Algovize

Luděk Kučera

January 12, 2008

1 Fast Fourier transform

Scene: *Fast Fourier Transform - FFT*

If you are not familiar with aplications of the discrete Fourier transform and you skipped the previous chapter, where the principal application is explained, I recommend to return back to read it. Even though it is not necessary for understanding the fast Fourier transform, you will learn why to become interested in fast computing of such a strange formula as the DFT is, and first of all, why it is important to perform the computation as quickly as possible.

In this chapter, we will give an algorithm that, given a vector $(a_0, \ldots, a_{n-1})$, computes efficiently a vector $(A_0, \ldots, A_{n-1})$ such that

$$A_k = \sum_{j=0}^{n-1} \omega^{k \cdot j} a_j \quad \text{for } k = 0, \ldots, n-1,$$

where ω is a number such that $\omega^n = 1$ anad numbers $1 = \omega^0, \omega, \omega^2, \ldots, \omega^{n-1}$ are mutually disjoint. An example of such a number is a complex number

$$\omega = e^{2\pi i/n},$$

wherein $i = \sqrt{-1}$.

The FFT algorithm requires that the dimension n of vectors is a power of 2, which is not too restrictive from the practical point of view; a power 2 width is quite common in computer science.

In this scene the DFT formula will be rewritten in such a way that it gives the same result, but it suggests faster way of computation. The method consits in a repeated application of the next sequence of steps (to perform the steps, keep clicking the button [**Next**], while the button [**Back**] can be used to go one step back):

Step: *Original formula*

The initial step shows the definition of the DFT in a matrix form. The equivalent mathematical formula in the botton of the screen can be hidden by the checkbox [**Formula**]. The dimension n can also be changed to another power of 2 smaller than 128 using the choice in the control bar; the default is $n = 8$, which is sufficiently non-trivial, but still allows using larger fonts.

Step: *Formal modificaion of the formula*

While the factors ω^0 (equal to 1) and the exponent of ω^1 were not explicitly shown in the previous step, they are shown now.

Step: *Column partitioning*

The columns are partitioned to odd and even, as suggested by their color.

Step: *Column separation*

Columns are moved to separate (originally) odd columns from (originally) even columns.

Step: *The power of ω modification*

All terms in the right half of the k-th row involve the factor ω^k, that is first shown in red and then separated.

Step: *The key step of the FFT*

The following modification of the formula is the key step, which is the only one making use of the equality $\omega^n = 1$ and makes the FFT as fast as it is.

Look at any term in the upper half of the scheme and at the term that appears exactly $n/2$ rows below it. The difference of the exponents to ω in the two term is always divisible by n. This is because the power of ω has the form $\omega^{s\cdot k}$, where k is the index of the row, where the term is located, and is the term was originally in the S-th column, then either $s = S$ if S is even or $s = S - 1$ if S is odd. Hence, the number s is always even and the difference of line indices of both terms is $n/2$. The difference of exponents of the two terms is therefore an even multiple of $n/2$, i.e., a multiple of n and it is sufficient to realize that $\omega^n = 1$, which implies

$$\omega^{s\cdot k} = \omega^{s\cdot k} \cdot 1^{s/2} = \omega^{s\cdot k} \cdot [\omega^n]^{s/2} = \omega^{s\cdot k} \cdot \omega^{sn/2} = \omega^{s(k+n/2)},$$

where the leftmost term is the power of ω of the upper term and the rightmost term is the power of ω of the lower term.

The exponent change in the bottom part of the matrix is first sugested and then performed.

Step: *Circuit-like description*

Our goal is not only to show how to write a program computing the FFT algorithm using an ordinary computer, but also to show a layout of a simple and elegant circuit that is used to compute the transformation. While the formula was written in a purely mathematical form in previous steps, and each row was in the form $(\ldots) + \omega^k(\ldots)$, where the left hand parenthesis enclose the sum of terms from the even columns and the right hand term refers to odd columns, now we use a semi-engineering formalism: the two terms are computed as mathematical sums and the results are moved along green wires to inputs of two gates - one multiplies the second value by an appropriate power of ω and the other adds the two obtained values together.

Step: *Comparison of the NE and the SE blocks*

The previous steps have modified the matrix in such a way that it decomposes in a natural way into four $n/2 \times n/2$ matrices. Using a geographic terminology, after the previous steps the north-east and the south-east blocks are identical, and the same is true for the north-west and the south-west blocks. The present step emphasizes the blocks of the first pair (the blocks on the right) - you can check the equality again.

Step: *Overlay of the NE and the SE blocks*

In order to clearly visualize equality of the blocks selected in the previous step, the NE one is moved to be put over the SE one. Let us note that the program still draws *both* blocks; the fact that only one is visible just stresses their equality.

Seen in an engineering way, we have just one subcircuit to compute the corresponding values and each component of the resulting vector is passed to inputs of two different gates.

Step: *Comparison of the NW and the SW blocks*

In this step, the NW and the SW blocks are emphasized; check that they are identical.

