Applications of Fast Fourier Transform(FFT)

Background: In 1807 Fourier published a controversial paper which contained a theorem:Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids. It render a novel view of transform of functions and can be applied in diverse fields. One of the famous algorithm based on Fourier transform is Discrete Fourier Transform(DFT). It has extensive applications in physics, number theory, combinatorics, signal processing, probability, statistics, cryptography, acoustics, optics and other fields. And what we learned on class called FFT is an efficient implementation of DFT. In the course, we only focus on fast way to multiply and evaluate polynomials. This report is aimed at concluding some pratical application of FFT learned after class.

1.Conversion between time domain and frequency domain

Introduction: Imagine that when you are processing signals with a computer, you see the following image:

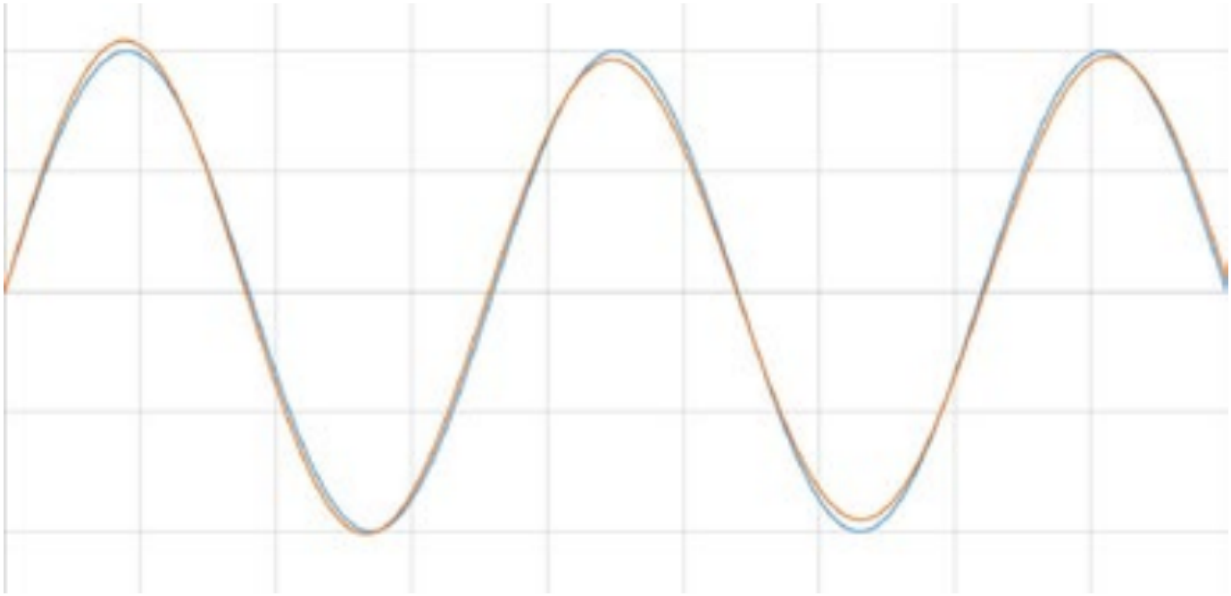


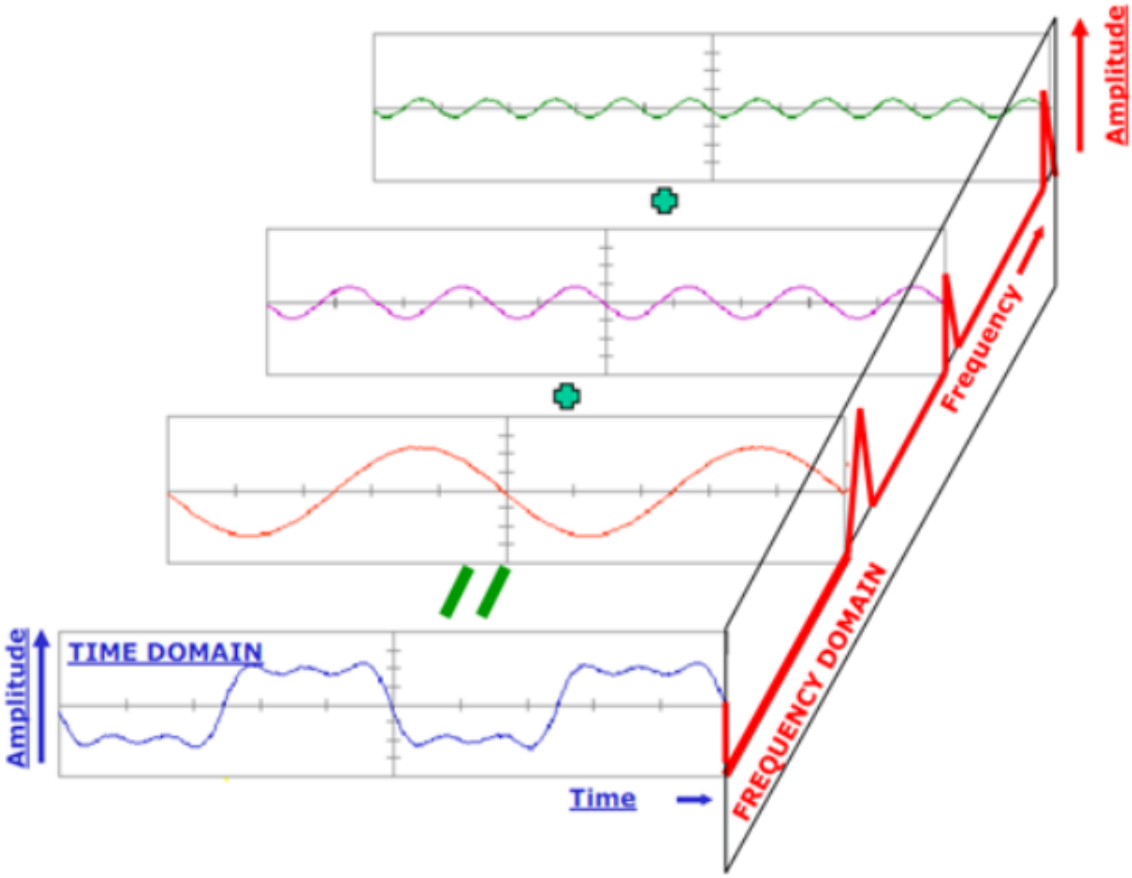
Figure 8. If these two waves were added, they would look like a perfect sine wave because they are so similar.

