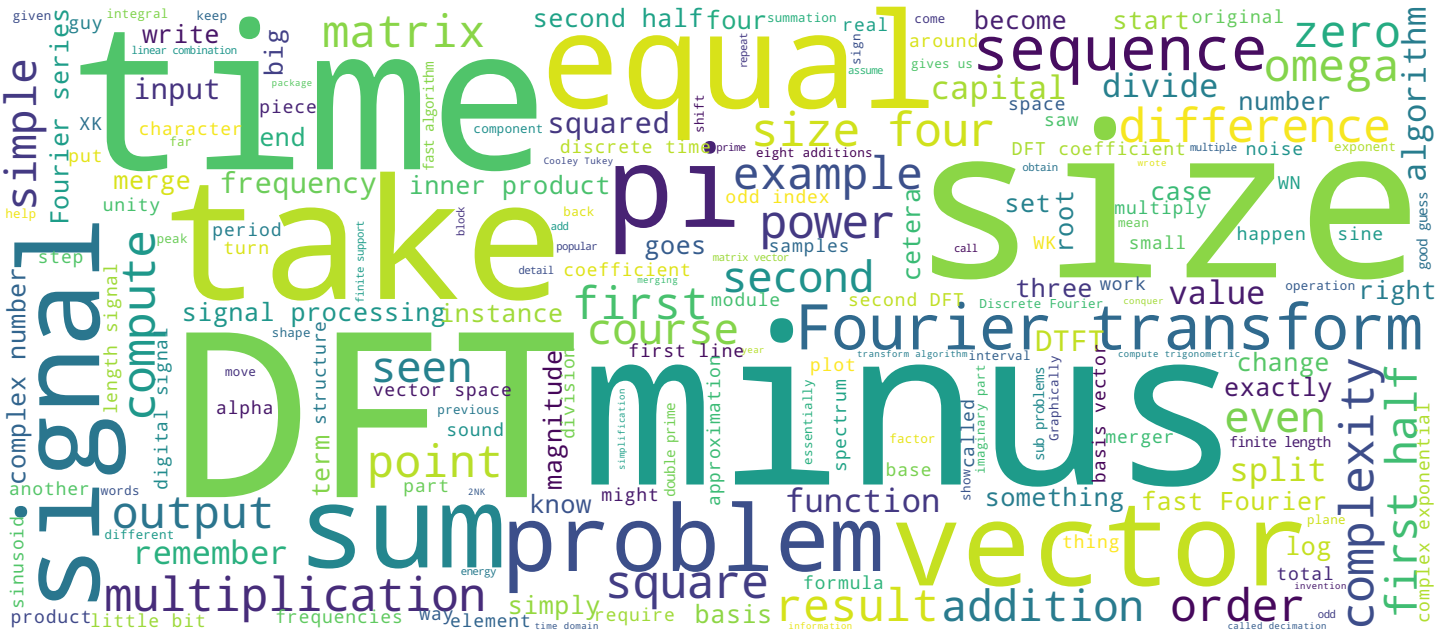
Overview

- A bit of history: From Gauss to the fastest FFT in the west
- Small DFT matrices
- The Cooley-Tukey FFT
- Decimation-in-Time FFT for length 2^N FFTs
- Conclusions: There are FFTs for any length!



Search MOOC

Video

EPFL

Fourier had the Fourier transform

The overview of the module is the following. First, we remind ourselves of the history of this topic, which goes back quite far with Gauss inventing the fast Fourier transform even before Fourier defined it. Then we will look at small examples just to get the feeling of what is going on with this matrix-vector multiplications that can be sped up by the fast Fourier transform or FFT algorithm, and we present the most famous case, which is the Cooley-Tukey FFT. We do one example in great detail, which is the so-called decimation-in-time FFT for lengths 2 to the N fast Fourier transforms, and we conclude. Fourier, of course, has the Fourier transform. If you are remembered of Fourier series, that's an invention by Joseph Fourier, and clearly if you compute Fourier series, it helps to have a fast algorithm, but at some time, it wasn't a discrete time problem. It was a continuous time problem with Fourier's series.

Notes

Summary

0m 00s

But Gauss had the FFT all along ;)

But it turned out another great mathematician, Gauss, had a similar problem computing some trajectories of planets. He had to compute trigonometric series and he was doing them by hand. He figured out a quick way to compute trigonometric series when the frequencies were actually harmonic and it turns out this was essentially the fast Fourier transform already, even before Fourier defined Fourier series.

Notes

Summary

1m 07s

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs

That leads us back to 1805 when Gauss wrote down in his personal notebook in Latin a quick way to compute trigonometric series. Fourier, as we know, invented Fourier series just a little bit later. Then a lot of people started computing Fourier series, spending a lot of time developing tricks, and in 1958, an English statistician, Good, came up with an algorithm which is still used but which is not the most popular one. The most popular one was invented in 1965 by Cooley and Tukey, and it's a rediscovery of the fast Fourier transform algorithm for N when N is a power of a prime, typically is a power of two or a power of three. After this, Shmuel Winograd combines all the methods in giving the most efficient possible fast Fourier transform algorithms.

Notes

Summary

1m 39s

The DFT matrix

- $W_N = e^{-j\frac{2\pi}{N}}$ (or simply W when N is clear from the context)
- powers of N can be taken modulo N, since $W^N = 1$.
- DFT Matrix of size N by N:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & W^3 & \dots & W^{N-1} \\ 1 & W^2 & W^4 & W^6 & \dots & W^{2(N-1)} \\ & & & \dots & & \\ 1 & W^{N-1} & W^{2(N-1)} & W^{3(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix}$$

The DFT matrix, we have seen this object before. We need an nth root of unity, that's here, E to the minus J 2 pi over N, and we often write just W because WN becomes a little bit heavy in a matrix context. The powers of N can be taken module N because of this relationship. Why is this true? Well, if you raise this to the power N, it will take out the divider by N and this is equal to 1. We can take the power's always module N. Very important point. This will lead to a lot of simplifications. A DFT matrix of size N by N has entries. Remember, the powers go from 0 to N minus 1. Okay, from zero to N minus 1, and here from 0, 1, 2, 3, N minus 1, et cetera. We see this very particular matrix, which is different entries, and we are going to look at a few examples just to become very familiar with this structure.

Notes

Summary

2m 43s

Small DFT matrices: $N = 2$

$$\mathbf{W}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$e^{-j\frac{2\pi}{2}} = -1$

When N is equal to 2, we have e to the minus J 2 pi over 2, and this is equal to minus 1. The first line of the W_2 matrix is 1 and 1. The second line is 1 minus 1, okay? No sweat.

Notes

Summary

3m 56s

Small DFT matrices: $N = 3$

$$\mathbf{W}_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & W & W^2 \\ 1 & W^2 & W^4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & W & W^2 \\ 1 & W^2 & W \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \frac{-1-j\sqrt{3}}{2} & \frac{-1+j\sqrt{3}}{2} \\ 1 & \frac{-1+j\sqrt{3}}{2} & \frac{-1-j\sqrt{3}}{2} \end{bmatrix}$$

So the second case, W3 has powers here 1 and 2, 2 and 4. We take module of all three, so you can see the 4 becomes a 1 here, so this matrix is a little bit simpler than that one, and here, we wrote out the actual values in terms of complex numbers, so this is simply e to the minus J 2 pi over 3, 4 pi over 3, 4 pi over 3, and here again, 2 pi over 3.

