CSE 599d - Quantum Computing
The Quantum Fourier Transform and Jordan's Algorithm

Dave Bacon
Department of Computer Science & Engineering, University of Washington

After Simon's algorithm, the next big breakthrough in quantum algorithms occurred when Peter Shor discovered his algorithm for efficiently factoring numbers. This algorithm makes us of the quantum Fourier transform. In this lecture we will deviate to discuss the (quantum) discrete Fourier transform and see an application of this transform which was only recently (2005) realized.

I. DISCRETE FOURIER TRANSFORM

There are many motivations for the discrete Fourier transform. Those of you in computer science have probably encountered them first in signal processing and perhaps further in cool theory results like switching lemmas. Those of you in physics have definitely encountered the continuous fourier transform, most likely first in quantum theory where we learn that the fourier transform goes between the momentum and position representations of a wave function.

Suppose that we have a vector f of N complex numbers, f_k, $k \in \{0, 1, \ldots, N-1\}$. Then the discrete Fourier transform (DFT) is a map from these N complex numbers to N complex numbers, the Fourier transformed coefficients $\tilde{f}_j$, given by

$$\tilde{f}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{-jk} f_k \tag{1}$$

where $\omega = \exp\left(\frac{2\pi i}{N}\right)$. The inverse DFT is given by

$$f_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{jk} \tilde{f}_k \tag{2}$$

To see this consider how the basis vectors transform. If $f_k^l = \delta_{k,l}$, then

$$\tilde{f^l}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{-jk} \delta_{k,l} = \frac{1}{\sqrt{N}} \omega^{-jl}. \tag{3}$$

These DFTed vectors are orthonormal:

$$\sum_{j=0}^{N-1} \tilde{f^l}_j^* \tilde{f}_j^m = \frac{1}{N} \sum_{j=0}^{N-1} \omega^{jl} \omega^{-jm} = \frac{1}{N} \sum_{j=0}^{N-1} \omega^{j(l-m)} \tag{4}$$

This last sum can be evaluated as a geometric series, but beware of the $(l-m) = 0$ term, and yields

$$\sum_{j=0}^{N-1} \tilde{f^l}_j^* \tilde{f}_j^m = \delta_{l,m}. \tag{5}$$

From this we can check that the inverse DFT does indeed perform the inverse transform:

$$f_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{jk} \tilde{f}_k = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{jk} \frac{1}{\sqrt{N}} \sum_{l=0}^{N-1} \omega^{-lk} f_l = \frac{1}{N} \sum_{k,l=0}^{N-1} \omega^{(j-l)k} f_l = \sum_{l=0}^{N-1} \delta_{j,l} f_l = f_j \tag{6}$$

An important property of the DFT is the convolution theorem. The circular convolution of two vectors f and g is given by

$$(f * g)_i = \sum_{j=0}^{N-1} f_j g_{i-j} \tag{7}$$

where we define $g_{-m} = g_{N-m}$. The convolution theorem states that the DFT turns convolution into pointwise vector multiplication. In other words if the components of the DFT of $(f * g)$ are $\tilde{c}_k$, then $\tilde{c}_k = \tilde{f}_k \tilde{g}_k$. What use is the convolution theorem? Well this leads us nicely to our next topic, the fast Fourier transform.

II. FAST FOURIER TRANSFORM AND AN APPLICATION

Naively how many math operations do we have to do to perform a discrete Fourier transform? Well for each component of the new vector we will need to perform N multiplications and then we will need to add these components. Since we need to do this for each of the N different component. Thus we see that N^2 complex multiplications and $N(N-1)$ complex additions are needed to compute the DFT. The goal of the fast Fourier transform is to perform the DFT using less basic math operations. There are may ways to do this. We will describe one particular method for $N = 2^n$ and will put off discussion of the case where $N \neq 2^n$ until later. So assume $N = 2^n$ from here until I say otherwise.

