 doi:10.20944/preprints202203.0129.v1

A GENERIC SCALABLE DFT CALCULATION METHOD FOR VECTORS AND MATRICES USING MATRIX MULTIPLICATIONS

A PREPRINT

Abdullah Aman Khan
School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, Sichuan, China
abdkhan@hotmail.com

January 11, 2021

ABSTRACT

Computation of Discrete Fourier Transform (DFT) is a challenging task. Especially, on computational machines/embedded systems where resources are limited. The importance of Fourier Transform (FT) cannot be denied in the field of signal processing. In this paper, we propose a technique that can compute Discrete Fourier Transforms for a matrice or vector with the help of matrix multiplication. Moreover, we discuss the trivial methods used for computation of DFT along with methods based on matrix multiplication used to compute discrete Fourier Transform. Furthermore, we explain the shortcomings. Our method can help in the calculation of a Discrete Fourier Transformation matrix by truncation of values from our proposed generic method which can help in computing DFT of varying lengths of vectors. On computing machines and programming environments, having support for matrix multiplication, our proposed methodology can be implemented.

Keywords Discrete Fourier Transform · Fourier Transform · Twiddle factor

1 Introduction

Modern computational systems require flexibility and scalability to carter the abruptly changing demands in technology. These demands force the designers to change the system parameters on the fly to ensure the quality of service. Fast Fourier transform (FFT) [1] is a fast and efficient algorithm for computing the Discrete Fourier transform (DFT) [1, 2, 3]. Over the past few years, Image, Audio, Digital communication systems and deep learning based systems have undergone massive production [4, 5, 6, 7, 8, 9, 10, 11, 12]. Most of the algorithms used for these systems involve the computation of Fourier transforms. The computational complexity of DFT is a significant factor. Although the Fast Fourier Transform algorithm decreased the computational complexity of the Discrete Fourier Transform, the importance of the Discrete Fourier transform method still stands in its place. For an educational purpose, a brief report for computing Discrete Fourier transform is presented along with a proposed technique that can compute Discrete Fourier transform using matrix multiplications.

For computation of DFT, a vector of N elements, a DFT Matrix containing the twiddle factors should store $N \times N$elements. Thus for a vector of M elements will require to compute a matrix containing twiddle factors of size MXM i.e. whenever the size of the input vector is changed a new DFT Transformation Matrix needs to be computed every time. Our approach stores a pre-computed general matrix that is similar to a lookup table. The number of required elements will be truncated from this general matrix and multiplied with the input vector, which in turn, yields the Fourier Transform for the input Matrix. This technique can be applied in a general computing environment and other dedicated machine architectures which support matrix multiplications.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 March 2022 doi:10.20944/preprints202203.0129.v1

2 Discrete Fourier Transform

The discrete-time Fourier transform is useful for interconversion of a signal from the spatial domain and frequency domain. The frequency-domain has the same information of the signal as the spatial domain but in a different form. A discrete Fourier transform of a two-dimensional square matrix can be computed by the following expression.

The DFT of a vector can be calculated using the following expression:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} k \in \mathbb{Z} \tag{1}$$

For expressing the inverse DTFT of a vector the following equation can be employed:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi kn/N} \quad n \in \mathbb{Z} \tag{2}$$

The following equation can be used to calculate DFT of a matrix:

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-\frac{i2\pi(xu+vy)}{N}} \quad for\ u, v \in [0, N-1] \tag{3}$$

A two Dimensional matrix can be represented by multiple one dimensional Fourier transforms:

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} F(x,v) e^{-\frac{i2\pi ux}{N}} for\ u \in [0, N-1] \tag{4}$$

Where $W_N^k = e^{-j2\pi x/N}$ or $e^{-j2\pi y/N}$ are the twiddle factors [13]. So, it can be concluded that the DFT of a square matrix$(M \times M)$ is computed by performing M one dimensional DFTs for each row. Afterward, the same procedure is applied to the resultant but column-wise.

3 Trivial method for calculating DFT

There are many trivial methods used to compute the Discrete Fourier transform of a vector / Matrix. Equation (1) can be used to compute the Discrete Fourier Transform of a given vector x. Typical methods for computing DFT along with MATLAB code and simulation are presented.