Notes

Summary

4m 18s

Small DFT matrices: $N = 4$

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 \\ 1 & W^2 & 1 & W^2 \\ 1 & W^3 & W^2 & W \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$e^{-j\frac{2\pi}{4}} = e^{-j\pi/2} = -j \qquad W_4 x$$

W4. Okay, so what is W4? It's e to the minus J 2 pi over 4, which is e to the minus J pi over 2, which of course is equal to minus J. And this is a matrix W4, written out as given in the general formula. Here we take module of four, so the W4 becomes a 1, W6 becomes a 2, W6 becomes a 2, and W9, module of four is just W. And now we know that W is minusJ, so we can write this out, and this matrix is very simple. First line is all ones, first column is all ones, and minus J, minus 1 j, minus 1, 1, minus 1, J, minus 1, minus J. Very simple matrix. Now, when you see this matrix, you really notice that if I multiply W4 times a vector X of lengths 4, I don't need many multiplications. For example, here, it's just a sum. Here, I have to exchange real and complex and imaginary entries. Here, I have to change the sign. Here, I just have to take the sum, but with change of signs, et cetera. W4 times X is not going to require any multiplications, only additions and subtractions and interchanges of real and imaginary parts. The reason we go through the small matrixes is to see that there is a lot of symmetry, a lot of structure that can be taken advantage of.

Notes

Summary

4m 53s

Small DFT matrices: $N = 5$

$$\mathbf{W}_5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 & W^4 \\ 1 & W^2 & W^4 & W^6 & W^8 \\ 1 & W^3 & W^6 & W^9 & W^{12} \\ 1 & W^4 & W^8 & W^{12} & W^{16} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 & W^4 \\ 1 & W^2 & W^4 & W & W^3 \\ 1 & W^3 & W & W^4 & W^2 \\ 1 & W^4 & W^3 & W^2 & W \end{bmatrix}$$

W5 is more complicated. It doesn't have these obvious simplifications that's only the module of five simplification. For example, this character here. Well, W16, module of five, the closest module of five is 15, so that be W and so on. Now this character does not have an obvious simplification. That requires quite a bit more thinking to see what the structure is, but so be it for now.

Notes

Summary

6m 45s

Small DFT matrices: $N = 6$

$$\mathbf{W}_6 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 & W^4 & W^5 \\ 1 & W^2 & W^4 & W^6 & W^8 & W^{10} \\ 1 & W^3 & W^6 & W^9 & W^{12} & W^{15} \\ 1 & W^4 & W^8 & W^{12} & W^{16} & W^{20} \\ 1 & W^5 & W^{10} & W^{15} & W^{20} & W^{25} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 & W^4 & W^5 \\ 1 & W^2 & W^4 & 1 & W^2 & W^4 \\ 1 & W^3 & 1 & W^3 & 1 & W^3 \\ 1 & W^4 & W^2 & 1 & W^4 & W^2 \\ 1 & W^5 & W^4 & W^3 & W^2 & W \end{bmatrix}$$

And just one more. W_6 starts to be a momentous matrix with a lot of entries. This is when you take the exponents module of six.

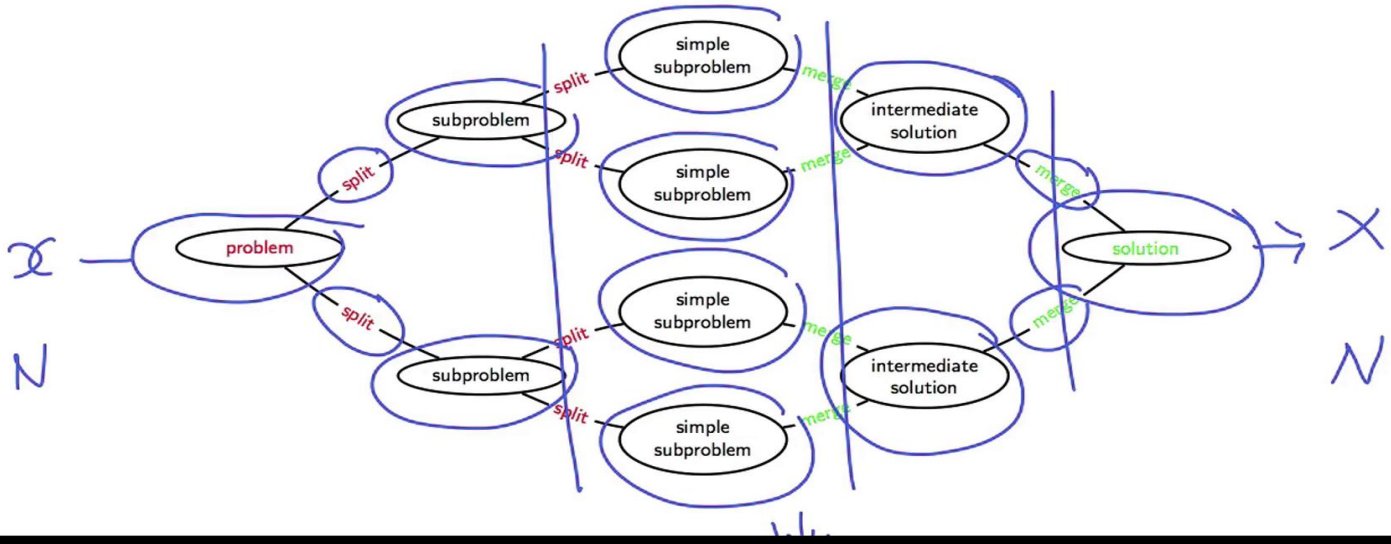
Notes

Summary

7m 18s

Divide et impera - Divide and Conquer (Julius Caesar)

Divide and conquer is a standard attack for developing fast algorithms.



Okay, so what is the idea behind the fast algorithm for the discrete Fourier transform? It is this Latin phrase, [foreign language 00:07:45], or divide and conquer. That usually is attributed to Julius Caesar. It's the idea that you take a large problem, you divide it into sub-problems that you have a better handle on. So take a problem, split it into two sub-problems, and if we could do this once, you can do it again, and assume these are simple problems by now. And so simple problems are, for example, W4. If we could take a large problem and break it down into four small problems like W4, we know this will be very simple to do. Now you have the simple problems, but you split the bigger problems into simple ones. You need emerging operation to get some intermediate solution. You can do this here also. And then you have to undo the split you did here, and that will be the merger, which gives you the solution. The key idea is how to do this split and merge, and if you know how to do it once, that was one of these steps, then you can do it a second time and then combine again, and then you have the result. In this case, we enter with a vector X, small X. It's a finite dimensional vector, let's say, of size N, and you end up with DFT coefficients, which is a finite dimensional vector of size N which contains the result of the matrix-vector multiplication.

Notes

Summary

7m 35s

Divide and Conquer for DFT - One step

Recall: computing $\mathbf{X} = \mathbf{W}_N \mathbf{x}$ has complexity $O(N^2)$.

Idea:

- Take a problem of size N where N is a power of 2.
- Cut into two problems of size $N/2$ that use complexity $\frac{N^2}{4}$ each,
- There might be some complexity to recover the full solution, say N.
- The divide-and-conquer solution has complexity $N^2/2 + N$ for one step
- For $N \geq 4$ this is better than N^2!