It's described in time domain.The goal is to analyze the signals, but you find that it seems just like a single sinusoid. That's quite confusing!

Worse still, if the goal is to filter the noise mixed in the signal you want during signal transport, then you find that you can't tell them apart in the picture above(It seems they have same amplitude and period).

So, how to solve this problem?

Basic idea: FFT serves as a tool to convert the signal between time domain and frequency domain, which can help analyze the signals.



Back to the problem above, if we use FFT to convert the signal shown in time domain to the signal shown in frequency domain, it will be easy to distinguish the noise.(Seen in the image below)

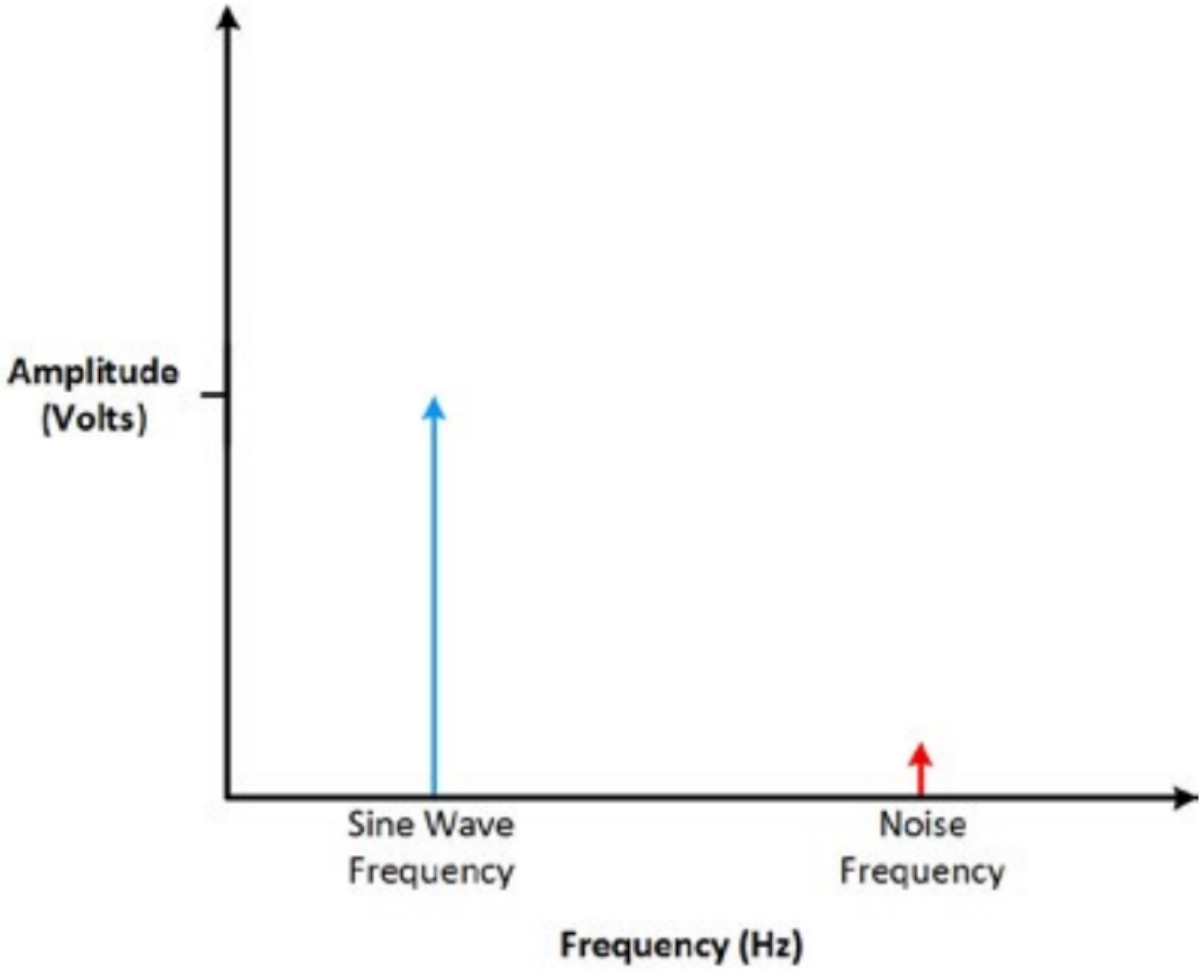
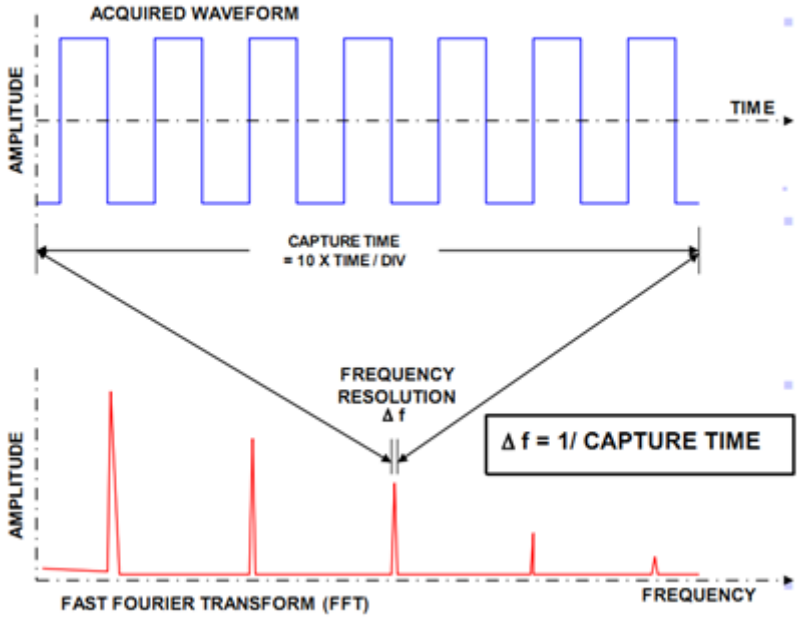


