate on the control register to prepare a superposition of all possible states.

2. The modular exponentiation function, implemented via quantum addition, which entangles the control and the work register, enabling to highlight states with period r in the control register.

3. An inverse quantum Fourier Transform (QFT) applied to the control register, enabling to convert this periodicity of states into states that directly encode r within the states.

These 3 quantum operations are found in many applications. We next describe how to implement the modular exponentiation via quantum addition and the QFT.

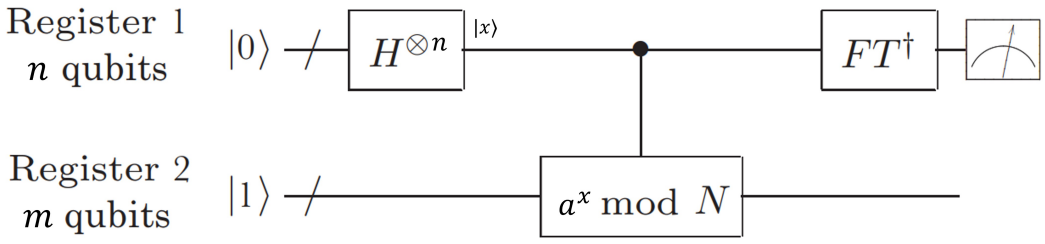

Figure 2: Adapted from [1]. High-level quantum circuit to perform the period finding: a global Hadamard gate, the modular exponentiation, and the inverse quantum fourier transform

3.2 Shor algorithm on an example

We now exemplify the algorithm on an example which explicitly shows the impact of each step. We want to factorize $N = 21$ using $a = 2$. As previously discussed, we know that we should find $r = 6$.

Since we know r, we can adapt the size of the control register in order to match the condition $2^n > r^2$. We choose to work with 6 qubits in the control register.

1. **Initialization.** We first apply the global Hadamard on the control register and one X gate on the work register:

$$|0\rangle^{\otimes 6} |0\rangle^{\otimes m} \rightarrow \frac{1}{\sqrt{64}} \sum_{x=0}^{63} |x\rangle |1\rangle \tag{23}$$

2. **Modular exponentiation function (MEF).** We then apply the modular exponentiation function to get $|\psi_{\text{MEF}}\rangle$:

$$\frac{1}{\sqrt{64}} \sum_{x=0}^{63} |x\rangle \otimes |1\rangle \rightarrow \frac{1}{\sqrt{64}} \sum_{x=0}^{63} |x\rangle |2^x \, [21]\rangle$$

$$\begin{aligned} |\psi_{\text{MEF}}\rangle = \frac{1}{\sqrt{64}} [&|0\rangle |1\rangle + |1\rangle |2\rangle + |2\rangle |4\rangle + |3\rangle |8\rangle \\ &+ |4\rangle |16\rangle + |5\rangle |11\rangle + |6\rangle |1\rangle + |7\rangle |2\rangle + |8\rangle |4\rangle + ...] \end{aligned}$$

We see the $2^x \, [21]$ terms appearing in the work register, associated to x in the control register. We observe the 6-periodicity in the work register, which means that we can factorize them and rewrite the state as:

$$\begin{aligned} |\psi_{\text{MEF}}\rangle = \frac{1}{\sqrt{64}} [&(|0\rangle + |6\rangle + |12\rangle + |18\rangle + |24\rangle + ...) \otimes |1\rangle \\ &+ (|1\rangle + |7\rangle + |13\rangle + |19\rangle + |25\rangle + ...) \otimes |2\rangle + ...] \end{aligned}$$

We now see more clearly the period appearing in the control register.