The most common method used for computing DFT involves the programming implementation of Equation (1), the following program snippet is capable of computing DFT of a provided vector. The code can calculate the DFT of the specified N Values:

```
% DFT Calculation By Equation

N=15;          %Define the Number Of points
x=rand(1,N);   % A vector with N Random Values
DFT=zeros(1,N); % Allocation Zeros to a new vector for storing Results

% The process (Expansion)
for K=0:N-1
   for n=0:N-1
      DFT(K+1)= DFT(K+1)+( x(n+1) * exp((-i*2*pi*K*n)/N))
   end
end
```

Similarly, the inverse Fourier transform of the vector can also be computed using Equation (2) and the following code:

```
%Inverse IDFT Calculation By Equation

N=15;           %Define the Number Of points
x=rand(1,N);    % A vector with N Random Values
IDFT=zeros(1,N); % Allocation Zeros to a new vector for storing Results
```

```
% The process Inverse Expansion
for K=0:N-1
   for n=0:N-1
      IDFT(K+1)= IDFT(K+1)+( x(n+1) * exp((i*2*pi*K*n)/N))
   end
end
IDFT=IDFT/N; %Now IDFT Contains The inverse
```

A DFT transformation matrix can be represented as a Vandermonde matrix as shown below. The elements of this matrix are commonly known as twiddle factors:

$$F = \begin{bmatrix} \omega_N^{0.0} & \omega_N^{0.1} & \dots & \omega_N^{0.(N-1)} \\ \omega_N^{1.0} & \omega_N^{1.1} & \dots & \omega_N^{1.(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{(N-1).0} & \omega_N^{(N-1)\cdot 1} & \dots & \omega_N^{(N-1)\cdot(N-1)} \end{bmatrix} \tag{5}$$

Where $\omega_N = e^{\frac{-2\pi l}{N}}$. Whenever a vector is multiplied by the transformation matrix F. It yields the DFT of the provided vector i.e.:

$$X = \mathrm{x}F \tag{6}$$

The transformation matrix contains $N \times N$ elements, where, N is the total number of elements in the vector k. The following MATLAB code can compute the Fourier transformation matrix F, which, can be used for computing the Fourier transformation matrix.

```
%Code for Computing DFT Transformation Matrix

N=8; % Transformation Matrix for N Number of Points
F=zeros(N,N);
for x=0:N-1
   for y=0:N-1
      F(x+1,y+1)=(exp((-2*pi*i)/N))^(x*y);
   end
end
```

For computing, the inverse DFT using the transformation Matrix the input matrix X (the DFT of vector x) is multiplied to the conjugate of the transformation matrix and by Dividing the resultant with N yields the original signal i.e.

$$x = \frac{1}{N} X F^* \tag{7}$$

Another way to compute the transformation matrix is to pre-compute the DFT of an Identity matrix of size $M \times M$ where M is the size of the vector x whose DFT needs to be calculated, the resulting matrix can be used as a transformation matrix itself. The following expression explained the calculation of the transformation matrix.

$$\mathrm{F} = \mathrm{DFT}\left(\mathrm{I_{MxM}}\right) \tag{8}$$

This method will yield the same transformation matrix as Vandermonde matrix explained above section. For computation of DFT for a two dimensional matrix typically the Fourier transform can be computed by computing the fourier transform of each column of the matrix x, then again computing the Fourier transform of each column of the transpose of the resultant and then again transposing. Mathematically it can be written as:

$$X = \mathrm{DFT}\left(DFT(x)'\right)' \tag{9}$$

If it is desired to calculate the DFT using the transformation matrix, no transposing is required the DFT can be calculated by simple matrix multiplication of the input matrix with the transformation matrix which can be represented as:

$$X = F \times F \tag{10}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 March 2022 doi:10.20944/preprints202203.0129.v1

Where X is the resulting two-dimensional DFT, F is the transformation matrix and x is the input square matrix.

