


Digital Signal Processing
Prof. C.S. Ramalingam
Department Electrical Engineering
Indian Institute of Technology, Madras

Lecture 82:
The Discrete Fourier Transform (4)
- The FFT algorithm

(Refer Slide Time: 00:16)

(Refer Slide Time: 00:18)


Introduction



- FFT is an **efficient algorithm for computing the DFT coefficients**
- It is a cornerstone of modern signal processing
- Filtering and correlation can be computed quickly and efficiently via the FFT
 - $x_1[n] \otimes x_2[n] \leftrightarrow X_1[k] X_2[k]$
 - It is more efficient to compute circular convolution using the FFT as follows:

$$y[n] = \text{DFT}^{-1} (X_1[k] X_2[k])$$

- Structure of IDFT very similar to that of DFT
 - Efficient, FFT-type algorithm exists for IDFT also



C.S. Ramalingam (EE Dept., IIT Madras)
Intro to FFT 2 / 31

Let us now, go to the very last topic in this DFT module, namely the FFT algorithm. So, FFT stands for the Fast Fourier transform; it is an efficient algorithm for computing the DFT coefficients. And, it is the cornerstone of modern signal processing, filtering and correlation happened all the while and until the FFT algorithm came along, we are going to look at the operation count; this was turning to be extremely cumbersome and what our methods they knew at that time was really slow.

In 1965, the FFT algorithm was discovered and that really changed upside down signal processing, which was until then a curiosity not of much interest, because it did not have much practical use. It did not have much practical use, because the implementation turned out to be extremely computationally burdensome.

And, the reason why this cost a huge change, remember, convolution the DFT framework is circular convolution. Circularly convolved in the time domain, you multiply in the transform domain and it is more efficient to carry out convolution by taking the transforms multiplying them and then taking the inverse transform. So, this is no different from using log for multiplication. Addition is easier than multiplication and if you want to multiply numbers, you use log; you take log of each number add them up and then take anti log.

Similarly, if you want to convolve, take the transforms, multiplying the transformed domain and then take the inverse transform. And fortunately, the structure of the inverse Discrete Fourier transform is very similar to that of the DFT and hence the efficient algorithm that was discovered for the forward transform, applied equally well with very little change to the inverse transform as well.

So, this really turned upside down, what is happening in signal processing. And, that is, entirely due to the discovery of the Fast Fourier transform algorithm. I had mentioned that, this happened in 1965, Cooley Tukey, An algorithm for machine computation of complex Fourier series.

(Refer Slide Time: 02:46)


The Discrete Fourier Transform (DFT)

- DFT of an N -point sequence $x_n, n = 0, 1, 2, \dots, N - 1$ is defined as

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi k}{N}n} \quad k = 0, 1, 2, \dots, N - 1$$

- An N -point sequence yields an N -point transform
- X_k can be expressed as an *inner product*:


$$X_k = \begin{bmatrix} 1 & e^{-j\frac{2\pi k}{N}} & e^{-j\frac{2\pi k}{N}2} & \dots & e^{-j\frac{2\pi k}{N}(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$



C.S. Ramalingam (EE Dept., IIT Madras)

Intro to FFT

3 / 31



So, this first few slides are a very brief review of the DFT. So, this is what the DFT definition is. So, $X[k]$ is this expression and we saw that an N point sequence yields N point transform and we also saw that this can be expressed as an inner product; the k th DFT coefficient can be expressed as an inner product as shown here.

(Refer Slide Time: 03:07)

The Discrete Fourier Transform (DFT)



- Notation: $W_N = e^{-j\frac{2\pi}{N}}$. Hence,

$$X_k = \begin{bmatrix} 1 & W_N^k & W_N^{2k} & \dots & W_N^{(N-1)k} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

- By varying k from 0 to $N - 1$ and combining the N inner products, we get the following:

$$\mathbf{X} = \mathbf{W}\mathbf{x}$$

- \mathbf{W} is an $N \times N$ matrix, called as the "DFT Matrix"



And then, if we use the notation $W_N = e^{-j2\pi/N}$, this inner product can be expressed in terms of this notation as follows. And if you vary k from 0 to $N - 1$, we get the matrix notation that was introduced while discussing DFT. And, this W is of course called as the DFT matrix which is just given by this.

(Refer Slide Time: 03:27)

The DFT Matrix



$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)(N-1)} \end{bmatrix}_{N \times N}$$

- The notation \mathbf{W}_N is used if we want to make the size of the DFT matrix explicit



And, as I had mentioned earlier, the suffix N is used if you want to make the size of this matrix explicit.

(Refer Slide Time: 03:38)

How Many Complex Multiplications Are Required?