3. **Projective measurement on the work register.** We apply the measurement on the work register. The measurement will select one of the possible work state, but we see that this only shifts the control register by a constant value which does not matter for finding the period, hence this measurement is not necessary. For the sake of simplicity we will here perform this measurement, assuming we measure $|1\rangle$ in the work register:

$$\begin{aligned}|\psi_c\rangle &= \frac{1}{\sqrt{11}}(|0\rangle + |6\rangle + |12\rangle + |18\rangle + |24\rangle + |30\rangle + |36\rangle + |42\rangle + |48\rangle + |54\rangle + |60\rangle) \\ &= \frac{1}{\sqrt{11}}\sum_{k=0}^{10}|6k\rangle\end{aligned}$$

As previously discussed, we could directly sample from this state. However, we have here 11 possible states, meaning that we would need to perform the quantum circuit many times to extract the correct period (in reality you do not know what is r, so if you sample, you need to make sure you're not missing a state...).

4. **Quantum Fourier Transform (QFT).** We perform the QFT on $|\psi_c\rangle$:

$$|\psi_{\text{QFT}}\rangle = \frac{1}{\sqrt{11}}\sum_{y=0}^{63}\frac{1}{\sqrt{64}}\sum_{k=0}^{10}e^{i2\pi 6ky/64}|y\rangle = \frac{1}{\sqrt{11\cdot 64}}\sum_{y=0}^{63}\frac{1-e^{2i\pi 66y/64}}{1-e^{2i\pi 6y/64}}|y\rangle \tag{24}$$

Where we used the geometric sum property $\sum_k^N \omega^k = (1-\omega^N)/(1-\omega)$. The geometric series value peaks for $6y/64 = s$ being an integer, meaning for $y \simeq s \times 10.667$. We observe that the peak value is never obtained for y being an integer, meaning that the inner sum never gets a maximum value (except $y = 0$ where the sum equals 11). The state can be rewritten as:

$$|\psi_{\text{QFT}}\rangle = \frac{1}{\sqrt{11\cdot 63}}[11\,|0\rangle + \frac{1-e^{2i\pi 66/64}}{1-e^{2i\pi 6/64}}|1\rangle + \frac{1-e^{2i\pi 132/64}}{1-e^{2i\pi 12/64}}|2\rangle + ...]$$

The largest terms happen for $y \simeq s \times 10.667$, which gives $y = 0, 11, 21, 32, 43, 53$ and we therefore simplify the wavefunction:

$$|\psi_{\text{QFT}}\rangle \simeq \frac{1}{\sqrt{11\cdot 63}}[11\,|0\rangle + \frac{1-e^{2i\pi 66\cdot 11/64}}{1-e^{2i\pi 6\cdot 11/64}}|11\rangle + \frac{1-e^{2i\pi 66\cdot 21/64}}{1-e^{2i\pi 6\cdot 21/64}}|21\rangle + ...]$$

We now obtain 6 states in $|\psi_{\text{QFT}}\rangle$, which are not the same as prior the QFT.

5. **Measurement of the control register and continued fraction expansion.** Upon applying measurement, we get the following probability densities:

$$\begin{aligned}|\langle 0\,|\psi_{\text{QFT}}\rangle|^2 &= 11/63 \simeq 0.175 \\ |\langle 11\,|\psi_{\text{QFT}}\rangle|^2 &\simeq 0.012 \\ |\langle 21\,|\psi_{\text{QFT}}\rangle|^2 &\simeq 0.145 \\ |\langle 32\,|\psi_{\text{QFT}}\rangle|^2 &\simeq 0.175 \\ |\langle 43\,|\psi_{\text{QFT}}\rangle|^2 &\simeq 0.145 \\ |\langle 53\,|\psi_{\text{QFT}}\rangle|^2 &\simeq 0.012\end{aligned}$$

We observe that the most probable state which is interesting is $|32\rangle$, for which the geometric series value is maximal. Let's assume that the processor outputs $|11\rangle$ (highly unlikely, but can still happen). We then compute the ratio $a = 11/64 \simeq 0.172$, which is not far away from $1/6$ with $s = 1$ and $r = 6$. We compute the continued fraction expansion:

$$a = 0 + \frac{1}{5 + \frac{1}{1 + \frac{1}{4 + \frac{1}{2}}}} \tag{25}$$

We can look at the value of a of the successive expansion orders k, look at the denominator of the fraction, and check if $2^r \equiv 1\,[21]$:

Order 1: $a = \frac{1}{5} \rightarrow r = 5 \rightarrow 2^5 \equiv 11\,[21] \rightarrow$ Not the correct value of r

Order 2: $a = \frac{1}{5 + \frac{1}{1}} = \frac{1}{6} \rightarrow r = 6 \rightarrow 2^6 \equiv 1\,[21] \rightarrow$ We found the value of r !

Order 3: $a = \frac{1}{5 + \frac{1}{1 + \frac{1}{4}}} = \frac{5}{29}$

Exercise 2. Apply Shor's algorithm to $N = 15$, using $a = 2$, following the same structure as what is done above.

3.3 Modular exponentiation using quantum addition

We here focus on detailing the modular exponentiation. As described above, we look for an unitary U which performs the following:

$$U : |x\rangle\,|1\rangle \rightarrow |x\rangle\,|a^x\,[N]\rangle \tag{26}$$

The core sub-routine which is used in this part of the algorithm is *quantum addition*. We first describe this operation, then explain how it is used in the modular exponentiation.

3.3.1 Quantum addition

Addition can be seen as the most basic computing capability, and it is therefore natural to introduce how it works in a quantum computer. As in classical computing, addition will be performed in the binary basis. Assuming two states $|y\rangle = |y_{m-1}...y_0\rangle$ and $|r\rangle = |r_{m-1}...r_0\rangle$, we will describe the unitary that performs the transformation:

$$|y\rangle\,|r\rangle \rightarrow |y\rangle\,|r + y\rangle \tag{27}$$

meaning that we add two quantum states, and store the sum on the second state. The quantum adder relies on quantum gates, in a similar fashion to classical addition. The

used method is similar to the method we learned when we were kids to perform additions. Suppose I want to add (in binary basis) $|y\rangle = |110\rangle$ and $|r\rangle = |001\rangle$:

$$\begin{aligned} &110 \\ +&001 \\ =&111 \end{aligned}$$

where we add each qubit composing y and r successively. Therefore, in order to obtain a *quantum adder*, we engineer an operation such that, on each qubit :

$$\begin{aligned} y_i|r_i &\rightarrow y_i + r_i \\ 0|0 &\rightarrow 0 \\ 0|1 &\rightarrow 1 \\ 1|0 &\rightarrow 1 \\ 1|1 &\rightarrow 0 \end{aligned}$$

This situation is obtained when we implement a CNOT gate: the state of the qubit $|r_i\rangle$ is flipped conditioned on $|y_i\rangle = 1$. We will use the notation CNOT$(y_i \rightarrow r_i)$ to describe a CNOT where $|y_i\rangle$ is the control qubit, and $|r_i\rangle$ is the target qubit.

There is still one situation that we need to handle: if the two bits are 1, then we have to report a carry-over to the next bit. In order to perform such operation in a unitary fashion, we need to add an extra carry-over state $|c\rangle = |c_{n-1}...c_0\rangle$ which will store the value of the carry-over. Assuming that $c_{i-1} = 0$, the situation we want to end up in is therefore:

$$\begin{aligned} y_i|r_i &\rightarrow c_i \\ 0|0 &\rightarrow 0 \\ 0|1 &\rightarrow 0 \\ 1|0 &\rightarrow 0 \\ 1|1 &\rightarrow 1 \end{aligned}$$

we therefore want $|c_i\rangle = 1$ when $|y_i\rangle = |r_i\rangle = 1$: this is a Toffoli gate, and we use the notation Toffoli$(y_i, r_i \rightarrow c_i)$ to describe it. In order to add the carry over to the sum, we need to perform the operation $y_i + r_i + c_i$, which is done by applying a CNOT$(y_i \rightarrow r_i)$ (as described above), and CNOT$(c_i \rightarrow r_i)$ (adds the carry-over).

