Quantum Algorithms

André Chailloux

contact: andre.chailloux@inria.fr

Chapter 0

Prequel

Quantum algorithms:

- Potentially much faster than classical algorithms.
- examples: Deutsch-Jozsa, Simon, Grover.
- Shor's quantum algorithm for factoring[1]: probably the most important for quantum algorithms (1994).

Factoring Problem
Input: a number N of n bits. Goal: find the decomposition of N in prime numbers $N = \Pi_i p_i^{\alpha_i}$.

Best known classical algorithm : general number field sieve which runs in $O(e^{O(n^{1/3}\log^{2/3}(n))})$ (heuristic complexity)[2]
Shor's quantum algorithm runs in $O(n^2 \log^2(n))$.

- Exponential speedup.
- Breaks RSA based cryptosystems (that's a lot!).

Extends to Discrete logarithm (Diffie Hellman key exchange, elliptic curve crypto).
Organisation of the lecture:

- Chapter 1: the quantum Fourier transform.
- Chapter 2: Shor's quantum factoring algorithm.

Chapters 1 and 2 will follow very closely chapters 4 and 5 from Ronald de Wolf's lecture notes[3].

[1] Peter Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, full version available here `https://arxiv.org/abs/quant-ph/9508027`

[2] Carl Pomerance, *A tale of two sieves*, Notices Amer. Math. Soc, 1996, `https://www.ams.org/notices/199612/pomerance.pdf`

[3] Full lecture notes: `https://homepages.cwi.nl/~rdewolf/qcnotes.pdf`

Chapter 1

The quantum Fourier transform

The quantum Fourier will be one of our main tools when constructing quantum algorithms. It will be at the heart of Shor's factoring algorithm.

1.1 The classical Fourier transform

Widely used: data compression, signal processing, complexity theory. Here, we will consider only the discrete Fourier transform.

1.1.1 Definition

Fourier transform F_N: $N \times N$ unitary matrix, with elements of the same magnitude.

$$F_2 := H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

$N = 3$ (for example): impossible to achieve with real numbers. Use complex numbers.
We will use roots of unity $\omega_N = e^{\frac{2i\pi}{N}}$. The discrete Fourier transform F_N is defined as

$F_N := \frac{1}{\sqrt{N}} \begin{pmatrix} \ddots & \vdots & \iddots \\ \dots & \omega_N^{jk} & \dots \\ \iddots & \vdots & \ddots \end{pmatrix}$ meaning that for any line $j \in \{0, \dots, N-1\}$ and any column $k \in \{0, \dots, N-1\}$, we have $(F_N)_{jk} := \omega_N^{jk}$.

Properties

- Each column $C_k = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 \\ \omega_N^k \\ \vdots \\ \omega_N^{(N-1)k} \end{pmatrix}$ has norm 1 and any two columns are orthogonal. Indeed

$$\forall k, k' \in \{0, \dots, N-1\},\ \frac{1}{N} \sum_{j=0}^{N-1} \omega_N^{jk} \omega_N^{-jk'} = \frac{1}{N} \sum_{j=0}^{N-1} \omega_N^{j(k-k')} = \delta_{kk'}.$$

1.1.2 Computing the classical Fourier transform

Computing the classical Fourier transform problem
Input: a column vector $v = \begin{pmatrix} v_0 \\ \vdots \\ v_{N-1} \end{pmatrix}$.
Goal: compute $\hat{v} = F_N \cdot v$.

Naive way: Perform the whole multiplication entry-wise

- $O(N)$ operations ($+$ and $\times$) per entry.
- $O(N^2)$ operations in total.

Fast Fourier transform

- $O(N \log(N))$ operations in total.
- recursive algorithm.

1.2 The quantum Fourier transform

1.2.1 Definition of the problem

Take $N = 2^n$. Since F_N is a $N \times N$ unitary matrix, we can interpret it as a quantum unitary operation acting on n qubits.

Computing the quantum Fourier transform problem
Input: a quantum state $\|\psi\rangle$ of n qubits. Goal: output $F_N(\|\psi\rangle)$.

How efficiently can we implement this quantum Fourier transform?