Figure 9. Looking at the seemingly perfect sine wave from Figure 8, you can see here that there is actually a glitch.

After conversion, they have distinct amplitude and frequency obviously, which lives up to our expectation. So, how do we use this powerful tool?

$$F[f(x)] = F(w) = \int_{-\infty}^{+\infty} f(x)e^{iwx}dx$$

We set f(t) as signal presented in time domain, then put it into the formula above, then the result F(w) is the signal presented in frequency domain(w: frequency, F(w): amplitude in frequency domain). The formula above is Fourier Transform and FFT is applied in speeding up the calculation of it.A more vivid demonstration is as follows:



Conclusion: Some signals are difficult to discern any features in the time domain, but if transformed into the frequency domain, it is easy. This is why many signal analysis uses FFT transform. In addition, FFT can extract the spectrum of a signal, which is also commonly used in spectrum analysis.

References:Understanding FFTs and Windowing 我所理解的快速傅里叶变换（FFT)

2.An Efficient FFT Based Neural Architecture Design

Introduction:In existing lightweight networks, whether it is the MobileNet, ShuffleNet, or NAS, 1×1 seems to be an unavoidable obstacle.The bottleneck of 1×1 refers to the computational complexity of pointwise convolutions used for channel fusions in CNN designs. These convolutions have a complexity of $O(n^2)$ with respect to the number of channels, which can be computationally expensive. So, can we find a more efficient way to compute pointwise convolutions?

Basic idea: Use butterfly transform, the main idea of FFT, to speed up pointwise convolutions from $O(n^2)$ to $O(nlogn)$.

So, how it's done?

A pointwise convolution is defined as follows:

$$Y = P(X; W)$$

This can be written as a matrix product by reshaping the input tensor X to a 2-D matrix $\hat{X} \in R^{N\times(HW)}$ with size $n\times(hw)$ (each column vector in the $\hat{X}$ corresponds to a spatial vector $X[:,i,j]$) and reshaping the weight tensor to a 2-D matrix $\hat{W} \in R^{N\times N}$ with size $n\times n$,

$$\hat{Y} = \hat{W}\hat{X}$$

where $\hat{Y}$ is the matrix representation of the output tensor Y. This can be seen as a linear transformation of the vectors in the column of $\hat{X}$ using $\hat{W}$ as a transformation matrix. The linear transformation is a matrix-vertor product and its complexity is $O(n^2)$. But we can use BFT to implement it in $O(nlogn)$.