In the above situation, we assumed that $c_{i-1} = 0$. In order to take into account the two possible values of c_{i-1}, we need to add 2 Toffoli gates: Toffoli$(y_i, c_{i-1} \rightarrow c_i)$ and Toffoli$(r_i, c_{i-1} \rightarrow c_i)$. Therefore, performing the addition for one row of qubit $|y_i\rangle, |r_i\rangle, |c_i\rangle$ requires 2 CNOT and 3 Toffoli gates, except for the first addition where only 1 Toffoli and 1 CNOT are required, as there is no carried-out value. The circuit is

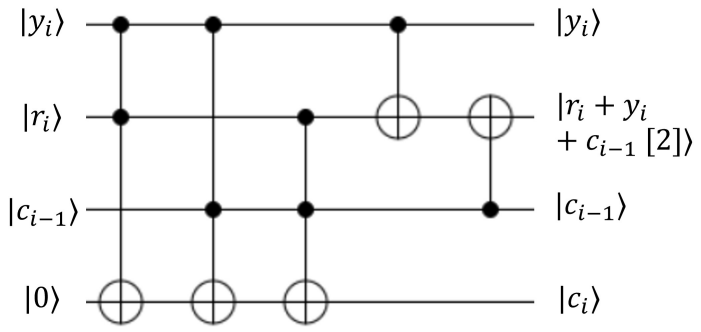

Figure 3: i^{th} iteration of one possible circuit to perform the addition $|y\rangle\,|r\rangle \to |y\rangle\,|r+y\rangle$

summarized in Fifure 3.

In general, at the end of the quantum addition, $|c\rangle$ is still entangled with $|y\rangle$ and $|y+r\rangle$. Therefore, $|c\rangle$ can have a deleterious impact on any further usage of these states (if $|c\rangle$ is measured for example). Therefore, it is necessary to disentangle the carrier state from the other states. To do so, one simply needs to apply the gates acting on $|c\rangle$ in the inverse order (assume $A = BC$ hermitian, then $A^{-1} = CB$). This amounts to $\sim 3m$ Toffoli gates.

Following the protocol that we described, a quantum addition between two states with m qubits is therefore composed of:

- $\sim 3m$ qubits ($|y\rangle$, $|r\rangle$ and $|c\rangle$)
- $\sim 2m$ CNOT gates
- $\sim 6m$ Toffoli gates

We note that if we invert the role of 0 and 1 in all controlled operations (called anti-controled operations), meaning that we perform anti-controlled NOT gates and anti-Toffoli, we obtain a quantum substractor instead of a quantum adder. We also note that there are other protocols to perform addition, which will rebalance the requirements in the number of gates between CNOT and Toffoli.

3.3.2 From exponentiation to multiplication

As described above, we look for an unitary U which performs the following:

$$U : |x\rangle\,|1\rangle \to |x\rangle\,|a^x\ [N]\rangle \tag{28}$$

x can be rewritten into its binary basis as:

$$x = \sum_{k=0}^{n-1} x_k 2^k \tag{29}$$

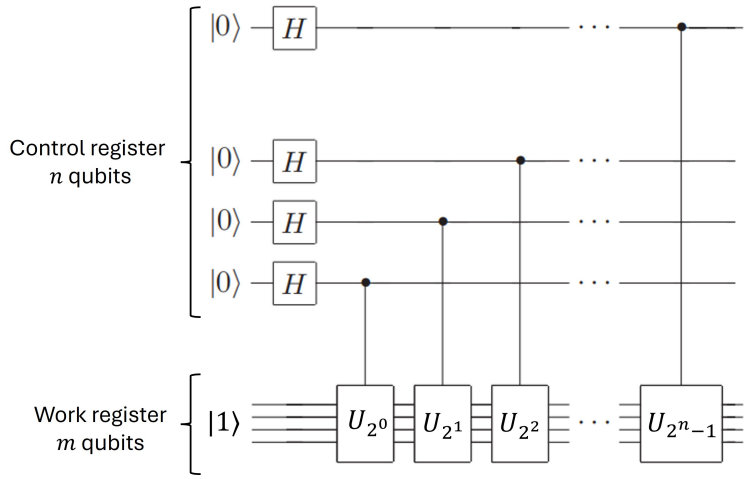

Figure 4: Adapted from [1]. High-level quantum circuit showing the global Hadamard gate, and the decomposition of the modular exponentiation into consecutive modular controled multiplications $c{-}U_{a^{2^k}}$

where n is the number of qubits in the control register. Following this, we can rewrite $a^x\,[N]$ as:

$$a^x = a^{\sum_k x_k 2^k} = \prod_{k=0}^{n-1} a^{2^k x_k}\,[N] = \prod_{k=0}^{n-1} (a^{2^k}\,[N])^{x_k}\,[N], \tag{30}$$

where we used the fact that $ab\,[N] = (a\,[N]\,b\,[N])\,[N]$. This means that in order to implement $a^x\,[N]$ on the work register, one needs to *multiply* n times the value stored in the work register by $a^{2^k}\,[N]$, *conditioned* on the value of x_k. Here, x_k is the value of the qubit in the control register at position k (either 0 or 1). We introduce the unitary c-$U_{a^{2^k}}$ with $0 \leq k < n$, which performs the following:

- if $x_k = 0$, no multiplication, applies identity to the work register,
- if $x_k = 1$, multiply the work register by $a^{2^k}\,[N]$

This unitary c-$U_{a^{2^k}}$ is similar to a CNOT gate, but applies $U_{a^{2^k}}$ instead of an X gate. We can express U as:

$$U = \prod_{k=0}^{n-1} \text{c-}U_{a^{2^k}} \text{ with } U_{a^{2^k}}\,|y\rangle = |y a^{2^k}\,[N]\rangle$$

Using these mathematic tricks, we are now in a situation where instead of having to apply the modular exponentiation U, we need to apply a *product of controlled modular multiplications*. The values a^{2^k} are computed classically before the execution of the algorithm. Each c-$U_{a^{2^k}}$ is controlled by one qubit of the control register x_k (hence the name of this register!), and acts on the entire work register, as displayed in Figure 4.

3.3.3 From multiplication to addition

Since a and a^{2^k} are integers, $U_{a^{2^k}}$ could be directly computed using the addition described above. However, this would be sub-optimal with respect to the technique described now. We can decompose y (m qubits) into its binary basis:

$$y = \sum_{i=0}^{m-1} y_i 2^i \rightarrow y a^{2^k} = \sum_{i=0}^{m-1} y_i 2^i a^{2^k} \tag{31}$$

meaning that $U_{a^{2^k}}$ can be performed by applying a succession of *controlled modular additions* conditioned on the value of y_k, in the exact same fashion as previously. Since the addition is controlled on $|y\rangle$, we cannot directly perform the addition on the work register $|y\rangle$ itself. We therefore need an extra register $|r\rangle$, called the *accumulation register*, in which we store the addition. We are therefore performing the operation U_+ where:

- if $y_k = 0$, no addition, applies identity to the work register
- if $y_k = 1$, add to the accumulator register a^{2^k} $[N]$

We detail how $U_{a^{2^k}}$ is implemented:

- The addition is performed by the adder protocol detailed above
- The controlled part is added by applying a control on y_i for the first gate in the adder protocol. If $y_i = 0$, performing this effectively cancels any addition that would occur to the accumulation register.
- The modulo N operation is applied after each addition, in order to prevent the necessity to store a state which is much larger than N.
- After all additions have been performed, the state in the accumulator register is $|r\rangle = |y a^{2^k}\rangle$
- This state is then transferred into the work register using for example a SWAP gate (see Lecture 1).