The Fast Fourier Transform (FFT) we will consider is based on observing the fact that the there are symmetries of the coefficients in the DFT,

$$\begin{aligned} \omega^{k+N/2} &= -\omega^k \\ \omega^{k+N} &= \omega^k. \end{aligned} \tag{8}$$

Suppose we want to perform the DFT of the vector f. Split the components of f up into smaller vectors of size $N/2$, e and o. The coefficients of e are the components of f which are even and the coefficients of o are the components of f which are odd. The order of the coefficients is retained. Then it is easy to see that

$$\begin{aligned} \tilde{f}_j &= \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} \omega^{-ij} f_i = \frac{1}{\sqrt{N}} \sum_{i=0}^{N/2-1} \omega^{-2ij} e_i + \sum_{i=0}^{N/2-1} \omega^{-(2i+1)j} o_i \\ &= \frac{1}{\sqrt{N}} \left(\sum_{i=0}^{N/2-1} \omega_{N/2}^{-ij} e_i + \omega_N^{-j} \sum_{i=0}^{N/2-1} \omega_{N/2}^{-ij} o_i \right) \end{aligned} \tag{9}$$

where $\omega_{N/2} = \exp\left(\frac{2\pi i}{N}\right)$ and we have denoted ω by ω_N for clarity. We have thus obtained the a formula for the DFT of f in terms of the DFT of e and o:

$$\tilde{f}_j = \tilde{e}_j + \omega_N^{-j} \tilde{o}_j \tag{10}$$

Now recall that j runs from 0 to $N-1$ and the DFTs of e and f are periodic with period $N/2$. Using this and the above symmetry we find that we can express our formula as

$$\begin{aligned} \sqrt{2}\tilde{f}_j &= \tilde{e}_j + \omega_N^{-j} \tilde{o}_j \quad 0 \leq j \leq N/2 - 1 \\ \sqrt{2}\tilde{f}_j &= \tilde{e}_j - \omega_N^{-j} \tilde{o}_j \quad N/2 - 2 \leq j \leq N - 1 \end{aligned} \tag{11}$$

Suppose that we first compute the DFT over e and o and then uses them in this formula to compute the full DFT of f. How many complex multiplications do we need to perform? Well to compute e and o requires $2\frac{N}{2}^2 = \frac{N^2}{2}$ multiplications. We need another $N/2$ to compute $\omega_N^{-j}\tilde{o}_j$. Forget about the square root of two, it can always be put in at the end as an extra N multiplications. Thus we require $\frac{N^2}{2} + \frac{N}{2}$ complex multiplications to compute the DFT as opposed to N^2 in the the previous method. This is a reduction of about a factor of 2 for large N.

Further it is clear that for $N = 2^n$ we can use the above trick all the way down to $N = 2$. How many complex multiplications do we need to perform if we do this? Let T_n denote the number of multiplications at the $N = 2^n$th level, such that $T_1 = 4$. Then

$$T_n = 2T_{n-1} + 2^{n-1} \tag{12}$$

which can be bounded by

$$T_n \leq 2T_{n-1} + 2^n \tag{13}$$

which has solution $T_n \leq 2^n n$. In other words the running time is bounded by $N \log N$. Thus we see that in the FFT we can compute the DFT in a complexity of $N \log N$ operations. This is a nice little improvement. Of historical interest apparently Gauss knew the FFT algorithm.

Here is a very cool application of the FFT. Suppose that you have two polynomials with complex coefficients: $f(x) = a_0 + a_1 x + \cdots + a_{N-1}x^{N-1}$ and $g(x) = b_0 + b_1 x + \cdots + b_{N-1}x^{N-1}$. If you multiply these two polynomials

together you get a new polynomial $f(x)g(x) = \sum_{i,j=0}^{N-1} a_i b_j x^{i+j} = \sum_{k=0}^{2(N-1)} c_k x^k$. The new coefficients for this polynomial are a function of the two polynomials:

$$c_k = \sum_{l=0}^{N-1} a_l b_{k-l} \tag{14}$$

where the sum is over all valid polynomial terms (i.e. when $k-l$ is negative, there is no term in the sum.) One sees that computing c_k requires N^2 multiplications.

But wait, the expression for c_k looks a lot like convolution. Indeed suppose that we form a $2N$ dimensional vector $a = (a_0, \ldots, a_{N-1}, 0, \ldots, 0)$ and $b = (b_0, \ldots, b_{N-1}, 0, \ldots, 0)$ from our original data. The vector c which will represent the coefficients of the new polynomial are then given by

$$c_k = \sum_{l=0}^{2N-1} a_l b_{k-l \bmod 2N} \tag{15}$$

Now we don't need to condition this sum on their being valid terms. Now this is explicitly convolution! Thus we can compute the coefficients c_k by the following algorithm. Compute the DFT of the vectors a and b. Pointwise multiply these two vectors. Then inverse DFT this new vector. The result will be c_k by the convolution theorem. If we use the FFT algorithm for this procedure, then we will require $O(N \log N)$ multiplications. This is pretty cool: by using the FFT we can multiply polynomials faster than our naive grade school method for multiplying polynomials. It is good to see that our grad school self can do things our grade school self cannot do. Some of you will even know that you can use the FFT to multiply integers N integers with a cost of $O(N \log^2 N)$ or used recursively: $O(N \log N \log\log N \log\log\log N \cdots)$.