- Each inner product requires N complex multiplications
 - There are N inner products
- Hence we require N^2 multiplications
- However, the first row and first column are all 1s, and *should not be counted as multiplications*
 - There are $2N - 1$ such instances
- Hence, the number of complex multiplications is $N^2 - 2N + 1$, i.e., $(N - 1)^2$

Now, I had mentioned that the computational burden is quite large. To get a feel for that, each inner product requires N complex multiplications and there are N such inner products. And, hence we require N square multiplications.

But if you look at the DFT matrix, the first row in the first column which corresponds to $N = 0$ and $k = 0$, they are all 1s. Therefore, multiplication by 1 is really not a multiplication. And, there are $2N - 1$ such instances and hence, the number of multiplications is really $N^2 - 2N + 1$; which is $(N - 1)^2$. So, this is the estimate of the complex multiplications involved.

(Refer Slide Time: 04:22)

How Many Complex Additions Are Required?




- Each inner product requires $N - 1$ complex additions
 - There are N inner products
- Hence we require $N(N - 1)$ complex additions


And, each inner product requires $(N - 1)$ complex additions and there are N such inner products. And, hence we require $N(N - 1)$ complex additions. So, we have $(N - 1)^2$ complex multiplications and $N(N - 1)$ complex additions.

(Refer Slide Time: 04:46)

Total Operation Count



- No. of complex multiplications: $(N - 1)^2$
- No. of complex additions: $N(N - 1)$
- The operation count for multiplications and additions assumes that W_N^k has been computed offline and is available in memory
 - If pre-computed values of W_N^k are not available, then the operation count will increase
- We will assume that all the required W_N^k have been pre-computed and are available




C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 8 / 31


So, this is the operation count. So what this assumes is, these W_N^k , these have been pre computed and already available for you to use for multiplication. So, if they are not pre computed, then more overhead is involved. So, what is normally assumed and which is what that happens in practice is, these have already been pre computed and are indeed available.

(Refer Slide Time: 05:34)

Operation Count Makes DFT Impractical



- For large N ,
 $(N - 1)^2 \approx N^2$
 $N(N - 1) \approx N^2$
- Hence both multiplications and additions are $O(N^2)$
- If $N = 10^3$, then $O(N^2) = 10^6$, i.e., a million!
- This makes the straightforward method **slow** and **impractical** even for a moderately long sequence



C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 9 / 31

And, for large N , $(N - 1)^2$ is roughly N^2 and $N(N - 1)$ can be approximated as is N^2 . And hence, both multiplications and additions are of the order of N^2 . And, if N is, were to be 1000 which is really not an unreasonable number, then the number of operations that are required is 10^6 which is million.

So, this makes this straight forward computation really slow and impractical, even for moderately long sequences. And this was the stumbling block that made this not progress until the FFT algorithm was discovered, because of this huge computational burden.

(Refer Slide Time: 06:24)

The Divide and Conquer Approach



- Suppose N is even and we split the sequence into two halves.
 - Each sequence has $N/2$ points
- Suppose we compute the $\frac{N}{2}$ point DFT of each sequence
 - Multiplications: $2 \times \left(\frac{N}{2}\right)^2 = \frac{N^2}{2}$
- Suppose we are able to combine the individual DFT results to get the originally required DFT
 - Some computational overhead will be consumed to combine the two results
- If $\frac{N^2}{2} + \text{overhead} < N^2$, then this approach will reduce the operation count



C.S. Ramalingam (EE Dept., IIT Madras)

Intro to FFT

10 / 31

Then came the divide and conquer approach. Suppose, N is even and we split the sequence into 2 halves, each sequence has $N/2$ points. And suppose, we compute the $N/2$ point DFT of each sequence, the length is N , the computational burden is N^2 . If the length is $N/2$, the computational burden will be $(N/2)^2$, but now you have 2 such sequences.

So, it is $N^2/2$ number of multiplications. And remember, you are really interested in the DFT of the whole sequence, not of the individual sequences. So, presumably there must be sum overhead to combine these 2 individual DFTs. First of all, we do not even know whether they can be combined. But assume they can be combined, surely it is not going to come for free, there is going to be sum overhead.

So, the computational burden now, the new computational burden is $(N^2/2) + \text{overhead}$. And if this turns out to be less than N^2 , then this approach will reduce the computational burden, otherwise there is no use.

(Refer Slide Time: 07:36)

The Divide and Conquer Approach



Let $N = 8$

- Straightforward implementation requires, *approximately*, 64 multiplications
- The "divide and conquer" approach requires, *approximately*, $2 \times \left(\frac{8}{2}\right)^2 + \text{overhead}$, i.e., 32 + overhead multiplications
- Questions:
 - Can the two DFTs be combined to get the original DFT?
 - If so, how? What is the overhead involved?
 - Will 32 + overhead be less than 64?