At the end of step k, the system is in the state $|x\rangle\,|a^{x_1 2 + x_2 4 + x_3 8 + \ldots + x_k 2^k}\rangle$. We then repeat this protocol for $0 \leq k < n$, and eventually implement the modular exponentiation U!

3.3.4 Applying the modulo N

There are various solutions for the modulo N operation, with different tradeoff in number of operations or qubit numbers. In all solutions, the operation is performed in 3 parts:

1. Some ancillary qubits check whether the value in the accumulation register is larger or lower than N (a sub-routine called quantum comparator)

2. This information is transferred to a flag qubit, which is in $|0\rangle$ if $r < N$, and in $|1\rangle$ is $r \geq N$ (in some solutions, the flag qubit is the same at the ancillary qubit described in the previous operation).

3. A quantum substraction (the "anticontrolled" protocol than the one performed in the quantum addition) of value N is applied, conditioned on the flag qubit.

These protocols work because they are performed at each addition, and therefore $r < 2N$.

3.3.5 Conclusion

We have here described the modular exponentiation function. We wrap-up the key aspects:

- The modular exponentiation function can be converted into a product of modular multiplications over the n bits that compose $|x\rangle$, controlled by the value of x_i
- The modular multiplications can be implemented via successive modular additions controlled by the value of y_i over the m bits that compose $|y\rangle$. The result of the addition is stored in an accumulation register $|r\rangle$ of size m.
- The modular additions are composed of a quantum adder routine followed by a verification protocol which substracts N to r if $r > N$.

We also wrap up the cost of the modular exponent function:

- The number of qubits is at least equal $n + 3m$: n qubits in the control register, m qubits in the work register, m qubits in the accumulation register, and m qubits for the carry-over in the additions. Depending on the method for the quantum comparator, an extra m qubits can be required.
- The number of Toffoli gates is at least equal to $26nm^2$. An addition as we did it is $6m$ Toffoli. The quantum comparison is equivalent to performing two additions, and the conditioned subtraction converts the CNOT gates into Toffoli. The modular addition is performed m times, and the multiplication n times.
- As previously discussed, in order for Shor to work, one needs $n > 2\log_2(N)$ and $m \geq \log_2(N)$. This means that the total number of qubits is about $6\log_2(N)$ (we took the m extra qubits here), and the number of Toffoli gates for the modular exponentiation function is $52\log_2(N)^3$. Note that these requirements depend on the protocol used for the addition and application of the modulo, but the scaling with N remains the same.

4 The quantum Fourier transform

We describe the last part of Shor's algorithm: the quantum Fourier transform. As previously discussed, the quantum Fourier transform is an efficient quantum algorithm

for performing a Fourier transform of quantum mechanical amplitudes. It does not speed up the classical task of computing Fourier transforms of classical data. But one important task which it does enable is phase estimation, the approximation of the eigenvalues of a unitary operator under certain circumstances. As reminder, assuming a state $|j\rangle$ composed of n qubits, we are looking for a unitary operation U_{QFT} which performs:

$$|j\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2i\pi jk/2^n} |k\rangle \tag{32}$$

This operation is equivalent to the classical discrete Fourier transform of an input signal $x_0, ..., x_{2^n-1}$ into values $y_1, ..., y_{2^n}$ defined as:

$$y_k \equiv \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} x_j e^{2i\pi jk/2^n} \tag{33}$$

In order to implement such unitary, we rewrite the operation of the QFT on $|j\rangle$, remembering that the states can be represented by the qubit $|x\rangle = |x_{2^n}...x_1\rangle$:

$$\begin{aligned}
|j\rangle \rightarrow & \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2i\pi jk/2^n} |k\rangle \\
= & \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} ... \sum_{k_{2^n}=0}^{1} e^{2i\pi j(\sum_{l=1}^{n} k_l 2^{-l})} |k_1..k_{2^n}\rangle \\
= & \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} ... \sum_{k_{2^n}=0}^{1} \bigotimes_{l=1}^{2^n} e^{2i\pi jk_l 2^{-l}} |k_l\rangle \\
= & \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \sum_{k_l=0}^{1} e^{2i\pi jk_l 2^{-l}} \\
= & \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} (|0\rangle + e^{2i\pi j 2^{-l}} |1\rangle) \\
= & \frac{(|0\rangle + e^{2i\pi 0.j_n} |1\rangle)(|0\rangle + e^{2i\pi 0.j_{n-1}j_n} |1\rangle)...(|0\rangle + e^{2i\pi 0.j_1 j_2 ... j_n} |1\rangle)}{\sqrt{2^n}}
\end{aligned}$$

where we introduced the notation for the binary fraction $0.j_l j_{l+1}...j_m = j_l/2 + j_{l+1}/4 + ... + j_m/2^{m-l+1}$.

In order to understand how to implement such operation, let's focus on one of the output qubit $|k_l\rangle$. Before the unitary, this qubit is in state $|j_l\rangle$, and after, we get $|k_l\rangle = |0\rangle + e^{2i\pi 0.j_l j_{l+1}...j_n} |1\rangle$. We therefore need to have a unitary which applies a phase gate R_k to the qubit, depending on the phase of the other qubits, with

$$R_k \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{i2\pi/2^k} \end{pmatrix}$$

The circuit that allows to perform the QFT is shown in Figure 5. To see that the pictured circuit computes the quantum Fourier transform, consider what happens when

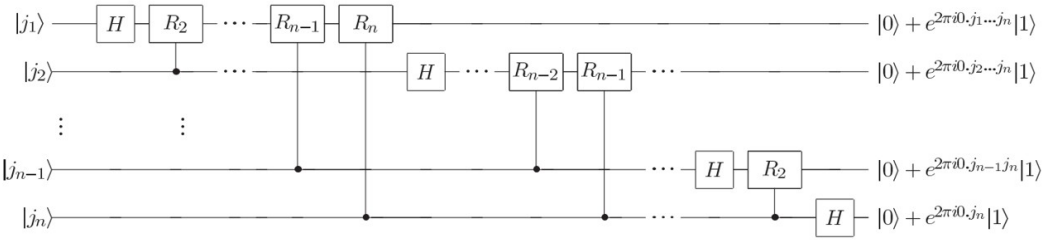

Figure 5: Adapted from [1]. Efficient circuit for the quantum Fourier transform. Not shown are SWAP gates at the end of the circuit which reverse the order of the qubits, or normalization factors of $1/\sqrt{2}$ in the output

the state $|j_1 ... j_n\rangle$ is input. Applying the Hadamard gate to the first bit produces the state:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.j_1}|1\rangle)|j_2 ... j_n\rangle \tag{34}$$

since $e^{2i\pi 0.j_1} = -1$ when $j_1 = 1$, and is $+1$ otherwise. Applying the controlled-R_2 gate produces the state:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.j_1 j_2}|1\rangle)|j_2 ... j_n\rangle \tag{35}$$

We continue applying the controlled-R_3, R_4 through R_n gates, each of which adds an extra bit to the phase of the co-efficient of the first $|1\rangle$. At the end of this procedure we have the state:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.j_1 j_2 ... j_n}|1\rangle)|j_2 ... j_n\rangle \tag{36}$$

Next, we perform a similar procedure on the second qubit. The Hadamard gate puts us in the state

$$\frac{1}{2}(|0\rangle + e^{2i\pi 0.j_1 j_2 ... j_n}|1\rangle)(|0\rangle + e^{2i\pi 0.j_2}|1\rangle)|j_3 ... j_n\rangle \tag{37}$$

and the controlled-R_2 through R_{n-1} gates yield the state:

$$\frac{1}{2}(|0\rangle + e^{2i\pi 0.j_1 j_2 ... j_n}|1\rangle)(|0\rangle + e^{2i\pi 0.j_2 ... j_n}|1\rangle)|j_3 ... j_n\rangle \tag{38}$$

We continue in this fashion for each qubit, giving a final state:

$$\frac{1}{2}(|0\rangle + e^{2i\pi 0.j_1 j_2 ... j_n}|1\rangle)(|0\rangle + e^{2i\pi 0.j_2 ... j_n}|1\rangle)...(|0\rangle + e^{2i\pi 0.j_n}|1\rangle) \tag{39}$$

SWAP operations, omitted from Figure 1 for clarity, are then used to reverse the order of the qubits and obtain the post-QFT state.