III. QUANTUM FOURIER TRANSFORM CIRCUITS

Now lets turn to the Quantum Fourier transform (QFT). We've already seen the QFT for $N = 2$. It is the Hadamard transform:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{16}$$

Why is this the QFT for $N = 2$? Well suppose have the single qubit state $a_0|0\rangle + a_1|1\rangle$. If we apply the Hadamard operation to this state we obtain the new state

$$\frac{1}{\sqrt{2}}(a_0 + a_1)|0\rangle + \frac{1}{\sqrt{2}}(a_0 - a_1)|1\rangle = \tilde{a}_0|0\rangle + \tilde{a}_1|1\rangle. \tag{17}$$

In other words the Hadamard gate performs the DFT for $N = 2$ on the amplitudes of the state! Notice that this is very different that computing the DFT for $N = 2$: remember the amplitudes are not numbers which are accessible to us mere mortals, they just represent our description of the quantum system.

So what is the full quantum Fourier transform? It is the transform which takes the amplitudes of a N dimensional state and computes the Fourier transform on these amplitudes (which are then the new amplitudes in the computational basis.) In other words, the QFT enacts the transform

$$\sum_{x=0}^{N-1} a_x|x\rangle \to \sum_{x=0}^{N-1} \tilde{a}_x|x\rangle = \sum_{x=0}^{N-1} \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{-xy} a_y|x\rangle. \tag{18}$$

It is easy to see that this implies that the QFT performs the following transform on basis states:

$$|x\rangle \to \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{-xy}|y\rangle \tag{19}$$

Thus the QFT is given by the matrix

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \omega_N^{-yx}|y\rangle\langle x|. \tag{20}$$

This matrix is unitary. Let's check this:

$$\begin{aligned} U_{QFT}U_{QFT}^{\dagger} &= \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\omega_N^{yx}|x\rangle\langle y|\sum_{x'=0}^{N-1}\sum_{y'=0}^{N-1}\omega_N^{-y'x'}|y'\rangle\langle x'| \\ &= \frac{1}{N}\sum_{x,y,x',y'=0}^{N-1}\omega_N^{yx-y'x'}\delta_{y,y'}|x\rangle\langle x'| = \frac{1}{N}\sum_{x,y,x'=0}^{N-1}\omega_N^{y(x-x')}|x\rangle\langle x'| \\ &= \sum_{x,x'=0}^{N-1}\delta_{x,x'}|x\rangle\langle x'| = I \end{aligned} \tag{21}$$

The QFT is a very important transform in quantum computing. It can be used for all sorts of cool tasks, including, as we shall see in Shor's algorithm. But before we can use it for quantum computing tasks, we should try to see if we can efficiently implement the QFT with a quantum circuit. Indeed we can and the reason we can is intimately related to the fast Fourier transform.

Let's derive a circuit for the QFT when $N = 2^n$. The QFT performs the transform

$$|x\rangle \rightarrow \frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}\omega_N^{-xy}|y\rangle. \tag{22}$$

Then we can expand out this sum

$$|x\rangle \rightarrow \frac{1}{\sqrt{2^n}}\sum_{y_1,y_2,\ldots,y_n\in\{0,1\}}\omega_N^{-x\sum_{k=1}^{n}2^{n-k}y_k}|y_1,y_2,\ldots,y_n\rangle \tag{23}$$

Expanding the exponential of a sum to a product of exponentials and collecting these terms in from the appropriate terms we can express this as

$$|x\rangle \rightarrow \frac{1}{\sqrt{2^n}}\sum_{y_1,y_2,\ldots,y_n\in\{0,1\}}\bigotimes_{k=1}^{n}\omega_N^{-x2^{n-k}y_k}|y_k\rangle \tag{24}$$

We can rearrange the sum and products as

$$|x\rangle \rightarrow \frac{1}{\sqrt{2^n}}\bigotimes_{k=1}^{n}\left[\sum_{y_k\in\{0,1\}}\omega_N^{-x2^{n-k}y_k}|y_k\rangle\right] \tag{25}$$

Expanding this sum yields

$$|x\rangle \rightarrow \frac{1}{\sqrt{2^n}}\bigotimes_{k=1}^{n}\left[|0\rangle + \omega_N^{-x2^{n-k}}|1\rangle\right] \tag{26}$$