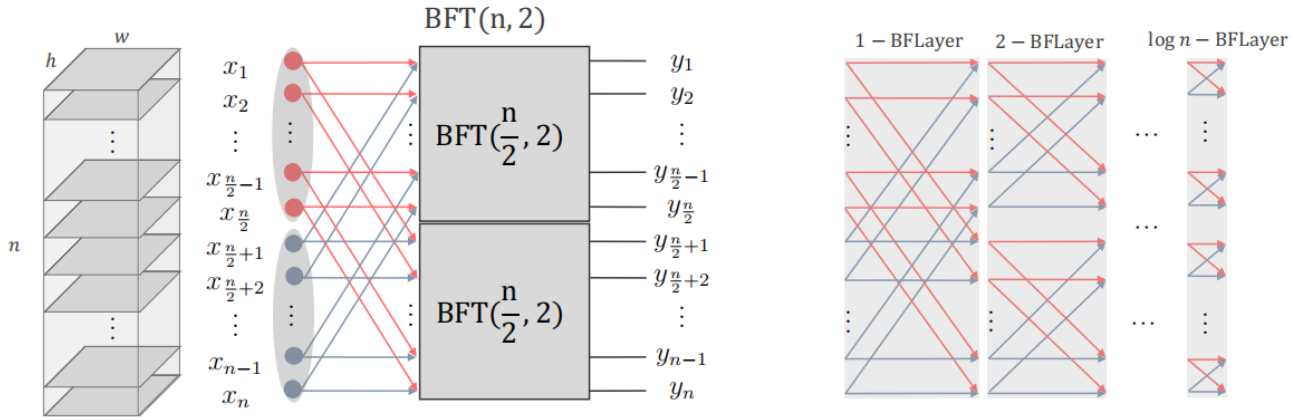


Figure 2: **BFT Architecture:** This figure illustrates the graph structure of the proposed Butterfly Transform. The left figure shows the recursive procedure of the BFT that is applied to an input tensor and the right figure shows the expanded version of the recursive procedure as $\log n$ Butterfly Layers in the network.

Channel Fusion through BFT: The function of Pointwise convolution is to fuse channel features and convert them into sequence operation combinations. In first layer, we partition input channels and output channels to k parts with size $\frac{n}{k}$ each, named $(x_1, x_2, x_3 \dots x_k)$ and $(y_1, y_2, y_3 \dots y_k)$ respectively. Then combine x_i with y_i with $\frac{n}{k}$ parallel edges D_{ij} , and recursively fuse information like this in next layer.

Translating what we do above, we get a transformation matrix called butterfly matrix shaped like this:

$$\mathbf{B}^{(n,k)} = \begin{pmatrix} \mathbf{M}_1^{(\frac{n}{k},k)}\mathbf{D}_{11} & \dots & \mathbf{M}_1^{(\frac{n}{k},k)}\mathbf{D}_{1k} \\ \vdots & \ddots & \vdots \\ \mathbf{M}_k^{(\frac{n}{k},k)}\mathbf{D}_{k1} & \dots & \mathbf{M}_k^{(\frac{n}{k},k)}\mathbf{D}_{kk} \end{pmatrix}$$

Furthermore, according to analysis, $\hat{Y} = \hat{W}\hat{X}$ can be represented in the format of linear transformation like this:

$$\mathbf{B}^{(n,k)}\mathbf{x} = \begin{pmatrix} \mathbf{M}_1^{(\frac{n}{k},k)} \sum_{j=1}^{k} \mathbf{D}_{1j}\mathbf{x}_j \\ \vdots \\ \mathbf{M}_i^{(\frac{n}{k},k)} \sum_{j=1}^{k} \mathbf{D}_{ij}\mathbf{x}_j \\ \vdots \\ \mathbf{M}_k^{(\frac{n}{k},k)} \sum_{j=1}^{k} \mathbf{D}_{kj}\mathbf{x}_j \end{pmatrix} = \begin{pmatrix} \mathbf{M}_1^{(\frac{n}{k},k)}\mathbf{y}_1 \\ \vdots \\ \mathbf{M}_i^{(\frac{n}{k},k)}\mathbf{y}_i \\ \vdots \\ \mathbf{M}_k^{(\frac{n}{k},k)}\mathbf{y}_k \end{pmatrix}$$

where $y_i = \sum_{j=1}^{k} D_{ij}x_j$. The correspoding algorithm described in pseudo code is shown below:

Algorithm 1: Recursive Butterfly Transform

Function `ButterflyTransform(W, X, n):` ;
`/* algorithm as a recursive function */`

Data: W Weights containing $2n\log(n)$ numbers
Data: X An input containing n numbers
if $n == 1$ **then**
  return $[X]$;
Make $D_{11}, D_{12}, D_{21}, D_{22}$ using first $2n$ numbers of W;
Split rest $2n(\log(n) - 1)$ numbers to two sequences W_1, W_2 with length $n(\log(n) - 1)$;
Split X to X_1, X_2;
$y_1 \longleftarrow D_{11}X_1 + D_{12}X_2$;
$y_2 \longleftarrow D_{21}X_1 + D_{22}X_2$;
$My_1 \longleftarrow$ `ButterflyTransform`$(W_1, y_1, n-1)$;
$My_2 \longleftarrow$ `ButterflyTransform`$(W_2, y_2, n-1)$;
return $Concat(My_1, My_2)$;

Conclusion: It's a novel idea to exploit FFT to replace pointwise convolutions in various neural architectures to reduce the computation while maintaining accuracy.

References:Butterfly Transform: An Efficient FFT Based Neural Architecture Design

---

3.A variant FFT algorithm: FFT Calculations using Prime Factor Algorithm

Introduction: Famous FFT algorithm raised by Cooley-Turkey has implemented an efficient calculation of DFT in $O(nlogn)$, but we find that a lot of memory references are used up over and over again to load the identical twiddle factors for the different stages of the butterfly diagram as a result of which the computational time tends to increase. And implementing FFT on digital signal processors(DSP) involves many memory references to access butterfly inputs and twiddle factors, which causes high power consumption for the execution of FFT. So, how do we improve it?