C.S. Ramalingam (EE Dept., IIT Madras)

Intro to FFT


11 / 31

So, to get a feel for numbers, if $N = 8$, straight forward implementation approximately requires 64 and the divide and conquer approach is $2 \times (N/2)^2 + \text{overhead}$. So, $2 \times (N/2)^2$ for $N = 8$ is 32. So, $32 + \text{overhead}$ is what you need? So, what are the questions, can the 2 DFTs we combined to get the original DFT? Remember now, we are talking about the DFT of the 2 halves.

So, whether they can be combined at all, we do not know yet. And if so, how and what is the overhead involved? So, in this particular case, will $32 + \text{overhead}$ be less than 64? So, this is what the problem at hand is.

(Refer Slide Time: 08:21)

The Decimation in Time (DIT) Algorithm




- From $\{x_n\}$ form two sequences as follows:

$$\{g_n\} = \{x_{2n}\} \quad \{h_n\} = \{x_{2n+1}\}$$
- $\{g_n\}$ contains the even-indexed samples, while $\{h_n\}$ contains the odd-indexed samples
- The DFT of $\{x_n\}$ is

$$\begin{aligned}
 X_k &= \sum_{n=0}^{N-1} x_n W_N^{nk} \\
 &= \sum_{r=0}^{\frac{N}{2}-1} x_{2r} W_N^{(2r)k} + \sum_{r=0}^{\frac{N}{2}-1} x_{2r+1} W_N^{(2r+1)k} \\
 &= \sum_{r=0}^{\frac{N}{2}-1} g_r W_N^{(2r)k} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} h_r W_N^{(2r)k}
 \end{aligned}$$

C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 12 / 31



So, we look at the decimation in time. Given the sequence $\{x_n\}$, we form 2 sub sequences, we form the sequences $\{g_n\}$ and $\{h_n\}$, where g is the even indices of the original sequence and then h is the sequence formed by the odd indices; $2n$ and $2n + 1$.

Now, let us look at the DFT is the original sequence. So, X_k is our usual definition, where I have used the W notation here. And, then we split this summation from 0 to $N - 1$ to those involving the even indices and those involving the odd indices. And, x_{2r} is nothing but g_r , x_{2r+1} is h_r . And this $W_N^{(2r+1)k}$, you can take out W_N^k outside, because this does not involve the index of summation. So, this is what we have, this is just simple straightforward manipulation; splitting it or even and odd indices. And now, let us look at $W_N^{(2r)k}$.

The Decimation in Time (DIT) Algorithm



- But,

$$W_N^{2rk} = e^{-j\frac{2\pi}{N}(2rk)} = e^{-j\frac{2\pi}{N/2}(rk)} = W_{N/2}^{rk}$$

and hence

$$\begin{aligned} X_k &= \sum_{r=0}^{N/2-1} g_r W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} h_r W_{N/2}^{rk} \\ &= G_k + W_N^k H_k \quad k = 0, 1, \dots, N-1 \end{aligned}$$

- $\{G_k\}$ and $\{H_k\}$ are $\frac{N}{2}$ point DFTs
- The overhead for combining the two $\frac{N}{2}$ point DFTs is the multiplicative factor W_N^k for $k = 0, 1, \dots, N-1$
- W_N^k is called "twiddle factor"



If you look at this, from the basic definition, you get this; $W_N^{(2r)k}$ is, replace W_N by $e^{j(\cdot)}$, you get this. And then you notice that, you can rewrite this as $e^{j\frac{2\pi}{N/2}(rk)}$. And in this form, you recognize that this is nothing but $W_{N/2}^{rk}$. So, $W_N^{(2r)k} = W_{N/2}^{rk}$.

Once you realize this, then the first term is nothing but the $N/2$ point DFT of g_n , G_k . The second term is nothing but the $N/2$ point DFT of the sequence h_n , H_k . So, these are valid DFTs by themselves and to get the final transform, you need to add these individual DFTs and this is an extra term that is present.

And, k goes from 0 to $N-1$, because that is what the original index k went from. But remember, G_k and H_k are $N/2$ point DFTs. N point DFT is periodic with period N , $N/2$ point DFT is periodic with period $N/2$. Therefore, when you let k go from 0 to $N-1$, from 0 to $(N/2)-1$, you get G_k . And, then when k hits $N/2$, you are back to g_0 , because this is periodic with period $N/2$.

Similarly, for H_k ; when the k index goes from 0 to $N-1$, beyond $(N/2)-1$ that is the first index beyond $(N/2)-1$ is $N/2$, $k = N/2$ is the same as $k = 0$. Therefore, we have answered all our questions, that is, it is possible to combine the 2 DFTs. Yes, there is the overhead involved and the overhead involved is the multiplicative factor W_N^k . And, this is called as the twiddle factor, this is all terminology based on literature of those days.