But now notice that $\omega_N^{-x2^{n-k}}$ is not dependent on the higer order bits of x. It is convenient to adopt the following expression for a binary fraction:

$$0.x_l x_{l+1}...x_n = \frac{x_l}{2} + \frac{x_{l+1}}{4} + \cdots + \frac{x_n}{2^{n-l+1}} \tag{27}$$

Then we can see that

$$|x\rangle \rightarrow \frac{1}{\sqrt{2^n}}\left[|0\rangle + e^{-2\pi i 0.x_n}|1\rangle\right]\otimes\left[|0\rangle + e^{-2\pi i 0.x_{n-1}x_n}|1\rangle\right]\otimes\cdots\otimes\left[|0\rangle + e^{-2\pi i 0.x_1x_2\cdots x_n}|1\rangle\right] \tag{28}$$

This is a very useful form of the QFT for $N = 2^n$. Why? Because we see that only the last qubit depends on the the values of all the other input qubits and each further bit depends less and less on the input qubits. Further we note that $e^{-2\pi i 0.a}$ is either $+1$ or -1, which reminds us of the Hadamard transform.

So how do we use this to derive a circuit for the QFT over $N = 2^n$?

Take the first qubit of $|x_1, \ldots, x_n\rangle$ and apply a Hadamard transform. This produces the transform

$$|x\rangle \to \frac{1}{\sqrt{2}}\left[|0\rangle + e^{-2\pi i 0.x_1}|1\rangle\right] \otimes |x_2, x_3, \ldots, x_n\rangle \tag{29}$$

Now define the rotation gate

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{-2\pi i}{2^k}\right) \end{bmatrix} \tag{30}$$

If we now apply controlled R_2, R_3, etc. gates controlled on the appropriate bits this enacts the transform

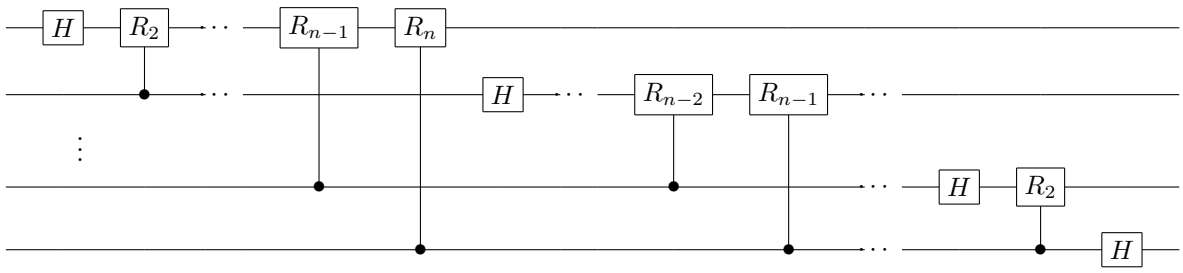
$$|x\rangle \to \frac{1}{\sqrt{2}}\left[|0\rangle + e^{-2\pi i 0.x_1 x_2 \ldots x_n}|1\rangle\right] \otimes |x_2, x_3, \ldots, x_n\rangle \tag{31}$$

Thus we have reproduced the last term in the QFTed state. Of course now we can proceed to the second qubit, perform a Hadamard, and the appropriate controlled R_k gates and get the second to last qubit. Thus when we are finished we will have the transform

$$|x\rangle \to \frac{1}{\sqrt{2^n}}\left[|0\rangle + e^{-2\pi i 0.x_1 x_2 \cdots x_n}|1\rangle\right] \otimes \left[|0\rangle + e^{-2\pi i 0.x_1 x_2 \cdots x_{n-1}}|1\rangle\right] \otimes \cdots \otimes \left[|0\rangle + e^{-2\pi i 0.x_n}|1\rangle\right] \tag{32}$$

Reversing the order of these qubits will then produce the QFT!

The circuit we have constructed on n qubits is



(33)

This circuit is polynomial size in n. Actually we can count the number of quantum gates in it: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Hadamards and controlled R_k gates plus $\lfloor \frac{n}{2} \rfloor$ swap gates. What was it that allowed us to construct an efficient circuit? Well if you look at the factorization we used, you will see that we have basically used the same trick that we used for the FFT! But now, since we are working on amplitudes and not operating on the the complex vectors themselves, we get an algorithm which scales nicely in the number of qubits. It is important to realize that the QFT cannot be used like the FFT on data. Thus there is a tendency to want to port quantum computers over to signal processing. Currently there are some preliminary ideas about how to do this, but the naive way you might expect this to work does not work.

