Quantum Fourier Transform

Eesh Gupta

December 23rd 2019

A critical tenet of most famous early quantum algorithms, Quantum Fourier Transform has a notoriety of being utterly non intuitive. Anything attached to "Quantum" almost always arouses confusion among our intuitive thinkers and when attached to an algorithm like "fourier transform"...God bless us all!

In this paper, I aim to explore the Quantum Foruier Transform algorithm using matrices . In this journey, we will start at something as simple as polynomial multiplication and gradually lay the foundations for creating the quantum circuit for QFT.

Some content may be supplemented by Umesh Vazirani's Youtube lecture on QFT circuit and Algorithms textbook (for FFT). The latter part of this paper is pure original content so beware!

1 Fast Fourier Transform

1.1 Representing a Polynomial

Our quest begins with a simple question: How can we represent a polynomial? The more popular representation is the coefficient representation where, given a set of coefficients $a_0, a_1, a_2, ..., a_{n-1}$, we can construct the polynomial

$$P(x) = a_0(x^0) + a_1(x^1) + a_2(x^2) + ... + a_{n-1}(x^{n-1})$$

. However, there exists a rather less convenient point-value-representation where a (n-1)'th degree polynomial can be represented by the polynomial being evaluated at a set of n points $(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_{n-1}, y_{n-1})$ such that $y_i = P(x_i)$ and all x_i are unique.

Conversion from coefficient representation to point value representation of a polynomial $P(x)$ can be summarized in 2 steps

- Choosing a set of n unique input points $\{x_0, x_1, x_2, ..., x_{n-1}\}$
- Evaluation of $P(x_i)$ at each x_i in the set $\{x_i\}$.

To visually organize such n evaluations, we use a nxn matrix X to be multiplied to a nx1 coefficient matrix $\vec{a}$, resulting in a nx1 output matrix $\vec{y}$.

Caption: When we take the dot product of the ith row vector of X with the $\vec{a}$, we are evaluating $y_{i-1} = (P(x_{i-1})$. Insert Figure 1.

Computing this matrix involves taking dot product of all n row vectors with the $\vec{a}$. Each dot product involves computing n products. If we treat each multiplication operation as $O(1)$ or constant time, then dotting each row vector with the coefficients matrix will require $O(n)$ operations. Since there are n row vectors, the total running time of such matrix multiplication will be $O(n^2)$. To avoid such quadratic order of growth, we will make a clever choice of input points that will allow us to achieve a new running time of $O(n \log(n))$.

1.2 Complex Roots of Unity

Entering the world of exponentials, let's use $e^{2\pi i/n} = \omega_n$ to make a cheeky substitution:

$$x_0 = \omega_n^0$$
$$x_1 = \omega_n^1$$
$$.$$
$$.$$
$$x_k = \omega_n^k$$
$$.$$
$$.$$
$$x_{n-1} = \omega_n^{n-1}$$

This substitution allows us to utilize the following useful properties of complex exponentials, the proofs for which are left to the reader:

- $\omega_n^n = 1$
- $\omega_n^{n/2} = -1$
- $\omega_n^{n/4} = i$
- $\omega_n^{k+n/2} = -\omega_n^k$ (Halving Lemma)

Insert FIgure 2 (30.2 from Algo textbook with n=n instead of n=8) Caption: X diagram to assist intuition with properties of complex roots of unity. Can you now explain why ω_n is called a complex root of unity?

These properties will allow us to speed up our computation of matrix multiplication. The resulting savings make Fast Fourier Transform incredibly powerful and the quantum version even more so. For now, we make the the matrix multiplication problem more explicit following the above substitutions.

Insert Figure 3(Figure 1 but with ω instead of x)

1.3 Evaluation

Evaluation begins with observing some symmetries which are integral to the FFT algorithm. Pay careful attention to the 'cuts' we make to the original matrix X for visual intuition. It is also advised for the reader to follow through with an example (like $n = 8$) or derive the general results by themselves to appreciate the beauty of these symmetries.

1. Computing $y_{i+n/2}$ from y_i using the halving lemma:

$$y_i = X(\omega_n^i)$$
$$y_{i+n/2} = X(\omega_n^{i+n/2}) = X(-\omega_n^i)$$

Thus, computing the upper half of $\vec{y}$, i.e. $y_1, ..., y_{n/2-1}$ and applying the negative sign to ω_i's takes care of the remaining terms in the lower half of $\vec{y}$ i.e. $y_(n/2), ..., y_{n-1}$. Hence our total computations are cut in half as we only have to compute the upper half of the $\vec{y}$. This entails ignoring the lower half of matrix X as the row vectors in lower half of matrix X correspond to y_i in the lower half of $\vec{y}$.