- QFT can be implemented with a quantum circuit of size $O(n^2)$. The rest of the chapter will be devoted to the construction of this algorithm.
- Exponentially faster than classical FFT which runs in $O(N \log(N))$.

⚠ In the classical setting, we are given an explicit (written) description of a vector $v = \begin{pmatrix} v_0 \\ \vdots \\ v_{N-1} \end{pmatrix}$ as an input and ask to have a similar description of the output. In the quantum setting, we are given a quantum state $|\psi\rangle = \sum_{i=0}^{N-1} v_i |i\rangle$ and ask to output the state $F_N(|\psi\rangle) = \sum_{i=0}^{N-1} \hat{v}_i |i\rangle$. Notice that we cannot fully recover the vector $\hat{v} = \begin{pmatrix} \hat{v}_0 \\ \vdots \\ \hat{v}_{N-1} \end{pmatrix}$ from $F_N(|\psi\rangle)$.

Even though the speedup is exponential, it doesn't allow to recover $\hat{v}$, which makes it incomparable with the FFT. The quantum algorithm in particular doesn't help in computing the classical Fourier transform. It will still have other important uses.

1.2.2 Efficient quantum circuit for QFT

Elementary gates

- Hadamard gate $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.
- Phase rotation $R_s = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2i\pi}{2^s}} \end{pmatrix}$.
- Controlled R_s gate written $C\text{-}R_s$ and acting on 2 qubits.

$$C\text{-}R_s(|0\rangle|x\rangle) := |0\rangle|x\rangle \quad ; \quad C\text{-}R_s(|1\rangle|x\rangle) := |1\rangle R_s(|x\rangle) \quad \text{which gives} \quad C\text{-}R_s(|b\rangle|x\rangle) = |b\rangle e^{\frac{2i\pi bx}{2^s}}|x\rangle.$$

For each k, we have $F_N(|k\rangle) = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{jk}|j\rangle$. For any integer j, we write its binary decomposition $j = j_1, \ldots, j_n$ where j_1 is the bit of highest weight. This means we can write $j = \sum_{l=1}^{n} 2^{n-l} j_l$. We have

$$\begin{aligned} F_N(|k\rangle) &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2i\pi jk}{2^n}}|j\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2i\pi\left(\sum_{l=1}^{n} j_l 2^{-l}\right)k}|j_1, \ldots, j_n\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \Pi_{l=1}^{n} e^{2i\pi j_l 2^{-l} k}|j_1, \ldots, j_n\rangle \\ &= \bigotimes_{l=1}^{n} \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2i\pi k 2^{-l}}|1\rangle\right) \end{aligned}$$

To prove the last equality, recall that the tensor product satisfies

$$(\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha'|0\rangle + \ \beta'|1\rangle) = \alpha\alpha'|00\rangle + \alpha\beta'|01\rangle + \beta\alpha'|10\rangle + \beta\beta'|11\rangle.$$

For any integer k, with binary decomposition $k = k_1, \ldots, k_n$, we define $0.k := \frac{k}{2^n} = \sum_{l=1}^{n} k_l 2^{-l}$. For example, $0.010 = \frac{1}{4}$ and $0.101 = \frac{5}{8}$. Notice that

$$\begin{aligned} e^{2i\pi k 2^{-l}} &= e^{2i\pi(\sum_{m=1}^{n} 2^{n-m} k_m)2^{-l}} \\ &= e^{2i\pi\left(\sum_{m=n-l+1}^{n} k_m 2^{m-(n-l+1)}\right)2^{-l}} \\ &= e^{2i\pi\left(\sum_{m'=1}^{l} k_{n-m'+1} 2^{-m'}\right)} \\ &= 0.k_n \ldots k_{n-l+1}. \end{aligned}$$

The second equality uses the fact $e^{2i\pi C} = 1$ for any $C \in \mathbb{N}$ which implies that only the $l-1$ bits of k of least weight matter in the term $e^{2i\pi k 2^{-l}}$. We can therefore rewrite

$$F_N(|k\rangle) = \bigotimes_{l=1}^{n} \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2i\pi k 2^{-l}}|1\rangle\right) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_n}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_{n-1}k_n}|1\rangle) \otimes \cdots \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_1 \ldots k_n}|1\rangle)$$