IV. JORDAN'S NUMERICAL GRADIENT ALGORITHM

There is a very neat application of the QFT which was only recently realized. It is motivated by the Bernstein-Vazirani problem. Recall that in the nonrecursive Bernstein-Vazirani problem we given a function $f_s(x) = x \cdot s$ and we desired to find s. Notice that in some since, s is the (discrete) gradient of the function (which in this case is linear.) Now we know that central to this was the Hadamard, which we have seen is the QFT for $N = 2$. So can we use the QFT over N to learn something about the gradient of a more general function? In 2004, Jordan showed that this was possible (while he was a graduate student! This should give hope to all graduate student around the world. Indeed, it should even give hope to postdocs and faculty members as well.)

Suppose that we are given a black box which computes a function $f : \mathbb{R}^d \to \mathbb{R}$ on some pre-specified domain. We will scale this domain to $[0, 1)^d$ and assume that we have the function computed to some finite precision of bits n. Now suppose that we want to calculate an estimate of the gradiant of this function $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_d}\right)$ without loss of generality at the origin. Now classically when we query f we only obtain one value of f. A natural way to calculate an estimate of the gradient of f is to evaluate f at the origin, and then f along the d different directions, i.e. $f(0, \ldots, 0, l, 0, \ldots, 0)$, and, evaluate the differences. For small enough l this will approximate the gradient using $d + 1$

queries. A more symmetric method is to query the function at $+l/2$ and $-l/2$ and take the differences. Thus we can easily see that to evaluate an estimate of the gradiant we need some $\Omega(d)$ queries. (Really we should be defining the class of functions which we can estimate, and being careful about accuracy, etc., but we will skip over these details.)

But can we do better using a quantum algorithm? Suppose that we have access to the function f by the unitary oracle:

$$U_f = \sum_{x_1,\ldots,x_d,y\in\{0,1\}^n} |x_1,\ldots,x_d\rangle\langle x_1,\ldots,x_d| \otimes |y + f(-\frac{l}{2} + l\frac{x_1}{2^n},\ldots,-\frac{l}{2} + l\frac{x_n}{2^n}) \bmod 2^n\rangle\langle y| \tag{34}$$

where the function's input has been appropriately scaled to a region of size l. Now we do what we did in the Bernstein-Vazirani problem. First we perform phase kickback. We do this by feeding into the $|y\rangle$ register of this unitary the state

$$|\tilde{1}\rangle = \sum_{x=0}^{2^n-1} \exp\left(\frac{2\pi i x}{2^n}\right)|x\rangle \tag{35}$$

along with superpositions over all possible $|x\rangle$ inputs. The state $|\tilde{1}\rangle$ can be created by performing a QFT on $|1\rangle$. The resulting state will be

$$\sum_{x_1,\ldots,x_d\in\{0,1\}} \exp\left(\frac{2\pi i f(-\frac{l}{2} + l\frac{x_1}{2^n},\ldots,-\frac{l}{2} + l\frac{x_d}{2^n})}{2^n}\right)|x_1,\ldots,x_d\rangle \otimes |\tilde{1}\rangle \tag{36}$$

For small enough l, this is just

$$\sum_{x_1,\ldots,x_d\in\{0,1\}} \exp\left(\frac{2\pi i\left[f(0,\ldots,0) + \frac{\partial f}{\partial x_1}l(2^{-n}x_1 - \frac{1}{2}) + \cdots + \frac{\partial f}{\partial x_d}l(2^{-n}x_d - \frac{1}{2})\right]}{2^n}\right)|x_1,\ldots,x_n\rangle \otimes |\tilde{1}\rangle \tag{37}$$

If we now perfrom QFTs individually on each of these d registers we will obtain (with high probability, under some reasonable assumptions)

$$|\frac{\partial f}{\partial x_1}\frac{l}{2^n},\ldots,\frac{\partial f}{\partial x_d}\frac{l}{2^n}\rangle \otimes |\tilde{1}\rangle \tag{38}$$

Thus we see that we have obtained the gradient of the function using a single quantum query! Now of course we need to be careful about analyzing this algorithm correctly, and if you are interested in seeing the details of such a calculation you can find it at `http://arxiv.org/abs/quant-ph/0405146`.

Acknowledgments

The diagrams in these notes were typeset using Q-circuit a latex utility written by graduate student extraordinaires Steve Flammia and Bryan Eastin.