Insert Figure 4(Output matrix's and A's lower half covered)

2. Further dividing each y_i:

We can reduce our work even more by dividing the $\vec{a}$ of X into 2 halves: one containing even coefficients$A_0$and one containing odd coefficients A_1. To illustrate:

$$A(\omega_n) = a_0\omega_n^0 + a_1\omega_n^1 + ... + a_{n-1}\omega_n^{n-1}$$

Let

$$A_0(\omega_n^2) = a_0\omega_n^0 + a_2\omega_n^2 + ... + a_{n-2}\omega_n^{n-2}$$
$$A_1(\omega_n^2) = a_1\omega_n^0 + a_3\omega_n^2 + ... + a_{n-1}\omega_n^{n-2}$$

Thus the original $A(\omega_n)$ reduces to

$$A(\omega_n) = A_0(\omega_n^2) + \omega_n A_1(\omega_n^2)$$

Using the first symmetry, we arrive at the following general result:

$$y_i = A(\omega_n^i) = A_0(\omega_n^{2i}) + \omega_n^i(A_1(\omega_n^{2i}))$$
$$y_{i+n/2} = A(-\omega_n^i) = A_0(\omega_n^{2i}) - \omega_n^i(A_1(\omega_n^{2i}))$$

So how does such even-odd coefficients division affect matrix A? Since we are dividing the $\vec{a}$ into even and odd entries, we must reorder the column vectors of X appropriately as well. Consequently, we must reorder the columns of matrix X such that even numbered columns are separated for odd numbered columns. (see figure 5)

(figure 6 —ith row vector in upper half and lower half) Notice that in the i'th row vector in the upper half of matrix A, we can factor out ω_n^i from the odd columns, resulting in the odd columns looking similar to even columns. Also, notice that in the i'th row in the lower half of matrix A, we can factor out $-\omega_n^i$ from the odd columns, resulting in the odd columns looking similar to even columns. These results directly follow from the equations above.

This symmetry allows us to further cut our computations in half, this time by slashing off right half of matrix A. We can obtain right half by applying appropriate phase corrections to the left half. (figure 7: show right half of reordered matrix A covered and bottom of coefficient matrix A covered)

Combining these symmetries, we obtain a matrix A with reordered column vectors, cut into quarters as shown in figure 8 below.

Instead of computing n row-vector-coefficient-matrix dot products resulting in n^2 computations, we use the following algorithm:

1. Divide the reordered matrix into quarters. *
2. Compute only the top left quarter.
3. Add relevant phase corrections to obtain the remaining 3 quarters

After all these dissections to A, we are still left with computing the top left quarter. The trick now is to divide that quarter further into quarters, repeating the same algorithm. This gives fast fourier transform its recursive structure.

Insert 9 showing recursive structure

Now how does this visual intuition relate to the actual algorithm? Assume matrix A as a coordinate grid with 4 quadrants.

```
Recursive FFT(a)
  n = a.length                    //length of some coefficient matrix a
  if n==1
    return a                      //establish base case
  w_n = e^(2*pi*i/n)              //setting up the variables for for loop
  w = 1
  a_[0] = (a_0, a_2, ...., a_(n-2)) //even coefficients
  a_[1] = (a_1, a_3, ...., a_(n-1)) //odd coefficients
  y^[0] = Recursive FFT(a_[0])    //recursive calls y^[0] and y^[1] that fill
  y^[1] = Recursive FFT(a_[1])    //up quadrant 1 and 2 respectively
  for k = 0 to n/2 -1             // ‘‘for each row in upper half of A’’
    y_k = y^[0] + w*y^[1]         //row vector in upper half with phase correction
    y_(k+n/2) = y^[0] - w*y^[1]   //row vector in lower half with phase correction
    w = w*(w_n)                   //updating phase correction for next set of rows
  return y                        //returning array of row vectors in some quarter
                                  //of some larger matrix.
```

To analyze the running time of this algorithm, one must recognize that we can only divide a nxn matrix into quarters $\log n$ times. And after each division, we still have to apply proper phase corrections to row vectors in the remaining 3 quarters at each recursive call. These operations scale linearly with input size. Since there there are $\log n$ recursive calls, each requiring on the order of $O(n)$ operations, we can accept anaytically, that the resulting running time of Fast Fourier Transform will be $O(n \log n)$, a linear speed-up over our initial, naive, gross quadratic algorithm!

