The Fast Fourier Transform and the Perfect Shuffle

This is part of my effort to discover how complicated algorithms should be presented nicely. We have tried our hands on sorting in situ; in my own eyes, the choice of the Fast Fourier Transform and the Perfect Shuffle as next proving grounds is ambitious: the corresponding texts I saw about nine years ago were all very ugly, my own one not excepted.

There are two sides to this coin. There is firstly the Fast Fourier Transform itself, which accomplishes with of the order $N \cdot \log N$ operations what would otherwise require a number of operations of order N^2. It embodies an ingenious invention, and its adequate presentation is no trivial task. There is secondly the question of how to embed the algorithm in a finite piece of equipment: in a sequential machine with a random access memory we would like to minimize storage demands, in a machine with concurrent computational activity we would like to minimize time and connectivity. My first concern was therefore to present the idea of the fast Fourier transform in a way that is equally adequate for both ways of embedding it.

The Fast Fourier Transform

Given the sequence $A(i: 0 \le i < N)$, we are requested to compute the sequence $f(i: 0 \le i < N)$ defined by

(0) $f(j) = (\underline{S}k : 0 \le k < N : A(k) \cdot x^{j \cdot k})$

where $x = e^{2 \cdot \pi \cdot i / N}$ —where "i" stands for the imaginary unit— and "$\underline{S}k$" stands for summation over k, followed by the range for k and the term to be summed. Straightforward application of (0) would lead to an N^2 algorithm. The Fast Fourier Transform derives its speed from the fact that for even N $x^{N/2} = -1$; in order to reap that benefit all through the computation, we shall restrict ourselves to the case that N is a power of 2 .

The Fast Fourier Transform is most easily described as massaging in $^2\log N$ steps a two-dimensional array of N values. Initially the array consists of the $A(i)$ in a single column; in each step the number of rows is halved and the number of columns is doubled; after $^2\log N$ steps we end up with a single row containing the N f-values.

More precisely, with
m = the number of rows of array c
n = the number of columns of array c ,
we initially have

$$m = N,\ n = 1, \text{ and } \quad c(i,0) = A(i) \text{ for } 0 \le i < m \quad ;$$

we end up with

$m = 1$, $n = N$, and $c(0,j) = f(\tilde{j})$ for $0 \leq j < n$

where the $({}^2\log n)$-bit binary representations of j and $\tilde{j}$ are each other's reverse (i.e. the digits of the one occur in the reverse order in the other). We leave to the reader the verification that initial state and final state satisfy

(1) $(\underline{S}\, i: 0 \leq i < m: c(i,j) \cdot x^{n \cdot i \cdot k}) = f(n \cdot k + \tilde{j})$ for $0 \leq k < m$ and $0 \leq j < n$

In the following description of a single step we shall use primes to denote values after that step. Since m (≥ 2) has to be halved and n $(\leq N/2)$ has to be doubled, we have $m' = m/2$ and $n' = 2 \cdot n$. We shall describe how by proper definition of c' a step maintains (1). Even k and odd k will be dealt with separately.

$k = 2 \cdot h$. We have $0 \leq h < m'$. We first substitute in (1)'s left-hand side, remembering $x^{n' \cdot m'} = 1$.

$$(\underline{S}\, i: 0 \leq i < m: c(i,j) \cdot x^{n' \cdot i \cdot h})$$

$$= (\underline{S}\, i: 0 \leq i < m': c(i,j) \cdot x^{n' \cdot i \cdot h} + c(i+m',j) \cdot x^{n' \cdot (i+m') \cdot h})$$

$$= (\underline{S}\, i: 0 \leq i < m': (c(i,j) + c(i+m',j)) \cdot x^{n' \cdot i \cdot h})$$

We define the even-indexed columns of c' by

(2) $c'(i,j') = c(i,j) + c(i+m',j)$ with $j' = 2 \cdot j$, $0 \leq i < m'$.

A consequence of $j' = 2\cdot j$ is $\tilde{j} = \tilde{j}'$ with the understanding that the primed reverse refers to binary representations of length ${}^2\log n'$, i.e. 1 bit longer than the unprimed reverse. As a result $n\cdot k + \tilde{j} = n'\cdot h + \tilde{j}'$ and we deduce for even j'

(3) $(\underline{S}i : 0 \leq i < m' : c'(i, j')\cdot x^{n'\cdot i\cdot h}) = f(n'\cdot h + \tilde{j}')$ for $0 \leq h < m'$ and $0 \leq j' < n'$.