4 Proposed Method

A problem arises with the transformation matrix that DFT can only be computed for the size of N points. If it is required to compute DFT of points other than N, The transformation Matrix has to be computed again, the DFT computation is already compute-intensive. Computing another transformation matrix for this new sized vector will require more computation and storage in the memory.

There is a possibility, that a general DFT Transformation matrix can be employed like a lookup table (which is larger). To construct a generic matrix, a mechanism is required for truncating values from a bigger pre-computed matrix. Suppose that the Generic Transformation matrix is computed for a NXN. This matrix will be capable of computing the DFT of N point vector. Now a situation occurs that it is required to compute the DFT of M Points vector Where $M < N$ a simple truncation can be done in the generic matrix already computed for N points. A new matrix can be produced form a larger generic matrix, the truncation can be done from the first row till M^{th} row and from the first column till M^{th} column.

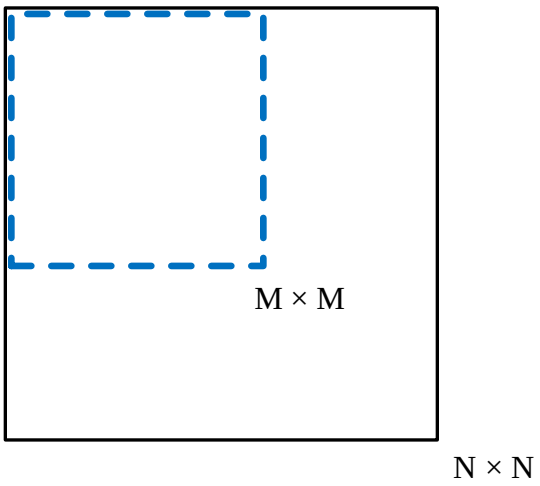

Figure 1: Truncation of pre-calculated values from larger Generic Matrix for computation of DFT.

In Figure 1, the bigger bold Square represents the generic matrix, and the dotted square represents the smaller truncated new transformation matrix. The purpose of using this mechanism is to provide a flexible and scalable stored table DTFT calculation Matrix. A twiddle factor, is stored in the transformation matrix. Here, we can see that it holds the effect of N. The goal is to create a general DTFT matrix, that can be further truncated according to the required points. To achieve this, we have to remove the effect of N in the equation. Instead of storing the Value we can simply store the value $-2\pi i$ after truncating. The new matrix can be divided by the total number of new point M and afterward taking the exponential. This yields a new transformation matrix.

Trivially, the twiddle factor is represented as:

$$W = \omega^{jk} \tag{11}$$

Where:

$$\omega = e^{-2\pi i/N} \tag{12}$$

By taking Log on both sides:

$$\log(\omega) = \log\left(e^{-2\pi i/N}\right) \tag{13}$$

Simplifying Equation 13 yields:

$$\log(\omega) = \frac{-2\pi i}{N} \tag{14}$$

$$\mathrm{N}\log(\omega) = -2\pi i = \omega_{new} = -6.2832i \tag{15}$$

Here instead of using the twiddle factor, we simply store $-2\pi i$ instead of the twiddle factor in the general DFT transformational Matrix. For our transformation matrix, we store the new twiddle factor ω_{new} . Then the generic Transformation Matrix can be represented as:

$$G = \begin{bmatrix} \omega_{\text{new}} .0.0 & \omega_{\text{now}} .0.1 & \dots & \omega_{\text{new}} .0.(N-1) \\ \omega_{\text{new}} .1.0 & \omega_{\text{new}} .1.1 & \dots & \omega_{\text{new}} .1.(N-1) \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{\text{new}} \cdot (N-1).0 & \omega_{\text{new}} \cdot (N-1).1 & \dots & \omega_{\text{new}} \cdot (N-1) \cdot (N-1) \end{bmatrix} \tag{16}$$

G represents the new generic transformation matrix, this matrix does not have the effect of the total number of elements, thus can be truncated for different sized matrices. After truncation, some more processing is required. The new truncated transformation matrix needs to be divide by N, which, is the total number of elements for the vector whose DFT is to be calculated:

$$F = e^{\frac{G}{N}} \tag{17}$$

The point to be noted is that the number of operations is almost the same as required by the computation of the regular transformation matrix. The above procedure can mathematically be explained by the following set of equations.