Basic idea: Use Type-1 index map from Multidimensional Index Mapping to implement Prime Factor Algorithm of FFT, which includes no butterfly diagram and reduces the number of twiddle factors.

So, how it's done? Let's focus on the principle of prime factor and how the mapping works.

Suppose $N = N_1N_2$and $gcd(N_1, N_2) = 1$,let $n = n_1N_2 + n2N_1(mod \quad N)$, according to Bezout's Theorem, for any input n we can find $n_1$andn_2 to satisfy the equation. And output satisfies the equation below:

$$k = k_1 \mod N_1$$

$$k = k_2 \mod N_2$$

According to The Chinese Remainder Theorem we get:

$$k = k_1 N_2^{-1} N_2 + k_2 N_1^{-1} N_1 \mod N$$

where k_1 and k_2 are unique.Then we take this map method into the DFT:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \ldots, N-1$$

Derived equation is shown below:

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} x_{n_1 N_2 + n_2 N_1} e^{-\frac{2\pi i}{N_2} n_2 k_2} \right) e^{-\frac{2\pi i}{N_1} n_1 k_1}.$$

Compare it with Cooley-Turkey's FFT if we do the same mapping on it:

$$X_{k_1 N_2 + k_2} = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} x_{n_1 + n_2 N_1} e^{-\frac{2\pi i}{N_2} n_2 k_2} \right) e^{-\frac{2\pi i}{N} n_1 k_2} e^{-\frac{2\pi i}{N_1} n_1 k_1}$$

It's obvious that we reduce the times of multiplication of twiddle factor $e^{-\frac{2\pi i}{N} n_1 k_2}$.

The pseudo code of algorithm is shown below:

1. Take input in the form of x(n) of N-points.
2. Factorize the N-point into N_1 and N_2 such that $N=N_1N_2$.
3. Define n_1 and n_2 as ;
   $n_{1=}0, 1, 2, 3\ldots..N_1\text{-}1$
   $n_{2=}0, 1, 2, 3\ldots..N_2\text{-}1$
4. Map n_1 and n_2 to n as
   $n = ((K_1n_1 + K_2n_2))\ N$; where K_1 are integers.
5. If N_1 and N_2 are relatively prime than go to step 7 or step 8.
6. Choose $K_1=aN_2$ and/or $K_2=bN_1$ and take $(K_1N_1)=(K_2N_2)=1$ where a and b are integers. Go to step 9.

7. Choose $K_1=aN_2$ and $K_2\neq bN_1$ and $(a,N_1)=(K_2N_2)=1$
OR
$(K_1\neq aN_2)$and$(K_2=bN_1)$and$(K_1,N_1)=(b,N_2)=1$.
Go to step 10.
8. Type one index map. Take $K_1=aN_2$ and $K_2=bN_1$. Go to step 11.
9. Type two index map. Take $K_1=aN_2$ or $K_2=bN_1$ but not both.
10. Determine the DFT of the given N-point sequence using the formula
C(k)=

$$\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1} x(n)\, W_N^{K_1K_3n_1n_2}\, W_N^{K_1K_4n_2}\, W_N^{K_2K_3n_1}\, W_N^{K_2K_4n_2} \qquad (6)$$

11. If N_1 and N_2 are relatively prime then apply the uncoupling constraints as:
$(K_1K_4)\ N=0$ and $(K_2K_3)\ N=0$.
12. If N_1 and N_2 are not relatively prime then apply the uncoupling constraints as:
$(K_1K_4)N=0$ or $(K_2K_3)N=0$.
13. Using the required decoupling constraints find C(k), DFT of the required sequence.
14. Display the result.

Conclusion: PFA serves as the supplement to FFT which reduces the times of multiplication and saves memory, but it's simultaneously difficult to do prime factor. Actually, we can combine PFA with Cooley-Turkey's FFT, which needs furthur exploring.

References:Efficient methods for FFT calculations using prime factor algorithm

---

4.Numerical solutions to Poisson's equation.

Introduction:Poisson's equation is a partial differential equation commonly used in mathematics in electrostatics, mechanical engineering, and theoretical physics. It's commonly used in electrostatics, for example:

If there is a three-dimensional spherically symmetric Gaussian distribution of charge density $\rho(r)$:

$$\rho(r) = \frac{Q}{\sigma^3\sqrt{2\pi}^3}e^{-r^2/(2\sigma^2)}$$

Q is the total charge, so the poisson's equation here is $\nabla^2\Phi = -\frac{\rho}{\varepsilon_0}$, and the solution $\Phi(r)$ is

$$\Phi(r) = \frac{1}{4\pi\varepsilon_0}\frac{Q}{r}erf(\frac{r}{\sqrt{2}\sigma})$$

$erf(x)$ represents the error function.

So, how do we solve Poisson's equation? The common solution is given by Green's function, but we can also use FFT to work it out.

Basic idea: Actually it's DFT that gives a numerical solution to Poisson's eqution and what FFT does is to accelerate it.

