How does FFT reduce the complexity of DFT from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\ log\ N)$?

March 26, 2020

In this post, I will use a simple example to explain the magic behind FFT. The DFT matrix when $N = 4$, for example, is given by,

$$DFT(4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Given a column vector $X = [x_0 \quad x_1 \quad x_2 \quad x_3]^T$, its DFT is given by $Y = DFT(4) \times X$. It is a dense matrix vector product and requires $\mathcal{O}(4^2)$ computations.

The FFT algorithm decomposes the above dense DFT matrix into a product of $log_2\ N$ *i.e.* 2 sparse matrices as shown below. The first two sparse matrices have exactly 2 non zero elements per row (independent of N). The third (last) matrix is an exchange matrix [1]. It just swaps the second and third row of the input and does not involve any computation.

$$DFT(4) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

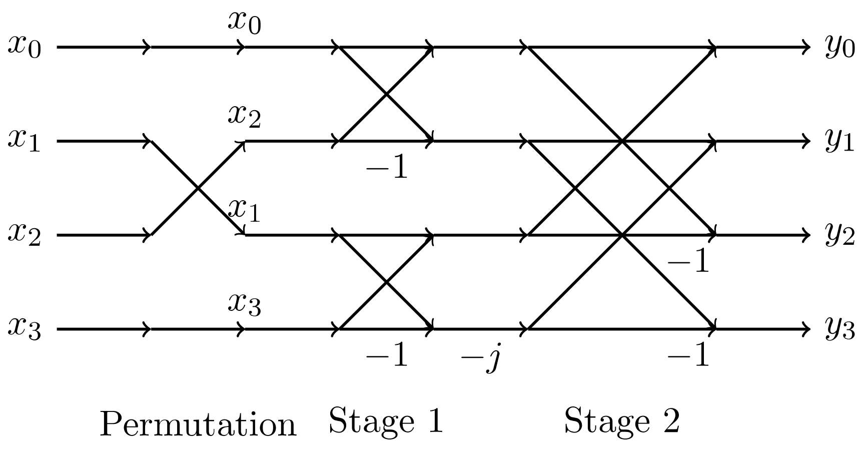
Using this formulation, the DFT of X is given by,

$$\begin{aligned} Y &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_1 \\ x_3 \end{bmatrix} \end{aligned}$$

In the next step, the (sparse matrix) vector product involves only $2N$ computations instead of N^2 computations. Again, in the final step, the (sparse matrix) vector product involves only $2N$ computations instead of N^2 computations *i.e.* for each (sparse matrix) vector product, we do $2N$ computations. Therefore, overall, the total number of computations to do a DFT is equal to the number of sparse matrices times $2N$ *i.e.* $(log_2\ N) \times 2N$ = $\mathcal{O}(N\ log\ N)$.

FFT Butterfly

The above sequence of $log_2\ N$ sparse matrix vector products is popularly represented graphically using butterfly diagrams [2].



DFT Decomposition

The question which now remains to be answered is how do we decompose the dense DFT matrix into a product of sparse matrices. As an example, we will decompose the $DFT(4)$ matrix. In the process, we will make use of the block matrix notation [3]. From above,

$$DFT(4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

If we permute the matrix to move the odd columns (1 and 3) to the left and the even columns (2 and 4) to the right, we get,

$$DFT(4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -j & j \\ 1 & 1 & -1 & -1 \\ 1 & -1 & j & -j \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using block matrix notation (for ease), we can write the first matrix as

$$M = \begin{bmatrix} A & B \\ A & -B \end{bmatrix} \text{where, } A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 \\ -j & j \end{bmatrix}$$

Note that $A == DFT(2)$. Furthermore, we can decompose B as,

$$B = \begin{bmatrix} 1 & 0 \\ 0 & -j \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = CA, \text{ where } C = \begin{bmatrix} 1 & 0 \\ 0 & -j \end{bmatrix}$$

Therefore,

$$M = \begin{bmatrix} A & CA \\ A & -CA \end{bmatrix} = \begin{bmatrix} I_2 & C \\ I_2 & -C \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix} \text{where, } I_2 \text{ is the identity matrix } \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Expanding out all the block matrices, we get

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

and

$$DFT(4) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$Q.E.D.$

Note that $DFT(2)$ appeared in the decomposition of $DFT(4)$. Likewise, $DFT(4)$ appears in the decomposition of $DFT(8)$ and so on.

Code

Here is the code to verify the DFT factorization.

```
import numpy as np
from scipy.linalg import dft

np.set_printoptions(precision=2, suppress=True)  # for compact output

sparse1 = np.array([
                    [1, 0,  1,  0 ],
                    [0, 1,  0, -1j],
                    [1, 0, -1,  0 ],
                    [0, 1,  0,  1j]])

sparse2 = np.array([
                    [1,  1, 0,  0],
                    [1, -1, 0,  0],
                    [0,  0, 1,  1],
                    [0,  0, 1, -1]])

permutation = np.array([
                        [1, 0, 0, 0],
                        [0, 0, 1, 0],
                        [0, 1, 0, 0],
                        [0, 0, 0, 1]])

# Get the DFT matrix
result1 = dft(4)

# DFT matrix factorization
result2 = (sparse1 @ sparse2 @ permutation)

print(np.absolute(result1 - result2))
```

References

1. Exchange Matrix
2. Butterfly Diagram
3. Block Matrix