The $n = 3$ ($N = 8$) case

From the above, we have

$$F_8(|k_1k_2k_3\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_3}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_2k_3}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_1k_2k_3}|1\rangle).$$

Because of this product structure, we can easily construct each qubit separately. Notice that with the $0.k$ notation, we can write $C\text{-}R_s(|b\rangle|x\rangle) = |b\rangle e^{\frac{2i\pi bx}{2^s}}|x\rangle = |b\rangle e^{2ix\pi \cdot 0.0\ldots 0b}|x\rangle$. where $0.\underbrace{0\ldots 0b}_{s \text{ bits}}$.

1st qubit:

$$|k_3\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{k_3}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_3}|1\rangle).$$

2nd qubit:

$$|k_2\rangle \xrightarrow{\text{H}} \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{k_2}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_2}|1\rangle)$$

$$|k_3\rangle\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_2}|1\rangle) \xrightarrow{C\text{-}R_2} |k_3\rangle\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_2k_3}|1\rangle)$$

3rd qubit:

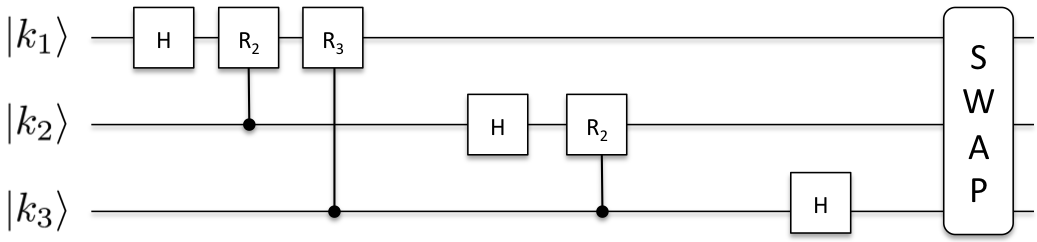
$$|k_1\rangle \xrightarrow{\text{H}} \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{k_1}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_1}|1\rangle)$$

$$|k_2\rangle\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_1}|1\rangle) \xrightarrow{C\text{-}R_2} |k_2\rangle\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_1k_2}|1\rangle)$$

$$|k_3\rangle\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_1k_2}|1\rangle) \xrightarrow{C\text{-}R_3} |k_3\rangle\frac{1}{\sqrt{2}}(|0\rangle + e^{2i\pi 0.k_1k_2k_3}|1\rangle)$$

The 3rd qubit is stored in the quantum register where $|k_1\rangle$ is and uses $|k_2\rangle$ and $|k_3\rangle$ as control qubits. The 2nd qubit is be stored in the quantum register where $|k_2\rangle$ is and uses $|k_3\rangle$ as a control qubit. The 1st qubit is stored where $|k_3\rangle$ is. In order to do this, we must first construct the 3rd qubit, then the 2nd and finally the 1st qubit. In order to have the good ordering of qubits, we end up inverting the order of all the qubits.

The circuit of F_8 is the following



The number of gates used here is $3 + 2 + 1$ gates + the gates in the SWAP.

General case

The construction for $n = 3$ can be extended to any n following the same pattern. The first qubit will consist only of an H gate while the last qubit will require applying H, $C\text{-}R_2, \ldots, C\text{-}R_n$. Similarly

as in the $n = 3$ case, we finish by inverting the order of all the qubits. The total number of gates used is therefore $n + n - 1 + \cdots + 1 + \text{SWAP} = O(n^2)$gates.

Improvements: we can reduce the number of gates if we allow for some small errors:

- As s grows, C-R_s fastly converges to the identity gate.
- We remove all those gates for $s \geq \Omega(\log(n))$. By a careful error analysis, we can show that the result will be $O(\frac{1}{\mathsf{poly}(n)})$ close to the desired state.
- The total amount of needed gates becomes $O(n \log(n))$ (SWAP is $O(n)$).