1.4 Summary

The aim of FFT was to evaluate the point-value representation of some polynomial A. The savings stemmed from usage of exponentials as input points, which gave rise to the halving lemma which drastically cut computations. These savings were illustrated by making "cuts" to the original matrix containing the input points

2 Quantum Fourier Transform

Now, we have the tools to implement the Quantum Fourier Transform. The trick is to utilize the Recursive FFT algorithm and encode it into a quantum circuit. While the resulting circuit may look very different from the algorithm, the underlying transformations to matrix A remain similar in both cases.

Insert Figure 10(n = 8 matrix)

2.1 Parallels to Fast Fourier Transform

While we were concerned with evaluating the point value representation of a polynomial in FFT section, in QFT, we are more concerned with transforming basis states from one basis to another. What does this mean? Consider matrix multiplication. In linear algebra, we are taught that a matrix represents a linear transformation that scales or rotates the initial basis vectors, transforming the space in some manner.

Insert Figure 13(3B1B version of matrix as linear transformation...maybe an untransformed matrix on left followed by action of matrix which outputs a transformed matrix on the right panel)

If we considered matrix A from the previous section as some transformation matrix and the coefficient matrix as some basis state, then the $\vec{y}$ would represent the transformed basis state. Hence, QFT and FFT are concerned with the same matrix multiplication problem, just under different contexts. Now, since QFT wants the input and outbut states to be basis states, the transformation matrix A must be unitary and the coefficient matrix must be normalized.

For the extended example, we will assume that coefficient matrix represents the state $|110\rangle$ which can be represented in matrix form as

insert figure 14(0 0 0 1 0 0 0 0)

In terms of the quantum circuit, the first qubit is in state $|0\rangle$ while the other qubits are in state $|1\rangle$.

Insert figure 15 (quantum circuit with initial states labeled)

Given the transformation matrix A in figure 10 and the coefficient matrix in figure 14, the matrix multiplication should be easy enough: the output matrix will be the 4th row vector of matrix A. In the following subsections, we will examine how the quantum circuit finds and constructs the 4th row vector from the initial state $|110\rangle$.

2.2 Reverse Fast Fourier Transform

Quantum Fourier Transform can be generally described as following:

1. Reorder the columns of matrix into even and odd halves
2. Add relevant phase corrections using single qubit and 2 qubit gates
3. Zoom into top left quarter and REPEAT!

This may seem as reverse of FFT as the former algorithm executes step 3 before step 2. Indeed, one may ask how can we add phase corrections before the computation of top left quarter?

2.3 Reordering columns

This step is rather assumed and there are no qubit operations that reorder the columns of matrix A. Regardless, one should not ignore this step.

Insert Figure 11(Matrix with reordered columns and empty 3 qubit circuit)

2.4 Phase Corrections

Phase correction phase of quantum fourier transform "finds" the row vector corresponding to the coefficient matrix/initial state. Consider the following formulas from FFT:

$$y_i = A(\omega_n^i) = A_0(\omega_n^{2i}) + \omega_n^i(A_1(\omega_n^{2i}))$$

$$y_{i+n/2} = A(-\omega_n^i) = A_0(\omega_n^{2i}) - \omega_n^i(A_1(\omega_n^{2i}))$$

Phase correction consists of 2 components

1. Hadamard gate to add or subtract $A_0(\omega_n^{2i})$ from $A_1(\omega_n^{2i})$.
2. Phase gates to add the factor of ω_n^i to $A_1(\omega_n^{2i})$.

The following sections will explain the intuition behind the use of hadamard and phase gates.

This quest to find the row vector will follow more along the lines of binary search. How does binary search work? Given an element Target, binary search evaluates whether an array contains Target and returns its index in the array:

1. Find the midpoint of the sorted array
2. Compare target with the middle element
3. If target¿middle element, binary search will only consider the upper half of the array (containing elements greater than middle element)
4. If target¡ middle element, binary search will only consider the lower half of the array (containing elements lesser than middle element)
5. Repeat!

The logic for the QFT circuit (phase correction aspect) follows a similar approach. But the aim of QFT circuit is not only to find the row vector, but also to apply proper phase correction as it goes along doing so.

Row-Vector-Search-And-Phase-Correction Algorithm:

1. Given:qubit least significant qubit (LSB) and k (the number of row vectors of the current matrix)
2. If LSB is $|0\rangle$, only consider the top half.
3. If LSB is $|1\rangle$, only consider the bottom half of matrix and apply phase factor of $\omega_n^{k/2}$.
4. Move on to the next qubit and divide k by 2.