(Refer Slide Time: 11:44)

The Decimation in Time (DIT) Algorithm



- The $N/2$ point DFTs $\{G_k\}$ and $\{H_k\}$ are periodic with period $N/2$
 - $G_{k+N/2} = G_k$
 - $H_{k+N/2} = H_k$
- $W_N^{k+N/2} = -W_N^k$
- Hence, if $X_k = G_k + W_N^k H_k$, then $X_{k+N/2} = G_k - W_N^k H_k$
 - $W_N^k H_k$ needs to be computed only once for $k = 0$ to $\frac{N}{2} - 1$
- Thus, the multiplication overhead due to the twiddle factors is only $\frac{N}{2}$

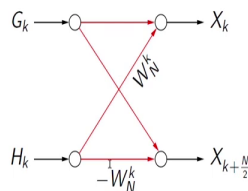


And, so, this is exactly what I had explained just now. That is, G_k and H_k are periodic with period $N/2$, and $W_N^{k+(N/2)} = -W_N^k$. If you replace k by $k + (N/2)$, all you will do is a multiplicative factor of -1 jumps out.

Therefore, if you want to form X_k over all k , then you need to compute this product which appears in the second term only from 0 to $(N/2) - 1$, because from $N/2$ to $N - 1$, this second term is just the negative of the previous term. Therefore, this overhead applies only for half the indices. So, this needs to be computed only from k going from 0 to $(N/2) - 1$. So, the multiplication overhead due to the twiddle factors are only $N/2$.

(Refer Slide Time: 12:50)

Butterfly Diagram



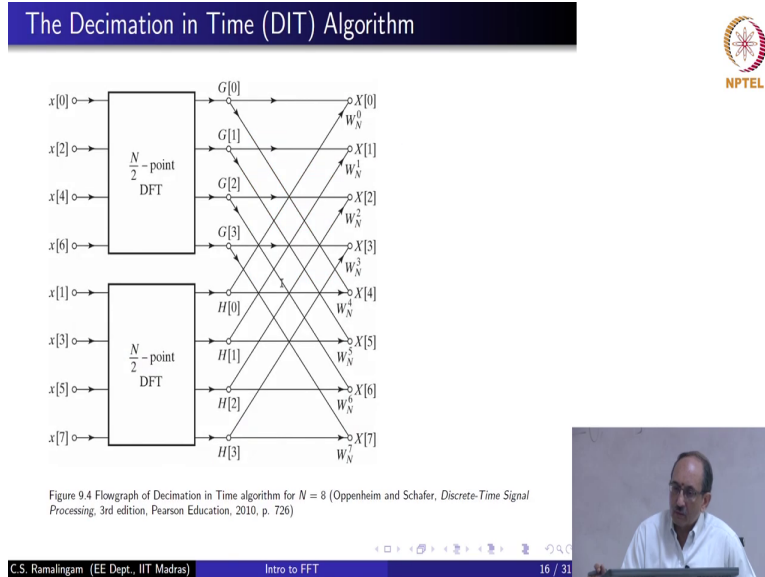
- $X_k = G_k + W_N^k H_k$
- $X_{k+N/2} = G_{k+N/2} + W_N^{k+N/2} H_{k+N/2}$
 $= G_k - W_N^k H_k$



And, this is what is called as the butterfly diagram, because this term looks like the wings of a butterfly. So, what this says is, to compute X_k , you take G_k and whenever an arrow takes off from a node, it is same value gets repeated on both nodes. When 2 arrows converge on a node, it is summation.

So, this is a branch node this is summation node. Therefore, what this says is, $X_k = G_k + W_N^k H_k$. Similarly, $X_{k+(N/2)} = G_k - W_N^k H_k$. So, whenever there is a number associated to the branch, that means weight of that branch. So, this captures these 2 equations. And, this is what is called as the decimation in time algorithm.

(Refer Slide Time: 13:57)




So, this is for a 4 point sequence or rather 8 point sequence. So, divide this into even indices and odd indices, so, 0, 2, 4, 6 and 1, 3, 5, 7 and then combine them. So, $G_0 + H_0$ times this twiddle factor and so on. And, remember when k goes beyond $(N/2) - 1$, G and H repeat. Therefore, for X_4 , again you need G_0 and H_0 . So, this is just a pictorial representation of that equation, as simple as that.


Now, immediately it should strike you by going from an 8 point DFT to two 4 point DFT, you saved computation. But now, you need to compute these individual 4 point DFTs. Again, follow the same idea. Take the two 4 point sequences, again divide them into odd and even parts and instead of doing the 4 point DFT, do the 2 point DFT. Now, you will have 4 such sequences. Therefore, if you have a power of 2, you can do this $\log_2 N$ times.

(Refer Slide Time: 15:20)

"Divide and Conquer" Results in Savings!



