Appendix C

Following on from the analysis within Appendices A & B, let's examine FFTs where the transform size **N** is of the form 4^m for some non-negative m.

For m = 1 we have a 4-point DFT, $\mathbf{w_4} = \exp[-i * 2 * \pi/4] = \exp[-i * \pi/2] = \mathbf{-i}$:

$$
\begin{aligned}
\mathbf{F}[0] &= \mathbf{f}[0] + \mathbf{f}[1] + \mathbf{f}[2] + \mathbf{f}[3] \\
\mathbf{F}[1] &= \mathbf{f}[0] - \mathbf{i}*\mathbf{f}[1] - \mathbf{f}[2] + \mathbf{i}*\mathbf{f}[3] \\
\mathbf{F}[2] &= \mathbf{f}[0] - \mathbf{f}[1] + \mathbf{f}[2] - \mathbf{f}[3] \\
\mathbf{F}[3] &= \mathbf{f}[0] + \mathbf{i}*\mathbf{f}[1] - \mathbf{f}[2] - \mathbf{i}*\mathbf{f}[3]
\end{aligned}
$$

Again, there's a clue for an algorithm here. Let **N** = $4^m = 4 * 4^{m-1}$, so in (7) put $\mathbf{n_1} = 4$ and $\mathbf{n_2} = 4^{m-1}$ = **N**/4:

$$
\begin{aligned}
\mathbf{F_N}[N/4*r+s] &= \sum_{q=0}^{4-1} w_4^{qr} w_N^{qs} \sum_{p=0}^{N/4-1} f[4*p+q]\, w_{(N/4)}^{ps} \\
&= \sum_{q=0}^{3} w_4^{qr} w_N^{qs} F_{(N/4)}[4,q,s] \\
&= F_{(N/4)}[4,0,s] \\
&\quad + w_4^r w_N^s F_{(N/4)}[4,1,s] \\
&\quad + (w_4^r)^2 (w_N^s)^2 F_{(N/4)}[4,2,s] \\
&\quad + (w_4^r)^3 (w_N^s)^3 F_{(N/4)}[4,3,s]
\end{aligned}
$$

or if we note that **r** goes from 0 to 3 then :

$$
\begin{aligned}
\mathbf{F_N}[s] = {} & F_{(N/4)}[4,0,s] + w_N^s F_{(N/4)}[4,1,s] \\
& + (w_N^s)^2 F_{(N/4)}[4,2,s] + (w_N^s)^3 F_{(N/4)}[4,3,s] \qquad (\mathbf{r=0})\ (11)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{F_N}[N/4+s] = {} & F_{(N/4)}[4,0,s] - i\, w_N^s F_{(N/4)}[4,1,s] \\
& - (w_N^s)^2 F_{(N/4)}[4,2,s] + i\, (w_N^s)^3 F_{(N/4)}[4,3,s] \qquad (\mathbf{r=1})\ (12)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{F_N}[N/2+s] = {} & F_{(N/4)}[4,0,s] - w_N^s F_{(N/4)}[4,1,s] \\
& + (w_N^s)^2 F_{(N/4)}[4,2,s] - (w_N^s)^3 F_{(N/4)}[4,3,s] \qquad (\mathbf{r=2})\ (13)
\end{aligned}
$$

$$\mathbf{F_N}[3N/4 + s] = F_{(N/4)}[4,0,s] + i\ w_N^s F_{(N/4)}[4,1,s] - (w_N^s)^2 F_{(N/4)}[4,2,s] - i\ (w_N^s)^3 F_{(N/4)}[4,3,s] \quad (r = 3)\ (14)$$

as $w_2^0 = 1$, $w_2^1 = -i$, $w_2^2 = -1$, $w_2^3 = i$, $w_2^4 = 1$, $w_2^6 = -1$, $w_2^9 = -i$