So is that going to help us? If we don't have a good trick to compute W times X, it's a matrix-vector product. The matrix has size N by N, X is size N of course, then the complexity is order N square. The [inaudible 00:09:43] is, you have to take N products between lines with a column, so inner products, and there are N of them. Each one takes order N computations, that's an N square, a quadratic complexity problem. We're going to study the problem when N is a power of two, and we cut the problem into two problems of size N over 2. If the complexity is quadratic, then the problems of size N over 4 have complexity. Sorry, the problems of size N over 2 have complexity N square over 4. We might have some complexity that is involved in recovering the full solution that might be a complexity order N, so our solution now needs N over 4 times 2, that's N squared over 2 plus N. That's to do with the recombination. Now if N is bigger than four, this is better than N squared, as you can immediately verify. It's not a big saving. It's a saving by essentially a factor of two here, because the dominant term, of course, is the N square. Instead of having the original N square complexity, we have an N square over two complexity.

Notes

Summary

9m 18s

Divide and Conquer for DFT - One step

Graphically

- Split DFT input into 2 pieces of size $N/2$
- Compute two DFT's of size $N/2$
- Merge the two results

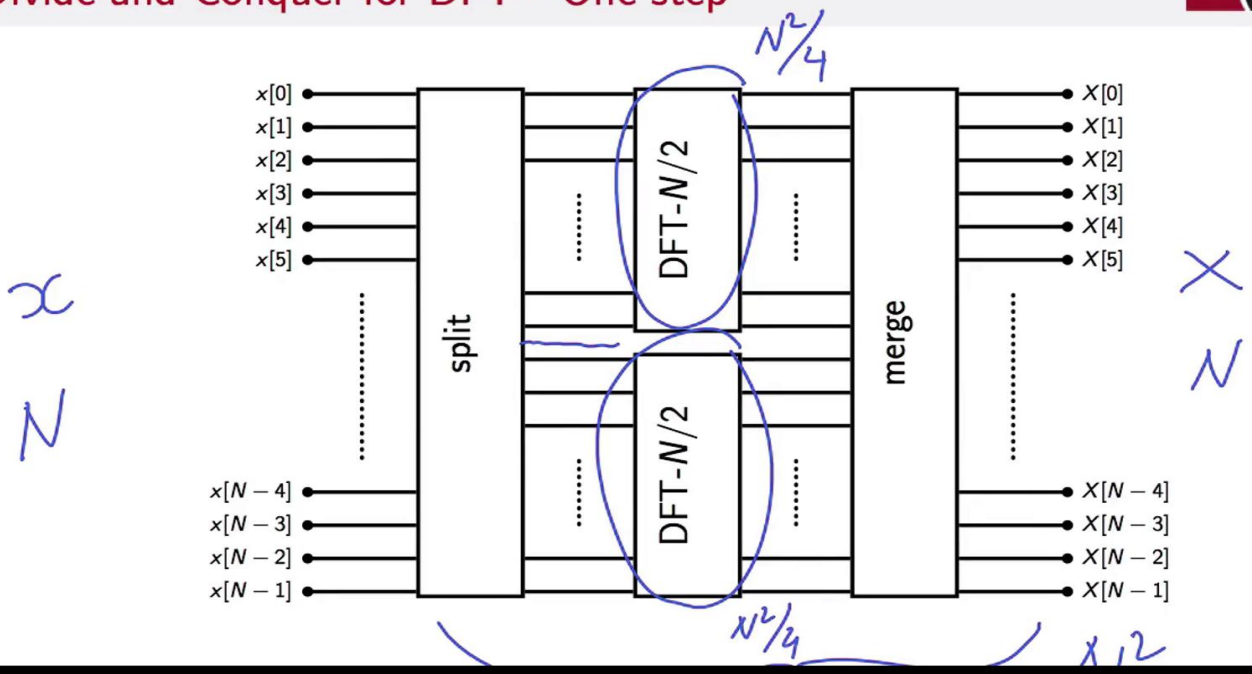
Let's show this graphically. Take a DFT, cut it into two pieces of size N over 2, compute two DFTs of size N over 2, merge the two results.

Notes

Summary

11m 10s

Divide and Conquer for DFT - One step



Good, so let's do this. We start with X. It's a size N vector. It has entry X0, X1, et cetera up to X N minus 1. Split it into two sub-problems, the upper and the lower halves, and compute DFTs of size N over 2 here. Do the merging and get the result here X, also of size N. And we just saw the original problem. This problem, if we don't do anything special about it, it has complexity N square, whereas these problems here have complexity N square over 4, this guy also, N square over 4, and so this overall thing now after split and merge has complexity N square over 2.

Notes

Summary

11m 24s

Divide and Conquer for DFT - Multiple steps

$W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Divide and conquer can be reapplied!

- If it worked once, it will work again (recall, $N = 2^K$)
- Cut the two problems of size $N/2$ into 4 problems of size $N/4$
- There might be some complexity to recover the full solution, say N at each step
- You can do this $\log_2 N - 1 = K - 1$ times, until problem of size 2 is obtained
- This requires order N complexity each time
- The divide-and-conquer solution has therefore complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better than N^2!

Now comes a magic trick. If we knew how to divide the problem once and save some computation, well, let's just do it again. Divide and conquer is reapplied, and since N is a power of 2, we can cut the two problems of size N over 2 into four problems of size N over 4, and we can keep going. Again, there might be some complexity to recover as a full solution. Assume the complexity is order N, as we will see in detail. Well, this division can be done order log 2 of N times, because you divide N into N over 2, or 4, or 8, et cetera, and if you do this log N minus 1 times, then you get sizes of problem two, which are very simple if you remember W2. This is very simple matrix which takes one addition to compute the first line versus times a vector and one subtraction for the second line.

Notes

Summary

12m 20s

Divide and Conquer for DFT - Multiple steps

Divide and conquer can be reapplied!

- If it worked once, it will work again (recall, $N = 2^K$)
- Cut the two problems of size $N/2$ into 4 problems of size $N/4$
- There might be some complexity to recover the full solution, say N at each step
- You can do this $\log_2 N - 1 = K - 1$ times, until problem of size 2 is obtained
- This requires order N complexity each time
- The divide-and-conquer solution has therefore complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better than N^2!

$\log_2 N \ll N$!

Each time we have this order and complexity to do the merging, and we get this magic result that an algorithm that uses divide and conquer at every step divides the problems into half problems over and over again will have a complexity N log in base 2 of N. And of course, if N is large, this is much better than N because log 2 N is much smaller than N. Very good then.