- For $N = 8$, the straightforward approach requires, approximately, 64 multiplications
- The "Divide and Conquer" approach, after the first stage, requires $32 + 4 = 36$ multiplications
- Thus, this approach clearly reduces the number of additions and multiplications required




C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 17 / 31


Before we look at that, for the $N = 8$ case, the divide and conquer approach requires only 36 multiplications; 32 comes from $2 \times (N/2)^2$ and then 4 comes from the overhead due to the twiddle factor. And the overhead due to the twiddle factor involves $N/2$ multiplications. In this case, it corresponds to 4. Therefore, from 64 you have come down to 36. So, you see the divide and conquer approach paying off. So, this very clearly reducing the number of additions and multiplications.

(Refer Slide Time: 16:01)

Reusing the "Divide and Conquer" Strategy



- The same idea can be applied for calculating the $\frac{N}{2}$ point DFT of the sequences $\{g_r\}$ and $\{h_r\}$
 - Computational savings can be obtained by dividing $\{g_r\}$ and $\{h_r\}$ into their odd- and even-indexed halves
- This idea can be applied recursively $\log_2 N$ times if N is a power of 2
 - Such algorithms are called **radix 2** algorithms
- If $N = 2^\gamma$, then the final stage sequences are all of length 2
- For a 2-point sequence $\{p_0, p_1\}$, the DFT coefficients are

$$P_0 = p_0 + p_1 \quad P_1 = p_0 - p_1$$


C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 18 / 31


So, same idea can be applied to the sub sequences g and h ; further dividing them into their own odd and even indices. And, this can be applied $\log_2 N$ times when N is a power of 2 and this is called the radix 2 algorithm. So, if $N = 2^\gamma$, the final stage sequences are all of length 2, and what about the smallest length DFT that you have to compute, which is the 2 point DFT.

Because you should keep dividing by 2, you will reach a stage where the individual sequences of length 2 and length 2 DFT is the easiest DFT you can think of, because you are evaluating the transform at 0

and at π . For 0, you just need to sum up the 2 terms. For the transform at $k = N/2$ when $N = 2$, you need to just subtract. So, the final 2 length transforms are also extremely simple.

(Refer Slide Time: 17:11)

DIT Flowgraph for $N = 8$



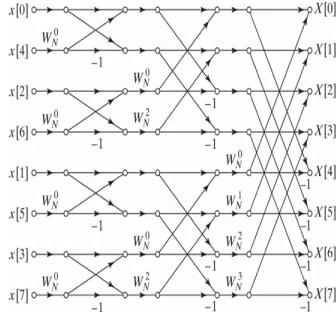



Figure 9.11 Flowgraph of Decimation in Time algorithm for $N = 8$ (Oppenheim and Schaffer, *Discrete-Time Signal Processing*, 3rd edition, Pearson Education, 2010, p. 730)


C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 19 / 31



So, this is the flow graph. So what I have done shown here is, further divided this which consisted of the even indices 0, 2, 4, 6, we have divided that into it is odd and even indices. Therefore, this becomes 0, 4, 2, 6, and 1, 3, 5, 7 now becomes 1, 5, 3, 7. So, this is the complete flow graph for the 8 point DFT. And remember, this is all sum and differences; this point is the sum, this point is the difference, because the weight factor is -1 .

(Refer Slide Time: 17:46)

Overall Operation Count



- The direct method requires N^2 multiplications
- After the first split, $N^2 \rightarrow 2\left(\frac{N}{2}\right)^2 + \frac{N}{2}$
 - $\frac{N}{2}$ is due to the *twiddle factors*
- After the second split, $\left(\frac{N}{2}\right)^2 \rightarrow 2\left(\frac{N}{4}\right)^2 + \frac{N}{4}$


Hence,

$$N^2 \rightarrow 2\left(\frac{N}{2}\right)^2 + \frac{N}{2} \rightarrow 4\left(\frac{N}{4}\right)^2 + \frac{N}{2} + \frac{N}{2}$$

first stage
second stage

- Generalizing, if there are $\log_2 N$ stages, the number of multiplications needed will be, approximately, $\frac{N}{2} \log_2 N$

C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 20 / 31




So, the overall operation count, so this requires N^2 multiplications. At the first split, we get this. Remember, we were sure of this before, the overhead was $N/2$. So, the $N/2$ is due to the twiddle factors. Now, you take each of these $N/2$ and then further divide them. So, now, for the second split,

this is the operation count. And hence, starting from N^2 , this factor was for the first stage, for the second stage, you have this. And if you keep going, you will find that if there are $\log_2 N$ stages, the number of multiplications needed will approximately be $\frac{N}{2} \log_2 N$.