The poisson's equation in general is like this:

$$\nabla^2\varphi = f$$

which can be described in detail like this:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f$$

If we do FFT on one column of the coefficient matrix, we get:

$$u_{j,k} = \sum_{m=0}^{N-1} U_{m,k}e^{im\pi j\Delta x}$$

$$and$$

$$f_{j,k} = \sum_{m=0}^{N-1} F_{m,k}e^{im\pi j\Delta x}$$

Correspondingly, doing IFT we get:

$$U_{j,k} = \frac{1}{N}\sum_{m=0}^{N-1} u_{m,k}e^{-im\pi j\Delta x}$$

$$and$$

$$F_{j,k} = \frac{1}{N}\sum_{m=0}^{N-1} f_{m,k}e^{-im\pi j\Delta x}$$

The poisson's equation can be converted into the format like this:

$$aU_{j,k} + \frac{\partial^2 U_{j,k}}{\partial^2} = F_{j,k} \quad if \frac{\partial^2 u_{j,k}}{\partial x^2} = aU_{j,k}$$

Then do difference on the second item, take the coefficient of $U_{j-1,k}$ as b then we get:

$$bU_{j-1,k} + (a-2b)U_{j,k} + bU_{j+1,k} = F_{j,k}$$

There are three unknowns in the equation, a total of N unknowns in a column, which can be written as a system of tridiagonal linear equations, for example:

$$\begin{bmatrix} (a-2b) & b & & \\ b & (a-2b) & b & \\ & b & (a-2b) & b \\ & & b & (a-2b) \end{bmatrix} \begin{bmatrix} U_{0,k} \\ U_{1,k} \\ U_{2,k} \\ U_{3,k} \end{bmatrix} = \begin{bmatrix} F_{0,k} \\ F_{1,k} \\ F_{2,k} \\ F_{3,k} \end{bmatrix}$$

The we do IFT to get the solution of the poisson's equation.

Conclusion: FFT provides convenience for solving Poisson's equations using DFT to some degree.

Reference:泊松方程,用FFT解泊松方程

5.Efficient FFT Implementation on a CGRA

Introduction:With the rapid development of integrated circuits, computer technology and software technology, the implementation of digital signal applications has been changing rapidly. Can we implement digital signal processing applications with high performance and meanwhile certain flexibility? We focus on the implementation of the Fast Fourier Transform (FFT) algorithm on a reconfigurable architecture called Coarse-Grained Reconfigurable Array (CGRA) using the radix-4 method.And we conclude that the proposed FFT implementation on the CGRA has performance advantages compared to other similar reconfigurable architectures.

Basic idea:The CGRA based FFT implementation optimizes the performance by using the parallelism of processing elements (PE) and the multiple access scheme of shared memory (SM).