Notes

Summary

13m 29s

Divide and Conquer for DFT - Multiple steps

Graphically

- Split DFT input into 2, 4 and 8 pieces of sizes $N/2$, $N/4$ and $N/8$, respectively
- Compute 8 DFT's of size $N/8$
- Merge the results successively into DFT's of size $N/4$, $N/2$ and finally N

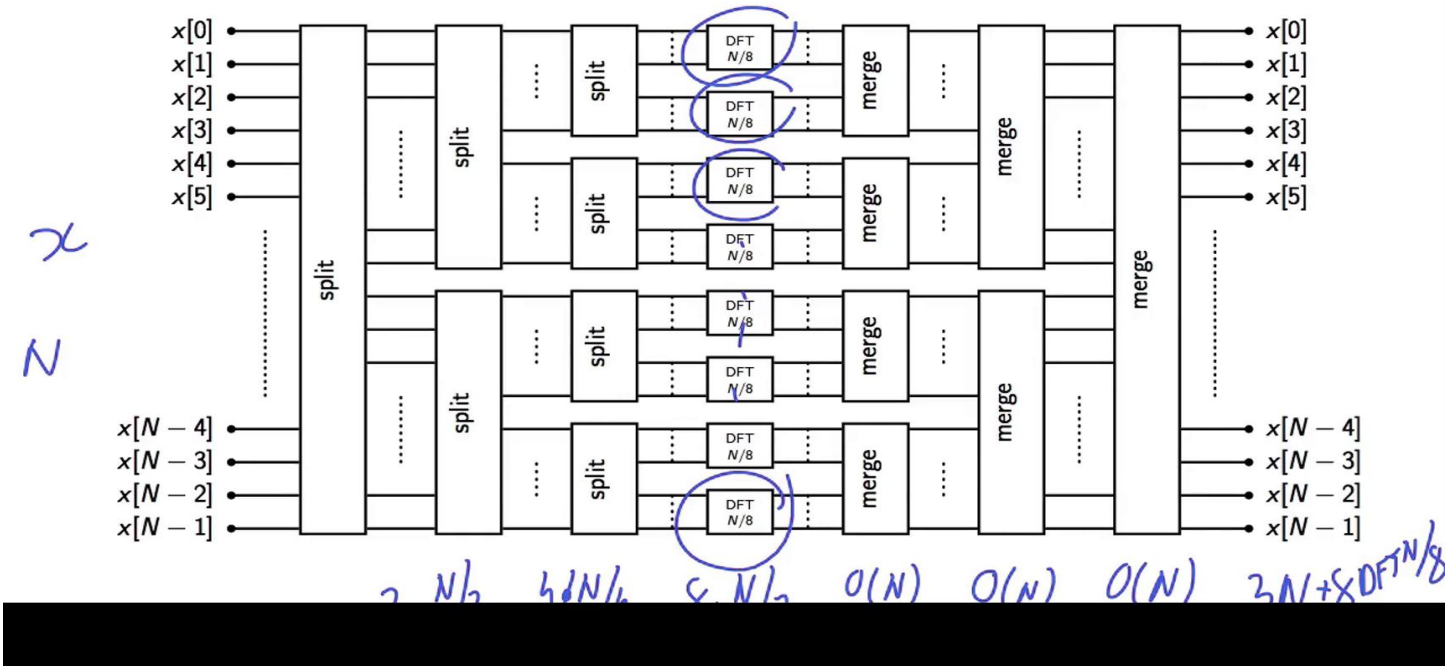
Graphically again, split the DFT in two sizes which are of size N over 2, N over 4, N over 8, for example. In this case, you end up... So you'll have two problems of size N over 2, four problems of size N over 4, eight problems of size N over 8, so this leads us to compute eight DFTs of size N over 8. Merge the results into DFTs first of size N over 4, that's the two N over 8 becomes this, and this becomes N over 2, then this becomes a result of size capital N.

Notes

Summary

14m 08s

Divide and Conquer for DFT - Multiple steps



Graphically, this is a very complex diagram, but you don't have to draw it yourself. You can just admire it here on the slide. So you take a vector X of size N, you split into two problems of size N over 2. You split into four problems of size N over four, and finally eight problems of size N over 8. So we have the original DFT of size capital N, which has been divided into smaller DFTs of size N over 8. Exactly eight of them. Then you have to merge this, and this will have complexity order N. As we announced, we'll see how this is concretely done on an example. And you can see in this particular case, we have 3 times N plus N times DFT of size N over 8. That will be the complexity of this particular algorithm here.

Notes

Summary

14m 50s

Divide and Conquer for DFT- Analysis of DIT-1/8

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W^{nk}, \qquad k = 0, 1, \ldots, N-1, \quad W = e^{-j\frac{2\pi}{N}}$$

Idea (a good guess is half of the answer!):

- break input into even and odd indexed terms (so-called "decimation in time"):

$$x[n], n = 0, 1, \ldots, N-1 \longrightarrow x[2n] \text{ and } x[2n+1], n = 0, 1, \ldots, \frac{N}{2} - 1$$

- break output into first and second half

$$X[k], k = 0, 1, \ldots, N-1 \longrightarrow X[k] \text{ and } X[k + N/2], k = 0, 1, \ldots, \frac{N}{2} - 1$$

After having seen a conceptual picture of the divide and conquer algorithm for the DFT, let's look specifically at an algorithm for the so-called division or decimation-in-time DFT. We start with the usual formula for the DFT, so it is summation over capital N terms of the time domain sequence times the root of unity raised to the N case power. We have seen this many times now.

Notes

Summary

16m 00s

Divide and Conquer for DFT- Analysis of DIT-1/8

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W^{nk}, \qquad k = 0, 1, \ldots, N-1, \quad W = e^{-j\frac{2\pi}{N}}$$

Idea (a good guess is half of the answer!):

- break input into even and odd indexed terms (so-called "decimation in time"):

$$x[n], n = 0, 1, \ldots, N-1 \longrightarrow x[2n] \text{ and } x[2n+1], n = 0, 1, \ldots, \frac{N}{2} - 1$$

- break output into first and second half

$$X[k], k = 0, 1, \ldots, N-1 \longrightarrow X[k] \text{ and } X[k + N/2], k = 0, 1, \ldots, \frac{N}{2} - 1$$

To start the algorithm, a good guess is half of the answer, so a good guess, please. Remember the good guess. It will help in the future. The good guess is that we want to divide the problem into two halves, and we could do this either by taking the first half of the sequence and the second half or because it's called decimation-in-time, we are going to take the even-indexed samples and the odd-indexed terms. That's why this is called decimation-in-time, like we have seen in certain operations in signal processing on sequences. So we take the sequence and we derive two sequences X of 2N and X of 2N plus 1. Now the input was broken into even and odd index terms, so on the output side, the DFT results, so that's capital XK we break into the first half, XK, plus the second half, XK plus capital N over 2, where K now runs only from 0 to capital N over 2 minus 1. We did two things here. We took even and odd indexes, so let's call this the even ones and the odd ones that was on the input, and on the output, we took the first half and the second half. All right.

Notes

Summary

16m 31s

Divide and Conquer for DFT- Analysis of DIT-2/8

Consider even and odd inputs separately:

$$X[k] = \sum_{n=0}^{N/2-1} x[2n]\, W^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W^{(2n+1)k}$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W^{2nk+k}$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk}$$

$$= X'_k + W^k X''_k, \quad k = 0, 1, \ldots, N-1$$

$W^{2nk} = e^{-j\frac{2\pi}{N}2nk} = e^{-j\frac{2\pi}{N/2}nk} = W_{N/2}^{nk}$

We now look at the even and odd input separately. This will be a bit of a formula menagerie here, so let's write XK as the first half of the sum, so N goes from 0 to N over 2 minus 1, and the second half, which is the odd index terms, so the even guys, the odd guys, and we do some rearrangement here. It's very simple. We expand the exponent, and the exponent, of course, is now W 2NK plus K. Then we move this part here in front because it doesn't depend on the summation, because summation is over N. So it only concerns this part. The first summation has not changed. It's the same from first line to second line to the last line. We'll just see this in a second. The only thing is that W 2NK. Well, we can write this, W 2NK is e to the minus J 2 pi over N times two N, and this is equal to e to the minus J 2 pi. over N over 2 times... Sorry, that was an nk here, times NK, and this, of course, is equal to W of root N over 2 times NK. All right. This is a simplification we have done here. This will be very important because this is simply a DFT of size N over 2 over the even index samples here with respect to root of unity of order and over two.H Here we identify a DFT, which is, DFT of size N over two.