$$\frac{N \log(\omega)}{N} = \frac{-2\pi i}{N} \tag{18}$$

$$N \log(\omega) = \frac{-2\pi i}{N} \tag{19}$$

Taking the Exponential of the above equation the expression reverts.

$$\omega = e^{\frac{-2\pi i}{N}} \tag{20}$$

This generic table holds the values that are the exponents. A Matlab simulation for the proposed method is provided to support our claim:

```
%Simulation: DFT of vector using Matrix Multiplication
%Generating a general DFT Matrix that can be cropped according to a new
%General Matrix row and columns
N=10;
v=rand(1,N); % V is the vector
Gr=256;      % Dimension of the Generic Matrix
Gc=Gr;

% Calculation The Generic Matrix
for x=0:Gr-1
   for y=0:Gc-1
      G(x+1,y+1)=(-6.2832*i*x*y);
   end
end


% Truncating Transformation matrix from Generic matrix
F=exp(G(1:N,1:N)/N);
% Calculating DFT
DFT=v*F
```

References

[1] Alan V Oppenheim. *Discrete-time signal processing*. Pearson Education India, 1999.

[2] Jean-Pierre Deschamps, Gustavo D Sutter, and Enrique Cantó. *Guide to FPGA implementation of arithmetic functions*, volume 149. Springer Science & Business Media, 2012.

[3] Duraisamy Sundararajan. *The discrete Fourier transform: theory, algorithms and applications*. World Scientific, 2001.

[4] Rajesh Kumar, Abdullah Aman Khan, Sinmin Zhang, WenYong Wang, Yousif Abuidris, Waqas Amin, and Jay Kumar. Blockchain-federated-learning and deep learning models for covid-19 detection using ct imaging. *arXiv preprint arXiv:2007.06537*, 2020.

[5] AU Usman. Dorsal hand veins based person identification. *Image Processing Theory, Tools and Applications. IEEE*, 2014.

[6] Abdullah Aman Khan, Jie Shao, Waqar Ali, and Saifullah Tumrani. Content-aware summarization of broadcast sports videos: An audio–visual feature extraction approach. *Neural Processing Letters*, pages 1–24, 2020.

[7] Rajesh Kumar, WenYong Wang, Jay Kumar, Ting Yang, Abdullah Khan, Wazir Ali, and Ikram Ali. An integration of blockchain and ai for secure data sharing and detection of ct images for the hospitals. *Computerized Medical Imaging and Graphics*, 87:101812, 2020.

[8] Abdullah Aman Khan, Haoyang Lin, Saifullah Tumrani, Zheng Wang, and Jie Shao. Detection and localization of scorebox in long duration broadcast sports videos. In *International Symposium on Artificial Intelligence and Robotics 2020*, volume 11574, page 115740J. International Society for Optics and Photonics, 2020.

[9] Saifullah Tumrani, Zhiyi Deng, Abdullah Aman Khan, and Waqar Ali. Pevr: Pose estimation for vehicle re-identification. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*, pages 69–78. Springer, 2019.

[10] Chunlin Jiang Jie Shao Abdullah Aman Khan, Saifullah Tumrani. Ricaps: Residual inception and cascaded capsule network for broadcast sports video classification. In *ACM International Conference on Multimedia in Asia*. ACM, 2021.

[11] Rehan Mehmood Yousaf, Saad Rehman, Hassan Dawood, Guo Ping, Zahid Mehmood, Shoaib Azam, and Abdullah Aman Khan. Saliency based object detection and enhancements in static images. In *International Conference on Information Science and Applications*, pages 114–123. Springer, 2017.

[12] Rajesh Kumar Amin ul Haq Abdullah Aman Khan, Sidra Shafiq. H3dnn: 3d deep learning-based detection of covid-19 virus using lungs computed tomography. In *17th International Computer Conference on Wavelet Active Media Technology and Information Processing*. IEEE, 2020.

[13] W Morven Gentleman and Gordon Sande. Fast fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, pages 563–578, 1966.