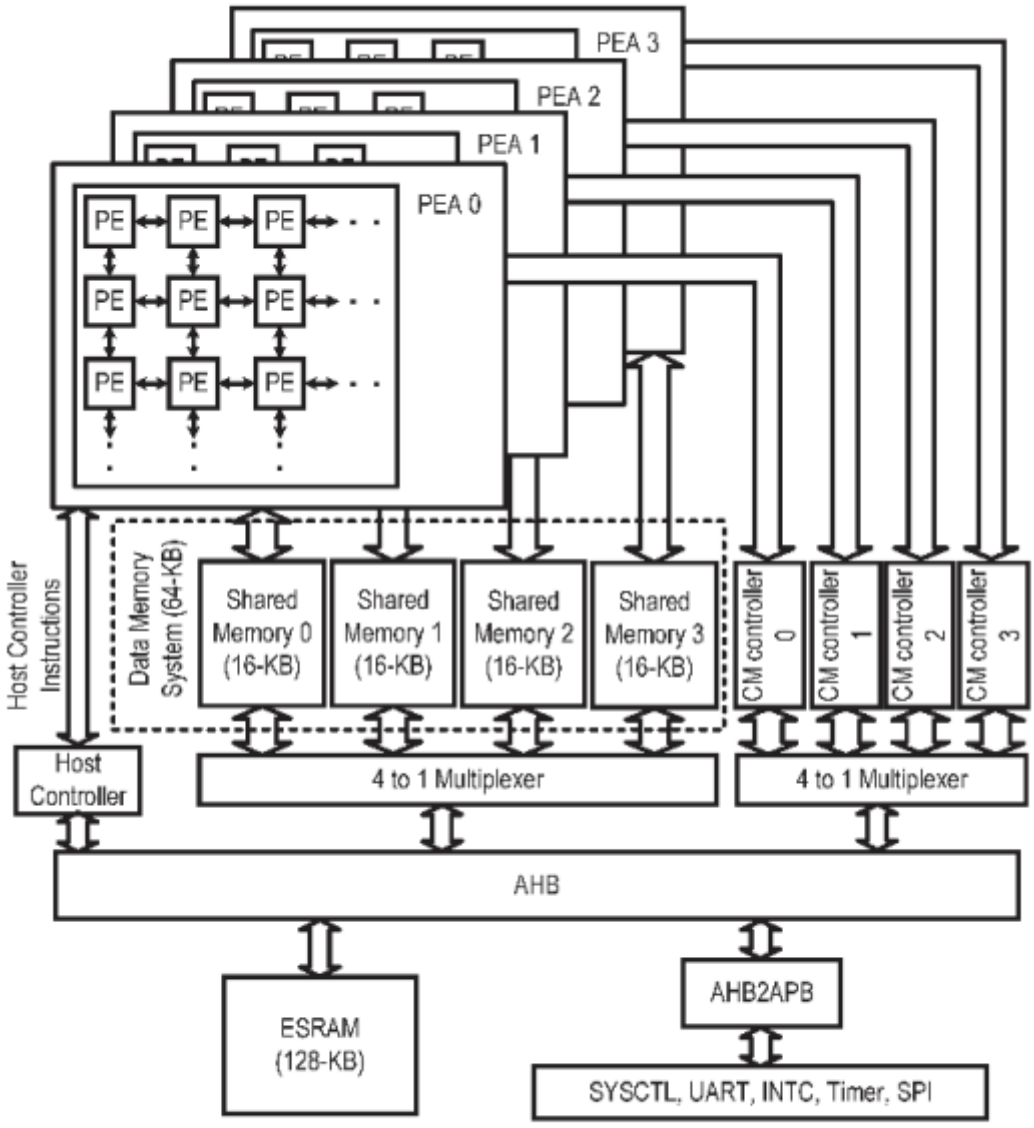


Figure 1. Overview of the CGRA.

How does it works?

First, what's radix-4? Actually according to the number of input-data in a butterfly operation unit, the implementations of FFT can be divided into methods such as radix-2, radix-4, ... and radix 2^m. The intuitive comparison between radix-2 and radix-4 is shown below:

Table 1. Computation comparison of Radix-2 and Radix-4

Points N	Radix-2	Radix-4
256	2^8	4^4
1024	2^{10}	4^5
16K	2^{14}	4^8
64K	2^{16}	5^8
Stages	L	$L/2$
Total Multiplication	(2L-4)N+4	(1.5L-4)N+4
Total Addition	(3L-2)N+2	(2.75L-2)N+2

L is defined as $L = log_2 N$.Through the comparative analysis of the data in the table, it can be found that from the perspective of the calculation amount, the higher the base number, the less the calculation amount, and the shorter the calculation time. The number of real multiplications in radix-4 implementation is reduced by about 30%. So, we choose radix-4 to implement FFT.

Next work to do is mapping radix-4 butterfly unit.

Using radix-4, DFT can be desribed like this:

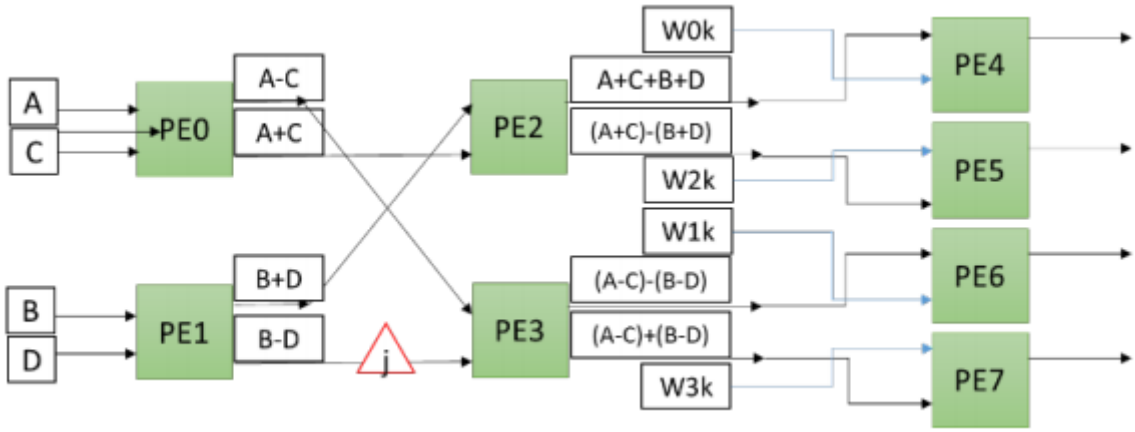
$$X(k) = \left[x(k) + x\left(k + \frac{N}{4}\right) + x\left(k + \frac{N}{2}\right) + x\left(k + \frac{3}{4N}\right)\right] * W_N^{\square}$$

$$X(k + N/4) = \left[x(k) - jx\left(k + \frac{N}{4}\right) - x\left(k + \frac{N}{2}\right) + jx(k + \frac{3}{4N})\right] * W_N^{1k}$$