Notes

Summary

18m 03s

Divide and Conquer for DFT- Analysis of DIT-2/8

Consider even and odd inputs separately:

$$X[k] = \sum_{n=0}^{N/2-1} x[2n]\, W^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W^{(2n+1)k}$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W^{2nk+k}$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk}$$

$$= X'_k + W^k X''_k, \quad k = 0, 1, \ldots, N-1$$

Now, the second half has exactly the same effect. Simply, we have this pre-multiplication but it's outside, but the summation inside is exactly again a DFT of size N over 2. And this DFT is simply taken over the odd indexed sample. Okay.

Notes

Summary

20m 11s

Divide and Conquer for DFT- Analysis of DIT-2/8

Consider even and odd inputs separately:

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N/2-1} x[2n]\, W^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W^{(2n+1)k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W^{2nk+k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
&= X'_k + W^k X''_k, \quad k = 0, 1, \ldots, N-1
\end{aligned}
$$

X'_k: $DFT_{N/2}$ even ; X''_k: $DFT_{N/2}$ odd !

We write this more compactly as two transforms X prime K and WK times X double prime K. And these are the results of these DFTs of size N over 2 over the even guys. And DFT of size N over two over the odd guys. Okay? This is exactly this divide of the problem to two sub-problems.

Notes

Summary

20m 29s

Divide and Conquer for DFT- Analysis of DIT-3/8

In words: We can compute X with:

- 2 half-size DFT's, which we call X' and X''
- multiplying the second DFT by W^k $k = 0 \ldots N/2 - 1$
- adding the result

In words, we can compute X, the DFT of small X with two half-size DFTs, which we have called X prime X double prime. Multiplies the second DFT by WK. K going from zero to N over two minus one and adding the results.

Notes

Summary

21m 07s

Divide and Conquer for DFT- Analysis of DIT-3/8

In words: We can compute X with:

- 2 half-size DFT's, which we call X' and X''
- multiplying the second DFT by W^k
- adding the result

$$X = X' + W^k X''$$

Okay. It will be X, first half here will be X prime plus W to the K times X double prime and this for indices over the first half.

Notes

Summary

21m 29s

Divide and Conquer for DFT- Analysis of DIT-4/8

Consider now the first and second half of the outputs separately:

$$X[k] = X'_k + W^k X''_k, \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$W^{N/2} = e^{-j\frac{2\pi}{N}\cdot\frac{N}{2}} = e^{-j\pi} = -1$$

$$X[k + N/2] = \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{n(k+N/2)} + W^{k+N/2} \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{n(k+N/2)}$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} - W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk}$$

$$= X'_k - W^k X''_k, \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

The next step is that we are going to consider separately the first half of the output and the second half. Okay. The first half we just keep the formula we have shown on the previous two slides. Okay, so XK is the sum of X prime and X double prime. There is a weight here WK, K goes from 0 to capital N over two minus one. For the second half, K still goes from 0 to capital N over two minus one. We simply add the shift here by N over 2. We just write out the formulas as we add and it simply creates these little elements here in the exponent, and we have to worry what this will do. Now, we remember that WN over two is E to the minus J 2 pi over N times N over two. This is equal to E to the minus J pi, which is equal to minus 1.

Notes

Summary

21m 49s

Divide and Conquer for DFT- Analysis of DIT-4/8

Consider now the first and second half of the outputs separately:

$$X[k] = X'_k + W^k X''_k, \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$W_{N/2}^{N/2} = 1$$

$$X[k + N/2] = \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{n(k+N/2)} + W^{k+N/2} \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{n(k+N/2)}$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} - W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk}$$

$$= X'_k - W^k X''_k, \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

Because of this, we can go to the following simplifications. In front, we have an N over 2 here that is at the power of WN over two. There we remember that this is equal to 1. We saw this at the very beginning. This thing disappears and the first half here is very simple. What do we recognize here? It's a DFT of size N over 2. Nothing too surprising here. This character here, we just saw before that gives us a minus one in front. Instead of the plus that we have above here, we have the minus sign, so W to the K. Last but not least we do the same simplification here, this falls out. It's an integral power in order to root of order N over 2. This is a second DFT.

Notes

Summary

23m 04s

Divide and Conquer for DFT- Analysis of DIT-4/8

Consider now the first and second half of the outputs separately:

$$X[k] = X'_k + W^k X''_k, \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$\begin{aligned} X[k + N/2] &= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{n(k+N/2)} + W^{k+N/2} \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{n(k+N/2)} \\ &= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} - W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\ &= X'_k - W^k X''_k, \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \end{aligned}$$

What we end up having is something very simple is that the second half looks very much like the first half except that we have a change of sign here.

Notes

Summary

24m 08s

Divide and Conquer for DFT- Analysis of DIT-5/8

In words: We can compute $X[k]$ and $X[k+N/2]$ with: $k = 0 \dots N/2 - 1$

- Divide input into even and odd indexed samples
- Compute two DFTs of size $N/2$
- Multiplication of the output of the second DFT by W^k using $N/2$ multiplications
- Combine output with sum/difference

We are going to put this in words. We can compute XK and XK plus N over 2 where K goes from 0 to capital N over 2 minus 1 to by dividing the input into even and odd index samples.

Notes

Summary

24m 18s

Divide and Conquer for DFT- Analysis of DIT-5/8

In words: We can compute $X[k]$ and $X[k + N/2]$ with:

- Divide input into even and odd indexed samples
- Compute two DFTs of size $N/2$
- Multiplication of the output of the second DFT by W^k using $N/2$ multiplications
- Combine output with sum/difference