From order of N^2 , we have come to $\frac{N}{2} \log_2 N$. And if you had seen the FFT algorithm before, you will find that the order of FFT requires $N \log N$ rather than $\frac{N}{2} \log_2 N$. To see where that comes from, if you do not consider this, that is, if you do not realize that $W_N^{k+(N/2)} = -W_N^k$, then the overall overhead count will be N and not $N/2$. And hence, this is how the progression looks. And hence, the overall count will be $N \log N$, this is what typically you will find in the literature.

(Refer Slide Time: 18:59)

Overall Operation Count




- If $W_N^{k+\frac{N}{2}} = -W_N^k$ is not considered, the overhead count will be N and not $\frac{N}{2}$
- In this case,

$$N^2 \rightarrow 2 \underbrace{\left(\frac{N}{2}\right)^2}_{\text{first stage}} + \underbrace{N}_{\text{second stage}} \rightarrow 4 \underbrace{\left(\frac{N}{4}\right)^2}_{\text{second stage}} + \underbrace{N+N}_{\text{second stage}}$$
- Hence the overall multiplication count will be $N \log_2 N$
- For $N = 1024$

$$N^2 = 1,048,576 \quad N \log_2 N = 10,240$$

Savings of two orders of magnitude!

C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 21 / 31



And, remember, we talked about the sequence that was thousand long and then we realized it was 10^6 in terms of operation count. Let us consider 1024, because this is the nearest power of 2 to 1000. N^2 is 1 million, $N \log_2 N$ is 10000. So, you have saved 2 orders of magnitude which is no laughing matter. Remember, suppose in matlab you do DFT and then hit return.

Suppose, it takes 100 seconds, a minute and 40; you have to wait for a minute and 40 before the answer comes. Instead, if you use FFT, it is 100 times faster. You hit return and you will get the answer in 1 second, to give you a feel for 2 orders of magnitude is really significant.

(Refer Slide Time: 20:25)

Input Sequence Order



- Recall that, for $N = 8$, the first split requires the data to be arranged as follows:
 $x_0, x_2, x_4, x_6, x_1, x_3, x_5, x_7$
- In the second and final split, the data appear in the following order:
 $x_0, x_4, x_2, x_6, x_1, x_5, x_3, x_7$
- The final order is said to be in "bit reversed" form:

Original	Binary Form	Reversed Form	Final
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7



C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 22 / 31

And, the input sequence first half was 0, 2, 4, 6 indices and then 1, 3, 5, 7. And then, further when you divided this into its odd and even indices, it became 0, 4, 2, 6 and 1, 5, 3, 7. Now, let us look at that pattern. So, this is the original index in decimal. This is the index in binary form; 0 is 000, 4 is 100, 5 is 101 and so on. Then what you do is, you take this binary form of the index and bit reverse it; 001 becomes 100 similarly, 110 becomes 011.

And, now, if you look at the decimal equivalent of the bit reversed form of the index, it is in exactly the form that you need. You needed 0, 4, 2, 6 and 1, 5, 3, 7. So, this is an important part of the FFT algorithm. Because, after all this computational advantage that you have gained, if bit reversal requires a lot of overhead, then you are back to square one; you have not gained anything. So, it is also important for you to realize that you need efficient processes for doing bit reversal.

(Refer Slide Time: 21:49)

An Algorithm For Sequence Reversal



- Consider the card sequence 7, 8, 9, 10, J, Q, K, A
- First, reverse pairwise:
 - 8, 7, 10, 9, Q, J, A, K
- Then swap the adjacent pairs:
 - 10, 9, 8, 7, A, K, Q, J
- Finally, swap the two groups of 4 (each group is half the original size):
 - A, K, Q, J, 10, 9, 8, 7 **Done!**




C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 23 / 31

And, if you have a card sequence 7, 8, 9, 10, Jack, Queen, King and Ace. If you want to reverse this, first you do pair wise reversal; 8, 7, 10, 9, *Q, J, A, K* and then you swap the adjacent pairs. So, 8, 7, 10, 9 becomes 10, 9, 8, 7 and then finally solve the two groups of 4. So, you have reversed the sequence and the reason why I am bringing this up is, remember you need to do bit reversal. So, here is a sequence and here is an algorithm that does sequence reversal which is what you need for bit reversing the index, which is important so that you present the input sequence in the bit reversed order rather than in normal order.


(Refer Slide Time: 22:37)

How To Use It For Bit Reversal



- The first step of swapping of bits pairwise can be done with bitwise AND/OR and bit shift operators
- Pick out the even and odd bits by using masks
 - $ABCDEFGH \& 01010101 = 0B0D0F0H$
 - $ABCDEFGH \& 10101010 = 0A0C0E0G$
- Left shift the first result and right shift the second result
 - $0B0D0F0H$
 - $0A0C0E0G$
- Bitwise OR the above results
 - $0B0D0F0H \oplus 0A0C0E0G = 0BADCFE0G$
- Pairwise bit swapping accomplished!