(End of $\underline{k = 2\cdot h}$.)

$\underline{k = 2\cdot h + 1}$. We have $0 \leq h < m'$. Substituting in (1)'s left-hand side and remembering $x^{m'\cdot n} = -1$ we deduce

$$(\underline{S}i : 0 \leq i < m : c(i, j)\cdot x^{n\cdot i\cdot (2\cdot h+1)})$$

$$= (\underline{S}i : 0 \leq i < m' : c(i, j)\cdot x^{n\cdot i\cdot (2\cdot h+1)} + c(i+m', j)\cdot x^{n\cdot (i+m')\cdot (2\cdot h+1)})$$

$$= (\underline{S}i : 0 \leq i < m' : c(i, j)\cdot x^{n\cdot i\cdot (2\cdot h+1)} - c(i+m', j)\cdot x^{n\cdot i\cdot (2\cdot h+1)})$$

$$= (\underline{S}i : 0 \leq i < m' : x^{n\cdot i}\cdot (c(i, j) - c(i+m', j))\cdot x^{n'\cdot i\cdot h}) .$$

We define the odd-indexed columns of c' by

(4) $c'(i, j') = x^{n\cdot i}\cdot (c(i, j) - c(i+m', j))$ with $j' = 2\cdot j + 1$, $0 \leq i < m'$.

With the same understanding as above, a consequence of $j' = 2\cdot j + 1$ is $n + \tilde{j} = \tilde{j}'$, hence $n\cdot k + \tilde{j} = n'\cdot h + n + \tilde{j} = n'\cdot h + \tilde{j}'$. Thus we have deduced (3) for odd j'.

(End of $\underline{k = 2\cdot h + 1}$.)

Relation (3) is the same one for the primed quantities as (1) is for the unprimed ones. The step therefore maintains the invariant (1).

The embedding for a sequential computation

In a sequential computation we wish to use the same N storage locations $d(i: 0 \le i < N)$ to represent the successive values of c, but in such a way that the successive assignments of type (2), (4) don't destroy old c-values that are still needed.

For this purpose the array c is stored column-wise, i.e.

(5) $$c(i,j) = d(m \cdot j + i)$$

Consider the right-hand sides of (2) and (4) for a pair i,j. According to (5) they involve the old values of $d(m \cdot j + i)$ and of $d(m \cdot j + i + m')$. The left-hand side of (2), i.e. $c'(i,j')$ with $j' = 2 \cdot j$, involves according to (5) $d(m' \cdot j' + i) = d(m \cdot j + i)$. The left-hand side of (4), i.e. $c'(i,j')$ with $j' = 2 \cdot j + 1$, involves $d(m' \cdot j' + i) = d(m' \cdot (2 \cdot j + 1) + i) = d(m \cdot j + i + m')$. The left-hand sides involve the same pair of locations as the corresponding right-hand sides; hence the transition from c to c' can be effectuated by a sequence of pair-wise assignments. Note that the pairing depends on m' and hence differs from step to step.

The embedding for a concurrent computation

In a concurrent computation we wish to use the same N nodes $d(i: 0 \leq i < N)$ of a network to represent the successive values of c, but in such a way that the network is connected as sparsely as possible.

To this purpose the array c is stored row-wise, i.e.

(6) $$c(i,j) = d(n \cdot i + j) \quad .$$

According to (6), the values involved in the right-hand sides of (2) and (4) are $d(n \cdot i + j)$ and $d(n \cdot i + j + N/2)$. The left-hand side of (2), $c'(i, 2 \cdot j)$, is $d(n' \cdot i + 2 \cdot j) = d(2 \cdot (n \cdot i + j))$; the left-hand side of (4) is $d(2 \cdot (n \cdot i + j) + 1)$. In short, the pair of sources $d(k)$ and $d(k + N/2)$ has to be connected to the pair of targets $d(2 \cdot k)$ and $d(2 \cdot k + 1)$. This connection pattern, which is the same for all steps, is known as the Perfect Shuffle.

Note that in the initial state and in the final state c is a single column or row, in which cases there is no difference between row- or column-wise storing.

After a discussion at the Tuesday Afternoon Club I wrote the above with A.J.M. van Gasteren looking over my shoulder.

Plataanstraat 5
5671 AL NUENEN
The Netherlands

25 February 1982
prof. dr. Edsger W. Dijkstra
Burroughs Research Fellow