Exercise 3. Perform the quantum fourier transform circuit for 3 qubits, and write the corresponding matrix representing the QFT operation.

Having displayed the required circuit to perform a QFT, we can count the number of operations that are required to implement the QFT. We start by doing a Hadamard gate and $n-1$ conditional rotations on the first qubit – a total of $n$gates. This is followed

by a Hadamard gate and $n-2$ conditional rotations on the second qubit, for a total of $n+(n-1)$ gates. Continuing this way, we see that $n+(n-1)+...+1 = n(n+1)/2$ gates are required, plus the gates involved in the swaps. At most $n/2$ swaps are required. Therefore, the QFT requires $\sim 3n + n^2/2 \sim O(n^2)$ gates. Since for Shor to work we need $n > 2\log_2(N)$, we obtain the the number of gates is $\sim 6\log_2(N) + \log_2(N)^2$.

In contrast, the best classical algorithms for computing the discrete Fourier transform on 2^n elements are algorithms such as the Fast Fourier Transform (FFT), which compute the discrete Fourier transform using $O(n2^n)$ gates. That is, it requires exponentially more operations to compute the Fourier transform on a classical computer than it does to implement the quantum Fourier transform on a quantum computer. The QFT is the step that enables to get the *exponential speedup* with respect to classical computing.

5 Conclusion

In this lecture, we have seen how Shor algorithm work.

We first comment on the quantum ressources that are required to perform Shor. In order to factorize a number N, one needs:

- We need $\sim 6\log_2(N)$ qubits: $2\log_2(N)$ in the control register, $\log_2(N)$ in the work register, $3\log_2(N)$ quits to enable the quantum addition and the modulo N operation (the number of qubits required for these operation can vary depending on the exact protocol which is used)
- The number of gates is $\sim 52\log_2(N)^3 + \log_2(N)^2 + 6\log_2(N)^2$, largely dominated by the cost $\sim 52\log_2(N)^3$ of the modular exponentiation.

From these numbers, we could look at when factorizing using Shor enters into a regime of practical quantum advantage. This will be discussed in Lecture 6.

Shor algorithm follows the structure:

- The computationally-simple part of the algorithm is performed on a classical computer
- The NP-hard part of the algorithm is performed on a quantum computer

This structure is present in almost every type of quantum algorithm. The quantum computer is used as a sub-routine of a larger algorithm, and we should not see the quantum computer as a classical computer: the quantum computer is only here to "compute" where it excels, leaving any other tasks to the classical computer. That's why quantum computers will be placed into high-performance computing centers, integrated into classical computers. This consideration pushed people to change the terminology from quantum computer to quantum processing units (also known as QPU), similar to CPU or GPU, as it is sought to be a sub-part of a classical computer.

Shor's algorithm uses many techniques derived from number theory for the classical part. This shows that in order to find new algorithms, researchers should be focused on classical expertise, with an understanding about what quantum computers can do. At the quantum level, the most important feature of the quantum computer that we have discussed are (1) modular exponentiation using quantum addition, and (2) the Quantum Fourier Transform. The quantum addition is relatively similar than classical addition, with the difference that it can handle state superposition. On its own, it does not enable to get speedups: it is the QFT that enables the exponential speedup, by creating the required interferences that enable to obtain the result of Shor algorithm with a single measurement.

References

[1] Nielsen and Chuang, Quantum computation and, Cambridge University Press (2008).