C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 24 / 31



So, how is it rather realized in practice? This is realized in practice by bitwise AND and OR operators. So, for example, if you have the sequence *ABCDEFGH* and if you mask it with this bit pattern, you get this. And you mask it with the second pattern, you pick out the other this one. And you need this, first you need to do pairwise reversal. For that, you need to pick out those two terms in the sequence.

So, what is done here is, you left shift the first result and right shift the second result. So, you do a bit mask first and then left shift and right shift and then do bitwise OR. When you go through these sequence of operations, you have done pairwise reversal; *AB* became *BA*, *CD* becomes *DC*. So, this is the first step. First step in bit reversal is pairwise reversal and the sequence of operations is this. So, pairwise swapping has been accomplished.



(Refer Slide Time: 23:52)

C Code For Bit Reversal

```
unsigned reverse_bits(unsigned input)
{
    //works on 32-bit machine
    input = (input & 0x55555555) << 1 | (input & 0xAAAAAAAA) >> 1;
    input = (input & 0x33333333) << 2 | (input & 0xCCCCCCCC) >> 2;
    input = (input & 0x0F0F0F0F) << 4 | (input & 0xF0F0F0F0) >> 4;
    input = (input & 0x00FF00FF) << 8 | (input & 0xFF00FF00) >> 8;
    input = (input & 0x0000FFFF) << 16 | (input & 0xFFFF0000) >> 16;

    return input;
}
```

- Bit reversal for the entire array can take a large overhead if performed inefficiently
- There are several efficient algorithms for sorting an array in bit-reversed order
 - *Bit reversal on uniprocessors* by Alan H. Karp, SIAM Review, Vol. 38, March 1996, pp. 1–26
 - <http://www-graphics.stanford.edu/~seander/bithack.html#BitReverseTable>



C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 25 / 31



And, this is valid only for a 32 bit machine, so these are all the sequences needed. So, you take the input, do bitwise and with this bit mask, do left shift by 1, do bitwise mask with this sequence, left shift by 2, 4, 8, and 16. Similarly, you do right shift by 1, 2, 4, 8, and 16.

So, this piece of code does bit reversal, you can go through this and study this more carefully and then you will find that this indeed does it very very efficiently. So, bit reversal can take a large overhead if it is performed inefficiently. And everything you ever wanted to know about bit reversal, but you are afraid to ask, you can look up Alan Karp's SIAM review paper and also this website.

(Refer Slide Time: 24:50)

In-Place Computation

- Notation:
 - First stage: $X_k^{(0)} = X_k$
 - Last stage: $X_k^{(\log_2 N)} = X_k$
- For the m -th stage butterfly
 - Input: $X_p^{(m-1)}, X_q^{(m-1)}$
 - Output: $X_p^{(m)}, X_q^{(m)}$
- The corresponding equations are
$$X_p^{(m)} = X_p^{(m-1)} + W_N^r X_q^{(m-1)}$$
$$X_q^{(m)} = X_p^{(m-1)} - W_N^r X_q^{(m-1)}$$




C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 26 / 31

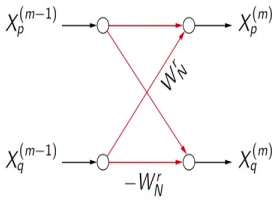
The other thing was, in those days, memory was really precious and hence they had to consider things like in place computation. That is, they had to use the available memory extremely efficiently.

So, this is the notation that we use. We use a cap $X_k^{(0)}$ to denote the sequence, and the final stage is the actual transform. And for the m^{th} stage butterfly, you really only need these two as the input and these two as the output and the corresponding equations are this.

(Refer Slide Time: 25:30)

In-Place Computation






- $X_p^{(m-1)}$ and $X_q^{(m-1)}$ are needed for computing $X_p^{(m)}$ and $X_q^{(m)}$
- They are not needed for any other pair
- Hence

$$X_p^{(m)} \leftarrow X_p^{(m-1)}$$

$$X_q^{(m)} \leftarrow X_q^{(m-1)}$$
- This is called "in-place computation"



C.S. Ramalingam (EE Dept., IIT Madras)
Intro to FFT
27 / 31



And, what this tells you is, again that those were the equations and this is the picture. So, if this were the input to the butterfly and if this were the output, for this stage, once you use these two to compute these two outputs, these two inputs will not be used in any other butterfly stage.

So, with the implication of that is, once these two inputs have been used to compute these two outputs, the computed output you can overwrite on these two memory locations. Because, these two are not going to be used by anything else later. So, after the result is computed, over write. So, this is how you save memory. And remember, these occurred in the 60s where memory was really costly. So, they were worrying about every last bit of efficiency both in terms of computational as well as storage. So, since these are not needed by any other pair, you can overwrite and this is what is called as in-place computation.