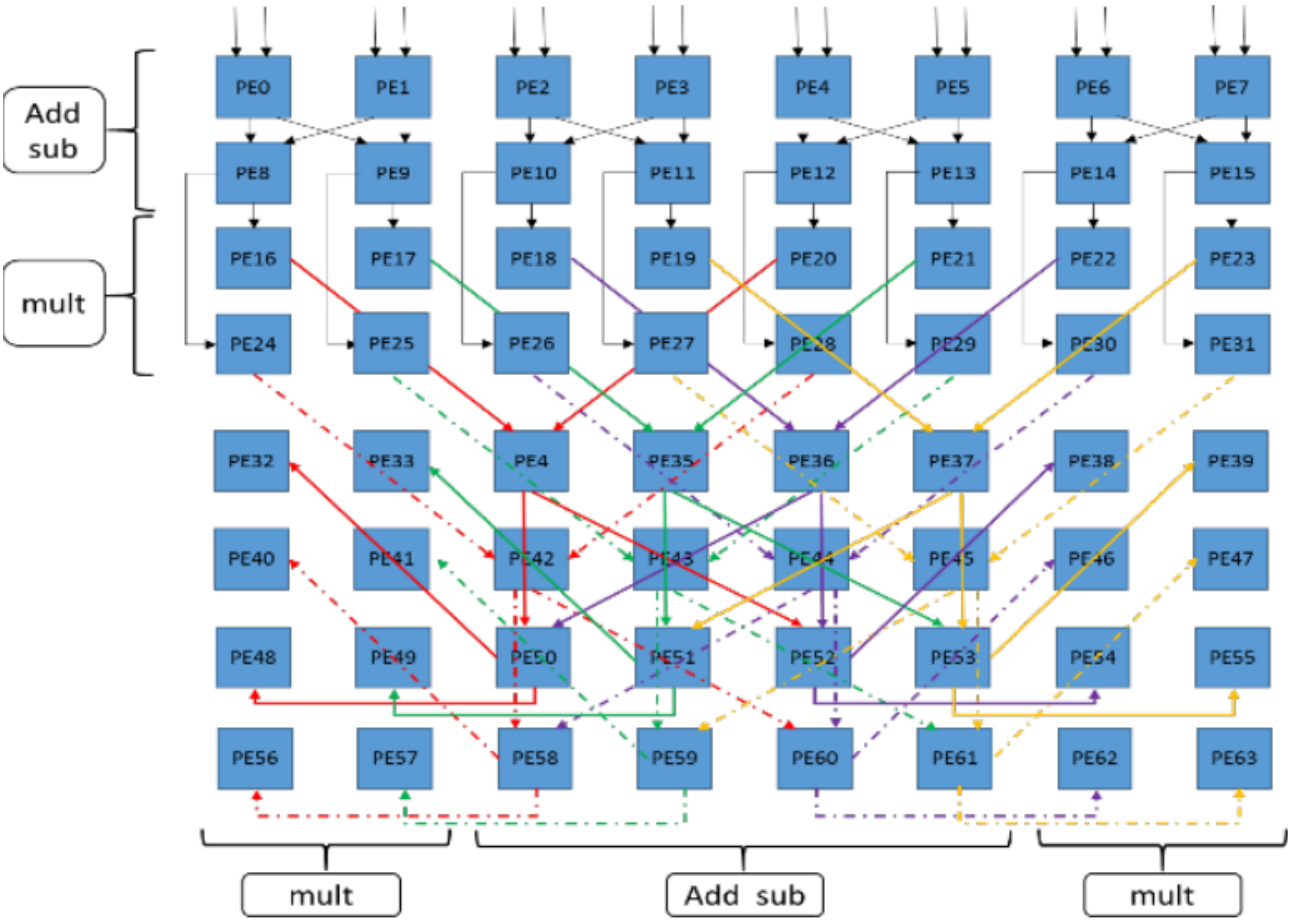
$$X(k + N/2) = \left[x(k) - x\left(k + \frac{N}{4}\right) + x\left(k + \frac{N}{2}\right) - x(k + \frac{3}{4N})\right] * W_N^{2k}$$

$$X(k + 3/4N) = \left[x(k) + jx\left(k + \frac{N}{4}\right) - x\left(k + \frac{N}{2}\right) - jx(k + \frac{3}{4N})\right] * W_N^{3k}$$

According to the above formula, the image below shows the mapping of a radix-4 butterfly unit on the reconfigurable array. In this figure, we define $A = x(k)$, $B = x(k + N/4)$, $C = x(k + N/2)$ and $D = x(k + 3N/4)$.PE0 to PE3 is configured as addition function and PE4 to PE 7 is configured as multiplication. A Radix-4 Butterfly Unit(BU) requires 8 PEs. Since PE is a single-cycle operation function unit, the calculation time of this butterfly operation unit is 3 clock cycles.



There are 64 PEs in the computing array, and 8 butterfly units can be mapped on the array in parallel. And a possible arrangement of 8 butterfly operation units on the array is shown below:



Conclusion: A radix-4 FFT implantation mapped to CGRA make a success with improvements of parallelism of the Processing Elements (PEs) and the multi-access scheme of the shared memory (SM). And the proposed FFT implementation on the CGRA has performance advantages compared to other similar reconfigurable architectures.

References:Efficient FFT Implementation on a CGRA

Summary: Actually the significant advantage of low computational complexity has led to the widespread application of FFT in the field of signal processing technology, which can be combined with high-speed hardware to achieve real-time signal processing. For example, FFT is used in the analysis and synthesis of speech signals, the multiplexing and conversion of fully digitized time division and frequency division (TDM/FDM) systems in communication systems, signal filtering and correlation analysis in the frequency domain, and spectral analysis of radar, sonar, and vibration signals to improve the resolution of target search and tracking. Besides, it is also widely used in diverse fields of science, what I list and introduce above is just a drop in the bucket. Wish I have a deeper insight into it in my future research study.