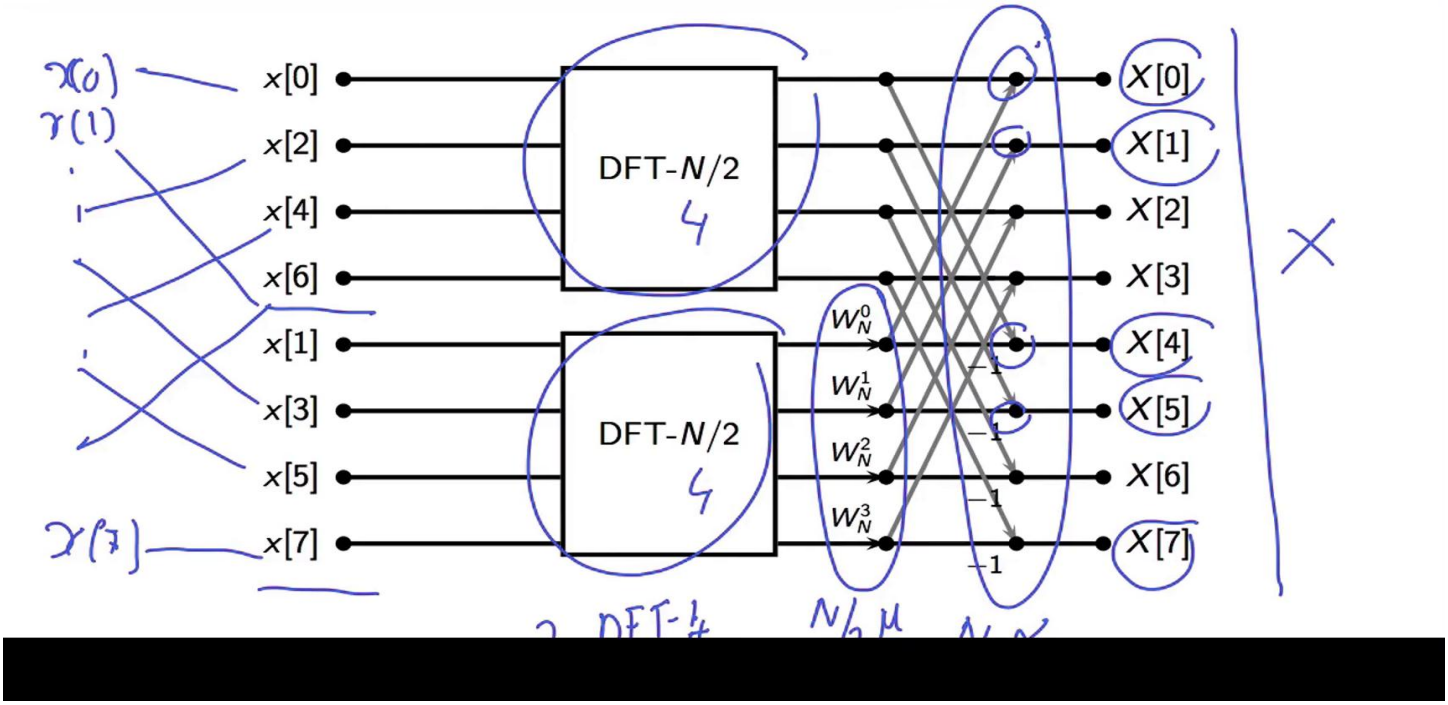
Compute two DFTs of size N over two, such the division. Multiplication of the output of the second DFT by WK that will cost us N over 2 multiplications. Combine the output with sum and difference that's what we have just seen on the last slide.

Notes

Summary

24m 38s

Divide and Conquer for DFT- Analysis of DIT-6/8



Graphically, let's draw this specifically on a length eight DFT. The input is X 0 to X 7. We have here DFTs of size four. Then the second output we multiply by WN to the case. That's WN 0, 1,2, 3. Then we take sum here and difference, that gives us X0 plus X0 plus N over 2, that's X4. That's one combinations and plus, minus gives us X1 and X5 et cetera until we get X7. All right. We have very clearly mapped out this division. We take the input. We start with X0, X1 to X7 and we divide. X2 will come here, et cetera. X3 will go there. X4 goes there. X5 goes there. X6 goes there. X7 goes there. We have done the division of the input into two house. The even and the odd. We have taken two DFTs of half size. We have weighted the output of the second DFT by these complex numbers. We have done sum and differences and we get our final result here. Capital X, okay? What was the cost? Well, we have two DFTs to compute of size four in this case. Then we have N over two multiplications, and we have N additions.

Notes

Summary

24m 58s

Divide and Conquer for DFT- Analysis of DIT-7/8

So, what is the complexity now?

- Split DFT input into 2 pieces of size N/2: free!
- Compute 2 DFT-N/2: twice $(N/2)^2$, or $N^2/2$
- Merge the two results: multiplication by $N/2$ complex numbers W^k
- Total: $N^2/2 + N/2$ which is indeed smaller than N^2 for any $N \geq 4$,
- In general, about half the complexity of the initial problem!

$N^2 \longrightarrow 2\left(N/2\right)^2 = N^2/2$

What is the complexity now? Split the DFT into two pieces of size N over 2. This is essentially free. Compute two DFTs of size N over 2. Well, if we don't know anything better, that's going to be N over 2 squared which is N squared over 4 times 2, it's N squared over 2. It seems it's before. We merge the two results. That's N over two complex numbers multiplication by WK. A total of N squared plus 2 plus N over 2, which is indeed smaller than N squared for any large enough. In general, it helps the complexity of the initial problem. Why? Because we have moved from N squared into two times N over to squared, which is N square over 2. All right? That's if we do only one step of division and merging.

Notes

Summary

26m 41s

Divide and Conquer for DFT- Analysis of DIT-8/8

So, what if we repeat the process?

- Go until DFT-2, since this is trivial (sum and difference)
- Requires $\log_2 N - 1$ steps
- Each step requires a merger of order $N/2$ multiplications and N additions
- Total: $N/2(\log_2 N - 1)$ multiplications and $N \log_2 N$ additions
- Savings of order $\log_2 N/N$

Key Result: **A DFT of size N requires order $N \log_2 N$ operations!**

Of course, as announced, we can repeat this. We'll repeat this until we get to trivial small DFTs of size two, the sum and difference. This requires log N base 2 of N minus one steps. Each step requires a merger. The merger as we had seen requires N over 2 multiplications and additions. The total now is N over 2 log N base 2 N minus 1 multiplications and log 2 of N additions. The saving is of the order of Log N over N and the key results that I want everybody to remember because it's such an important result. A DFT of size N requires order N log N operations. Why do I say it's such a key result? Because essentially signal processing has lived and died by first algorithms. The invention of the first three transform these algorithms that I showed you by Cooley and Tukey in 1965 created an explosion of interest in digital signal processing, because all of a sudden, computations which seemed very hard before were actually feasible even on very simple computers at some time. The birth of digital signal processing is really linked to the invention or the rediscovery of the fast Fourier transform algorithm.

Notes

Summary

27m 44s

Matrix factorization view of DFT, N = 4

- Separate even and odd samples
- Compute two DFT's of size 2 having output $X'[k]$ and $X''[k]$
- Compute sum and difference of $X'[k]$ and $W^k X''[k]$

even

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

DFT2

odd

This uses 8 additions and no multiplications!

Now, we can see this as matrix factorizations. Let's see this first on a DFT of size four. What do we do? We separate in even and odd samples. Then we complete two DFTs, we compute the sum and difference. We wrote it out here. I'm not expecting you that you understand this immediately by looking at four matrices of size four by four. Let me try to explain this matrix here. Does the division into even and odd? How do we see this? The first entry takes X1. Sorry, X zero. The second entry here will take out X two. The third entry takes out X one and the last one X three. Now we have separated odd and even. Then we do two DFTs of size two. Remember, the initial problem is of size four. Half-size problems are of size two. These are DFTs of size two. Then we have to multiply the outputs by the roots of unity meaning the outputs of this character here. That's these two guys. We have to recombine them by sum and difference. This was combining to this matrix here. Which does both the multiplication here by the roots of unity and the sum and difference. And this is exactly W4, this matrix we had seen before. This uses eight additions, no multiplications. Where are the additions? Here 1, 2, 3, 4, 5, 6, 7, 8. Eight additions. No multiplication. DFT of size four is very cheap. Please remember.