2.4.1 A Brief Look at Controlled Phase Gates

Let's contruct a sample 2 qubit circuit with a controlled phase gate. (Insert Figure 25: the circuit in question)

The first qubit is called the control qubit and the second qubit is called the target qubit. Here is the working

- If the control qubit is $|0\rangle$, then nothing happens to target qubit.
- If the control qubit is $|1\rangle$, then a specified phase is added to the state of target qubit.

Here is a truth table and matrix multiplication to illustrate the above described logic. HADAMARD GATE IS NOT SIMPLY A PHASE GATE

2.4.2 Hadamard Gate

Consider the formulas again:

$$y_i = A(\omega_n^i) = A_0(\omega_n^{2i}) + \omega_n^i(A_1(\omega_n^{2i}))$$

$$y_{i+n/2} = A(-\omega_n^i) = A_0(\omega_n^{2i}) - \omega_n^i(A_1(\omega_n^{2i}))$$

The former formula describes the ith row vector in the top half of A and the latter formula describes the i'th row vector in the bottom half of A. Careful

Nielsen and Chuang graduates may recognize something weird yet insightful: If we let $A_0(\omega_n^{2i})$ to be state $|0\rangle$ and $A_1(\omega_n^{2i})$ to be the state $|1\rangle$, then the same equations can be written as

$$y_i = |0\rangle + \omega_n^i |1\rangle$$

$$y_{i+n/2} = |0\rangle - \omega_n^i |1\rangle$$

which resembles an application of hadamard gate followed by phase gates. In fact, y_i corresponds to the resulting state of some qubit lsb being $|0\rangle$ and $y_{i+n/2}$ corresponds to the resulting state when the qubits' initial state is $|1\rangle$. We can represent this by adding a hadamard gate to the first qubit:

Insert (Figure 12: H gate on first qubit)

One can also rexamine the following equations by considering $|0\rangle$ representing the even columned entries of a row vector and $|1\rangle$ representing the odd columned entries of a row vector.

$$y_i = |0\rangle + \omega_n^i |1\rangle$$

We add the even columned entries to odd columned entries if the row vector is contained in the upper half of matrix A. On the other hand, if the row vector in lower half, we subtract the odd columned entries from the even columned entries. This may be similar to invoking the first symmetry from section 1.3. For visual intuition, refer to figure 8.

$$y_{i+n/2} = |0\rangle - \omega_n^i |1\rangle$$

2.4.3 Phase Gates

What about the additional phase correction on the odd columned entries? Note that the factor of ω_n^i corresponds to the row vector in the upper half of A. For example, the first row corresponds to i = 0 and hence a factor of $\omega_n^0 = 1$ is added to $|1\rangle$. However, the third row vector corresponds to i = 2 and hence a factor of ω_n^2 is added to $|1\rangle$. So how can we "find" our row vector. Well, that's encoded in the intial qubit states. To illustrate, the state $|011\rangle$ tells us that

- First Qubit is $|1\rangle$: We are concerned with only the lower half of matrix A i.e. i $\in$ [4,7]. Let's call this lower half of matrix A 'B'.
- Second Qubit is $|1\rangle$: We are concerned with only the lower half of B i.e. i $\in$ [6,7]. Let's call this lower half of matrix B 'C'.
- Third Qubit is $|0\rangle$: Here we are concerned with the upper half of C i.e. $i = 6$. At this point we know that our mystery row vector is the seventh row of matrix A.

(Insert Figure 18: 3 matrices showing the transformations above)

Careful readers may also note that if we read the state backwards i.e. read $|011\rangle$ as 110, this is simply the binary representation of 6 so we can easily deduce

from the state itself what row vector we are concerned with. (Note that this only implies for simple states without superposition)

From this discussion, we conclude that locating a row vector requires us to consider the states of other qubits as well. However, QFT is not content by just finding the row vector; we must also add the phase correction ω_n^i. We know that i depends on the states of all qubits but there is no easy way to reverse the string and convert between binary and decimal representations on a quantum circuit. So how do we find i and add the phase correction ω_n^i?