Step: *Overlay of the NW and the SW blocks*

The SW and the NW blocks are moved to coincide. The resulting scheme can be understood in two ways:
a programmer sees it as a logical scheme of computation, where any value computed according the expressions on the right is stored and then used twice to get the component of the final result;

an engineer understands the scheme as a suggestion for a layout of a circuit, where green lines represent physical (groups of) wires that pass each value obtained in the right part of the circuit to two gate inputs on the left.

Step: *Repeating recursion*

An important observation: the two blocks that appear now on the right represent discrete Fourier transformation of vectors of dimension $n/2$. The upper block, arising from the ofiginal NW and SW blocks, transforms a vector $(a_0, a_2, \ldots, a_{n-2})$ and the lower block, an overlay of the original NE and SE blocks, transforms $(a_1, a_3, \ldots, a_{n-1})$.

It is just sufficient to realize that if $\omega^n = 1$, then $\vartheta = \omega^2$ is the $n/2$-th root of unity, i.e. $\vartheta^{n/2} = 1$. Moreover, the first $n/2$ powers of ϑ $(\vartheta^0, \vartheta^1, \vartheta^2, \ldots, \vartheta^{n/2-1})$ are even members of the sequence of the first n powers of ω $(\omega^0, \omega^1, \omega^2, \ldots, \omega^{n-1})$, and hence they are distinct. This is why it is possible to replace ω^2 by ϑ in the blocks.

To avoid confusion with numbers ω, ϑ, etc., we denote the m-th root of unity explicitly as ω_m in the screen sceme. Since the original ω was in fact ω_n, the value of $\vartheta = \omega^2$ is denoted as $\omega_{n/2}$.

Further steps repeat the above given sequence of steps for both blocks simultaneously to eventually reduce the original $n \times n$ block to n one-element block.

Scene: *High dimension*

The higher is the dimension, the higher is advantage of the FFT against the naive computation of the DFT. In the present scene, the dimension is increased to $n = 64$ (the resolution of common displays does not make it possible to produce a nice drawing for larger powers of 2).

Even in this case the terms of the DFT matrix degenerate to simple black dots; each dot represents one multiplication by a power of ω and (except for the leftmost column) one addition. In this way it is easy to imagine how large is the complexity of computing DFT in a naive way.

In this scene clicking the button [**Next**] performs one recursion cycle. One click reduces the scheme to computing and combining results of the discrete Fourier transformation of two vectors of dimension 32, the next step uses 4 transformations of vectors od dimension 16, and we have to repeat the recursion $6 = \log_2 64$ times to get the final FFT circuit.

In all stages of development of the circuit, a black dot represents one multiplication and (in most cases) one addition, while green lines are just simple wires for an engineer and logical links for a programmer. In my personal view, the graphical representation in this scene gives much better understanding of advantages of the FFT algorithm than a statement that, for a general n, about $2n \log n$ arithmetic operations are performed instead of about n^2 operations of the naive algorithm.

The final circuit uses three types of green wires: horizontal, those in the SE-NW direction and those in the NE-SW directions. Each of the groups consists of parallel wires that do not intersect, and hence it is possible to use three layers when manufacturing the circuit, which is well acceptable from the point of view of the recent circuit technology.

Green wires occupy a larger area of the figure than logical gates. This not a coincidence: it is possible to prove that, under assumption of a given fixed minimal distance of wires, *any* circuit computing the discrete Fourier transform requires wire area proportional to n^2 if the computational speed comparable to the FFT circuit in the figure is kept. In other words, small wire area circuits must be slow. The result has nothing to do with the computational complexity of the FFT algorithm or any other algorithm computing DFT, the proof is based on analysis of input-output data flow of the DFT.

Scene: *Parallel computation*[1]

The scene shows once again a final FFT circuit. The circuit consists of $\log_2 n$ columns, each of them consisting of n pairs of simple addition and multiplication gates. The data flows from the right to the left, and therefore each gate performs just one atomic computation on each input vector.

[1] Animation is not yet finished

This implies that if the circuit is used as a logical scheme for a single-processor computer, $O(n \log n)$ operations are performed during the computation.

FFT is often used to process digital signal (acustic, visual, etc.) in real time, and the dimension of vectors can be large. Therefore it is convenient (and sometimes absolutely necessary) to compute the transform in parallel. All computation in one column can be performed in two steps, one parallel multiplication and one parallel addition. Thus, $2 \log n$ parallel steps are sufficient.

It is also clear that the scheme represents a circuit with extremely simple data flow; in order to transform one vector, $\log_2 n$ stages are necessary, each of the stages consists of n pairs of arithmetic operations (1 multiplication, 1 addition) that are performed simultaneously.

Clicking the button [**Execute**] triggers a simple animation illustrating the data flow through the circuit; vectors being processed are distinguished by different colors.

Scene: *Pipelined computation*[2]

The scene repeats the previous one. Let us note that different stages of parallel computation described in the previous scene are performed in distinct columns. When the first stage is finished, the rightmost column of gates becomes idle, and therefore it can immediately be used to process another vector. In this way vectors can be pipelined through the circuit in such a way that the time span between two inputs is constant (equal to the processing time of one gate pair). A pipelined circuit is an extremely fast and powerful device to transform massive vector streams in real time.

Clicking the button [**Execute**] triggers a simple animation illustrating the data flow through the circuit; vectors being processed are distinguished by different colors.

[2] Animation is not yet finished