Notes

Summary

29m 16s

Matrix factorization view of DFT, N = 8, 1/8

Now this is going to be big...

Too big for a single slide!

$$\mathbf{W}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & W^3 & \dots & W^7 \\ 1 & W^2 & W^4 & W^6 & \dots & W^{14} \\ & & & \dots & & \\ 1 & W^7 & W^{14} & W^{21} & \dots & W^{49} \end{bmatrix} = \dots$$

If we could do it for N is equal to 4, it's tempting to do it for N is equal to 8. Now this is going to be big and is going to be too big for a single slide. We first write down what W eight is. It's the usual matrix, first line and first column of ones, and then successive roots of unity to power 1, 2, 3, 4 up to 7, then even powers, et cetera. Up to W49.

Notes

Summary

31m 05s

Matrix factorization view of DFT, N = 8, 2/8

Step 1: separate even from odd indexed samples
Call this $\mathbf{D}_8$ for decimation of size 8

$$\mathbf{D}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This requires no arithmetic operations!

Step one is to separate even and odd indexed samples. This is exactly what we have seen before. This picks out the even samples, this picks out the odd samples. This we call D8 or a decimation of size eight matrix. It requires only index changes no arithmetic operations.

Notes

Summary

31m 39s

Matrix factorization view of DFT, N = 8, 3/8

Step 2: Compute two DFTs of size N/2 on the even and on the odd indexed samples
Each submatrix is $\mathbf{W}_4$, and the matrix is block diagonal, where $\mathbf{0}_4$ stands for a matrix of 0's

$$\begin{bmatrix} \mathbf{W}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{W}_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & j & -1 & -j & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & -j & -1 & j & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & j & -1 & -j \\ 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & -j & -1 & j \end{bmatrix}$$

This requires two DFT-4, or a total of 16 additions!

Then we want two DFTs of size four. Now, we're not going to factorize these guys, which is already long hand just a couple of slides ago. But you recognize that DFT four here. This is a zero matrix and we write this compactly as a matrix which is a block matrix with two W four matrices and two zero matrices. Now, this requires two DFTs of size four. That's 16 additions. Right? So far, so good.

Notes

Summary

32m 02s

Matrix factorization view of DFT, N = 8, 4/8

Step 3: Multiply output of second DFT of size 4 by W^k
This is a diagonal matrix, with $\mathbf{I_4}$ for the identity of size 4,

$$\begin{bmatrix} \mathbf{I_4} & \mathbf{0_4} \\ \mathbf{0_4} & \mathbf{\Lambda_4} \end{bmatrix} = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & W & & \\ & & & & & & W^2 & \\ & & & & & & & W^3 \end{bmatrix}$$

where $\mathbf{\Lambda_4} = \begin{bmatrix} 1 & & & \\ & W & & \\ & & W^2 & \\ & & & W^3 \end{bmatrix}$

This requires 2 multiplications ($W^2 = -j$ is free)

Now we have to multiply the output of the second DFT of size four by WK with K going from zero to three. The first half is untouched. The second half is weighted by one, W, W square, W cube. In compact notation, we can again use block matrices. This is an identity matrix of size four. This is a diagonal matrix with the weights given by one, W, W square, W three. This character here. We have again, zero matrixes of size four. This requires two complex multiplications. Why? It's this multiplication here. This one W square is actually equal to minus J and this doesn't require complex operations, just requires interchanging real and imaginary part. So far we have 16 additions, two multiplications.

Notes

Summary

32m 39s

Matrix factorization view of DFT, N = 8, 5/8

Step 4: Recombine final output $X[k]$ and $X[k + N/2]$ by sum and difference, $\mathbf{S}_8$

$$\mathbf{S}_8 = \begin{bmatrix} \mathbf{I}_4 & \mathbf{I}_4 \\ \mathbf{I}_4 & -\mathbf{I}_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

This requires 8 additions!

16a, 2m, 8a

Now we have to recombine the final output. If you remember, it's sum and differences of XK and XK plus N over two. We do this for the first output is a sum here. Second output is a sum, third output is a sum. Forth is a sum. The fifth output is a difference again, a difference for the sixth, seventh and eighth output. This we can write compactly by recognizing that these guys are identity matrices. Here, there's simply a sign change. In block matrix form, we have, is very simple formula, the identity of size four, identity of size four, identity of size four minus identity of size four. This requires eight additions. If my calculations were correct, we had 16 additions with our two DFTs of size four, then we had two complex multiplications, and now we have again eight additions. These are additions and multiplications.

Notes

Summary

33m 46s

Matrix factorization view of DFT, N = 8, 6/8

In total:
Product of 4 matrices

$$W_8 = \begin{bmatrix} I_4 & I_4 \\ I_4 & -I_4 \end{bmatrix} \cdot \begin{bmatrix} I_4 & 0_4 \\ 0_4 & \Lambda_4 \end{bmatrix} \cdot \begin{bmatrix} W_4 & 0_4 \\ 0_4 & W_4 \end{bmatrix} \cdot D_8$$

2-DFT-4

This requires 24 additions and 2 multiplications!

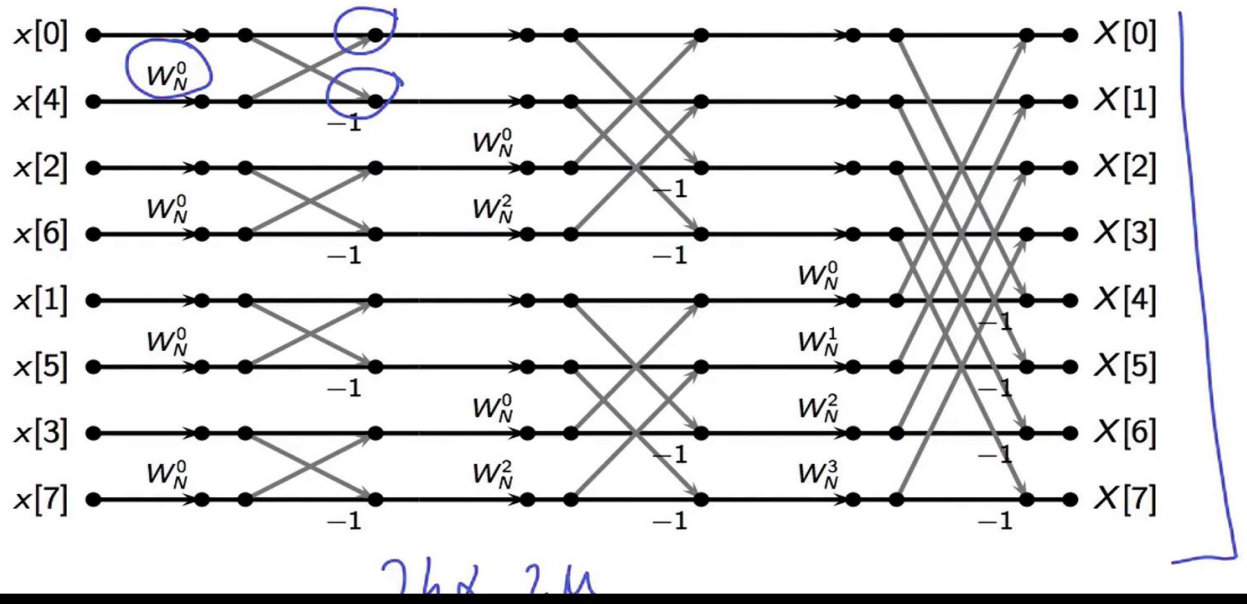
We have a total of 24 additions and two multiplications. Just as a summary, we wrote down this factorization, D eight is a separation into even and odd index terms. These are the two DFTs of size four. This is weighting by one W W square W cube. This is the sum and difference to recover the output, which is W eight. Please remember this nice result here that is small DFT has very low complexity.