Notice the pattern as I describe the process of adding phases in our example of $|011\rangle$:

1. First Qubit is $|1\rangle$: Add factor of $\omega_n^4 = -1$
2. Second Qubit is $|1\rangle$: Add factor of $\omega_n^2 = i$
3. Third Qubit is $|0\rangle$: Don't do anything (If third qubit was $|1\rangle$ we would have added a factor of ω_n^1)

This results in the state

$$|0\rangle + \omega_n^6 |1\rangle = |0\rangle - \omega_n^2 |1\rangle$$

. Notice that at every stage, the power on ω_n is being divided by 2. And at the end, we get i = 6 as we predicted earlier. This evaluation is subtle but at essence, its just performing a conversion from binary to decimal. When it considers the first qubit, it already reverses the state. Since 1 is the third digit from the left and we know that the third digit corresponds to $1 * 2^2 = 1 * 4 = 4$, the phase added in the first step is ω_n^4. And if we skip to the third step, we see that the first digit from left is 0 and $0 * 2^0 = 0 * 1 = 0$ which corresponds to no phase being added at that step. With the tools acquired from above discussion, you should be able to compute the phase added in the first recursive call to our state from section 2.1, $|110\rangle$. Can you now explain the new additions to our quantum circuit below. (Insert figure 19: quantum circuit with full first recursive call)

CAUTION ERROR WITH PHASE GATES AND HADAMARD GATE. NEED TO MERGE THESE SECTIONS AS JUST ONE SECTION: PHASE CORRECTION OR SOMETHING

2.4.4 Repeat

The above subsections take care of 1 recursive call. Recall from FFT algorithm that in the next recursive call, we are to zoom into the top left quarter: (Insert FIgure 20: unordered columns)

Before reordering the columns, pay special attention to the entries of the matrix above. Notice that every row vector is a geometric series of ω_n^{2k} for $0 < k <= 3$. If we were to construct a 2 qubit circuit for QFT or a 4*4 matrix for FFT, it would look something like this: (Insert FIgure 21: normal 4*4 FFT) Notice that , in contrast to figure 20, every row vector is a geometric series of

ω_n^k for $0 < k <= 3$. Since our algorithm is more adapted to the second case, we have to make a few adjustments. For example, our central equations in the second recursive call would be

$$y_i = A(\omega_n^{2i}) = A_0(\omega_n^{4i}) + \omega_n^{2i}(A_1(\omega_n^{4i}))$$

$$y_{i+n/2} = A(-\omega_n^{2i}) = A_0(\omega_n^{4i}) - \omega_n^{2i}(A_1(\omega_n^{4i}))$$

as opposed to

$$y_i = A(\omega_n^{i}) = A_0(\omega_n^{2i}) + \omega_n^{i}(A_1(\omega_n^{2i}))$$

$$y_{i+n/2} = A(-\omega_n^{i}) = A_0(\omega_n^{2i}) - \omega_n^{i}(A_1(\omega_n^{2i}))$$

Why does this make sense? Note that the latter set of equations represent the row vectors of the first recursive call. To evaluate these equations, one needs to evaluate $A_0(\omega_n^{2i})$ and $A_1(\omega_n^{2i})$. In the second recursive call, the evaluation of those terms would require us to replace ω_n^i with ω_n^{2i} in the latter set of equations. Hence we would obtain the first set of equations. Now how would we proceed with computations of the second recursive call? Well it would be exactly as we did with the first recursive call although we only work with second and third qubits now:

1. Second Qubit is $|1\rangle$: Add factor of $(\omega_n^2)^2 = \omega_n^4 = -1$
2. Third Qubit is $|0\rangle$: Don't do anything (If third qubit was $|1\rangle$ we would have added a factor of $(\omega_n^2)^1 = \omega_n^2 = i$)

(Insert figure 22: Show circuit)

2.4.5 Last Repeat

Third recursive call and we are only left with the third qubit. Let's look at the top left quarter of the reordered matrix from the second recursive call. (Insert figure 23: Show top left quarter) Notice how now we are dealing with series of ω_n^{4k} for $0 < k <= 1$? At this point the pattern should be apparent to you:

1. Third Qubit is $|0\rangle$: Don't do anything (If third qubit was $|1\rangle$ we would have added a factor of $(\omega_n^4)^1 = \omega_n^4 = -1$)

(Insert Figure 24: Show final circuit)

2.5 Conclusion

It took us 10 pages and 25 figures to finally understand the QFT circuit. I hope it is more apparent to you why the QFT circuits look the way they do with their staircase like pattern. It is my understanding that Quantum Computing looks remarkably beautiful from this linear algebra POV than pure math equations. But at the same time, one needs to sift through the unpleasant parts of mathematics to really value the beauty of this approach. I thank you for going through this journey with me.