(Refer Slide Time: 26:29)

In-Place Computation

- x_0 and x_4 are not needed once that butterfly is computed
- Hence they can be overwritten with the results of the butterfly computation
 - Same is true for other pairs also





C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 28 / 31

So, if you look at this, you will realize $x[0]$ and $x[4]$ are not needed once that butterfly is computed and hence they can be overwritten. So, this is just to show you that these two terms do not occur anywhere else. And hence once this is computed, you can overwrite this result where $x[0]$ was stored and where $x[4]$ was stored. So, same is true for every other pair.

(Refer Slide Time: 26:55)

The Decimation in Frequency (DIF) Algorithm

- Another method of splitting the input sequence into half is as follows:
 $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$
- Similar savings are obtained in this case also
- The output X_k will now appear in [bit reversed order](#)
- This method is called as the [Decimation in Frequency algorithm](#)



C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 29 / 31


So, this is decimation in time; what happened there was, you need to bit reverse the input and then the transformer was in natural order. Given a sequence, if you are asked to split it into two parts, splitting into odd and even indices not the first thing that will occur to you. Whereas, what will first occur to you would be, split it into first half and second half and if you did this, exactly similar type of savings will occur. Again, you need to work out the details.

But what will happen is, the transform will now occur in bit reversed order. So, there is no escaping from bit reversal, but fortunately efficient algorithms exist. So, if the input is a natural order, you

use decimation in frequency. Transform will be in bit reverse order, you need to bit reverse that bit reversed sequence to get the natural order. Or you take the given sequence, bit reverse it, apply to the decimation in time algorithm and get the transform in natural order. So, this is called decimation in frequency.

(Refer Slide Time: 28:02)

DIF Flowgraph for $N = 8$



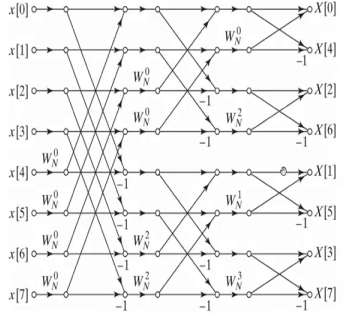



Figure 9.22 Flowgraph of Decimation in Frequency algorithm for $N = 8$ (Oppenheim and Schaffer, *Discrete-Time Signal Processing*, 3rd edition, Pearson Education, 2010, p. 740)


C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 30 / 31



And, this is the flow graph for $N = 8$. We will post this on moodle, you can look at this more carefully; the input is in natural order, output is in bit reversed order. And, remember, all of this computation savings is coming because if this were a power of 2, you are expressing it as, I mean dividing and conquering.


(Refer Slide Time: 28:30)

Prime Factor Algorithms



- When N is not a power of 2 but is a composite number, it can be expressed in terms of its **prime factors**
 - Example: $N = 6 = 3 \times 2$
- We can now split the given sequence into 3 segments of 2 samples each
 - $x_0, x_3, x_1, x_4, x_2, x_5$
- Three 2-point DFTs are computed and combined to get the final DFT
- Significant computational savings is obtained, as before
- **Efficient algorithms exist even when N is prime!**
 - http://en.wikipedia.org/wiki/Rader's_FFT_algorithm

C.S. Ramalingam (EE Dept., IIT Madras) Intro to FFT 31 / 31



Suppose, if N is not a power of 2, but is a composite number. Say for example, if N were 6, then you can still break this up into its prime factors. Remember, any composite number can be expressed as

products of primes and their powers. Therefore, you can split the sequence into $(x_0, x_3), (x_1, x_4), (x_2, x_5)$ and these three 2 point DFTs can be combined to get the final DFT and significant computational savings is obtained as before. Now, immediately it should make you think, suppose N were prime say 17, clearly it cannot be broken up into factors. So, it does not mean no efficient algorithm exists, is the natural question.

Student: zero-padding.

No, without zero-padding, zero-padding is a copout. Yeah, you can and that is what is normally done. You can zero-pad it to the nearest power of 2 and then compute the radix 2. But suppose you are interested in really finding 17 point algorithm, guess what? It does exist and if 17 is prime, $17 - 1$ is not. So, somebody really thought about this, some really brilliant mind thought about this and came up with efficient algorithms even when N is prime. You can look up Charles Rader's algorithm; Rader algorithm for prime number. So, even for prime, this exists.

So, people have squeezed every last bit out of the FFT algorithm and made it as efficient as possible and this is really the reason why signal processing is what it is today. If this algorithm did not exist, then we would not be seeing the proliferation of digital devices that we are seeing today. That require filtering and correlation all the time. So, with this we come to the end of this course.