Notes

Summary

35m 03s

Flowgraph view of DFT, N = 8, 7/8



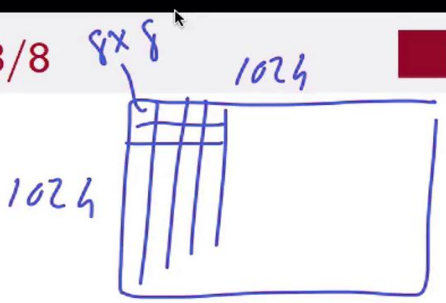
We can write this out in a flow graph for you. I'm not going to go through this detail, but you can see what is happening. We separate the inputs now, not just even an odds because we do this twice or we end up having X0 and X4 as neighbors X2, X6, X1, X5, X3, X7. Then we do the weightings, the sum and difference and we keep going. Here is a famous last sum and difference, which indeed computes the result here. This flow diagram is everything we have seen before put into a flow diagram where everything moves from left to right. It gets weighted when we have a complex number next to it and it gets summed or differenced when we have a merger of two arrows. You can count here the number of multiplications. You will get the same number of 24 adds and two multiplications. These operations are always on complex numbers.

Notes

Summary

35m 43s

Matrix factorization view of DFT, N = 8, 8/8



Is this a big deal?

- In image processing (e.g. digital photography) one takes block of 8 by 8 pixels
- One computes a transform (called DCT) similar to a DFT
- It has a fast algorithm inspired by what we just saw

You might say this was a lot of effort on a very small size DFT. Let's see where this has a real impact in practice. In image processing, if you have a digital camera, you take large images, thousand by a thousand pixels or something like this. Out of these large images, small blocks are derived. You take a large image, let's say 1024 by 1024 pixels to make it simple. Then you divide it into small blocks here and the small blocks are of size eight by eight.

Notes

Summary

36m 52s

Matrix factorization view of DFT, N = 8, 8/8

Discrete cosine tra.

Is this a big deal?

- In image processing (e.g. digital photography) one takes block of 8 by 8 pixels
- One computes a transform (called DCT) similar to a DFT
- It has a fast algorithm inspired by what we just saw

On that eight-by-eight block, we take a transform called the Discrete Cosine Transform, very important transform. Discrete Cosine Transform. And DCT is essentially a real version of a DFT. I don't want to get into the trigonometric mess of defining the DCT in detail. We could do this in the supplementary lecture. For now, let's just assume it's a real version of a DFT and it has a fast algorithm that is very close with DFT algorithms.

Notes

Summary

37m 27s

Matrix factorization view of DFT, N = 8, 8/8

Is this a big deal?

- In image processing (e.g. digital photography) one takes block of 8 by 8 pixels
- One computes a transform (called DCT) similar to a DFT
- It has a fast algorithm inspired by what we just saw

A little bit more complicated, little details, phase shifts and so on. But to a first view, it is like a real version of a Discrete Fourier Transform.

Notes

Summary

37m 59s

Some numbers

- Direct: $64^2 = 4096$ multiplications required (dominant cost, fixed point multiplications)
- The transform can be computed in rows and columns separately, or 16 DFT's
- The algorithm we saw has 2 multiplications, or a total of 32 multiplications
- Saving of 2 orders of magnitude!

If we takes a direct transform, well, 8 by 8 is 64 so there is a matrix of size 64 by 64 that multiplies a vector where we have the pixels from the 8 by 8 block. Sixty four square is 4000 and something 4096, and for every block you have to take these transform. Each multiplication has to be computed with fixed point multiplication. It's expensive, takes time or hardware. Now there is the first trick is that these transform is a so called separable transform. You can take your 8 by 8 block. Let me try to make an approximation of an 8 by 8 block. Sorry. It takes me longer than it should, but that's an 8 by 8 block. It turns out DCT transform you can take transform over each line. That's eight transforms and transforms or eight columns, that's another eight transformed. That's 16 DFTs, let's say. Now the algorithm we saw has two multiplications. We have a total of 32 multiplications, 32 rather than 4000. That's two orders of more than two orders of magnitude, right? If you have a fixed point processor in your camera instead of working for 100 seconds, it will work for half a second or so. That's a big deal if you have taken pictures with your digital camera.

Notes

Summary

38m 12s

Some numbers

- Direct: $64^2 = 4096$ multiplications required (dominant cost, fixed point multiplications)
- The transform can be computed in rows and columns separately, or 16 DFT's
- The algorithm we saw has 2 multiplications, or a total of 32 multiplications
- Saving of 2 orders of magnitude!

JPEG

You know that you would like to take the next picture and you don't want to wait for 100 seconds. This is a real big deal and this is at the heart of an image compression algorithm that is called JPEG and that we will discuss later in this digital signal processing class.

Notes

Summary

39m 54s

Conclusions

Don't worry, be happy!

- The Cooley-Tukey is the most popular algorithm, mostly for $N = 2^N$
- Note that there is always a good FFT algorithm around the corner
- Do not zero-pad to lengthen a vector to have a size equal to a power of 2
- There are good packages out there (e.g. Fastest Fourier Transform in the West, SPIRAL)
- It does make a BIG difference!

FFTW

It's time to conclude. The short answer to this module is, don't worry, be happy. There is a Cooley-Tukey algorithm very popular that for N which is the power of 2 gives a complexity that is of order N log in based to N. Really big deal. Maybe one of the most important algorithms ever invented. Now, this is the one we explained here. I want to be clear that for every possible length there is an FFT algorithm. It's not going to be a Cooley-Tukey. It's not going to be the simple, divide and concure in pieces which are half pieces of the initial problem, et cetera. But there are algorithms for every possible lengths, even lengths which are prime, even lengths which are a product of different primes, et cetera. When you compute an FFT, you should never zero path the vector so as to get the length that is a power of two. This is something that people do, and it's a mistake because you add zeros when maybe there shouldn't be any zeros. You should just take the initial length and look at the package that is out there, for example, the Fastest Fourier Transform in the West FFTW, which has a whole series of routines and works very well.

Notes

Summary

40m 13s

Conclusions

Don't worry, be happy!

- The Cooley-Tukey is the most popular algorithm, mostly for $N = 2^N$
- Note that there is always a good FFT algorithm around the corner
- Do not zero-pad to lengthen a vector to have a size equal to a power of 2
- There are good packages out there (e.g. Fastest Fourier Transform in the West, SPIRAL)
- It does make a BIG difference!

There are also other packages. One is called SPIRAL that also computes very efficiently all sorts of different Fast Fourier transforms. If you use these packages, it will make a big difference. It will make differences by orders of magnitude. With this, I think we have finished this entire module on Fourier Transform, the Fourier series and the DFTs, the Discrete Fourier Transform. That we concluded with this topic on computational complexity and fast algorithms that has made a big difference in the implementation of signal processing algorithms.

Notes

Summary

41m 33s