Thus you compare/contrast with the radix-2 analysis. If you ***already*** have $\mathbf{F_{N/4}}$ calculated then you can ***build*** $\mathbf{F_N}$ by combining components indexed **0** modulo **4** ($F_{(N/4)}[4,0,s]$) with components indexed **1** modulo **4** ($F_{(N/4)}[4,1,s]$) and components indexed **2** modulo **4** ($F_{(N/4)}[4,2,s]$) and finally components indexed **3** modulo **4** ($F_{(N/4)}[4,3,s]$) via (11) - (14), then placing those combinations respectively in the ***first, second***, ***third*** and ***fourth*** quarters of $\mathbf{F_N}$. $\mathbf{F_{N/4}}$ has length **N**/4 and **s** ranges from 0 to **N**/4 -1, see (6), and $\mathbf{F_N}$ is fourfold that length ie. **N** and so will fit four sets of size **N**/4. Note that as :

s ranges from 0 to **N**/4 -1,

s + **N**/4 ranges from **N**/4 to **N**/4 + **N**/4 – 1 = **N**/2 -1,

s + **N**/2 ranges from **N**/2 to **N**/2 + **N**/4 – 1 = 3**N**/4 -1,

s + 3**N**/4 ranges from 3**N**/4 to 3**N**/4 + **N**/4 – 1 = **N** – 1

…... thus continuing the magic.

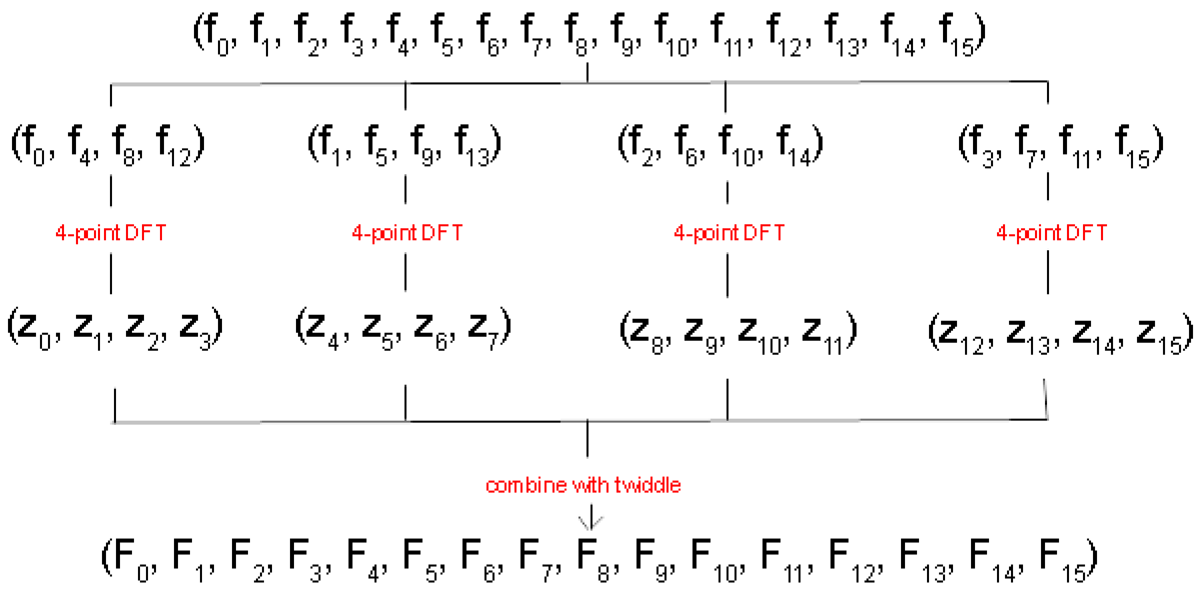
Likewise for radix-4 we may list the major tasks to be done by any code :

- DFT for size 4
- multiplication by the twiddle factors
- performing (11) through (14) and so effectively permuting the order of operands

Now the index permutations occur not with binary digit reversal, but ***quaternary*** digit reversal/reflection for radix-4, as demonstrated here for 16-points :

Original index in decimal	Original index in quaternary	Reversed index in quaternary	Reversed index in decimal
0	00	00	0
1	01	10	4
2	02	20	8
3	03	30	12
4	10	01	1
5	11	11	5
6	12	21	9
7	13	31	13
8	20	02	2
9	21	12	6
10	22	22	10
11	23	32	14
12	30	03	3
13	31	13	7
14	32	23	11
15	33	33	15

This then yields the following data flow for a 16-point radix-4 DFT :



which is 'squatter' than the radix-2 case. Here one has to do 4 by 4-point DFTs that being 4 *

16 = 64 multiplications, adding in 16 more for the twiddles totalling thus to 80. Compared to the bland 16-point DFT (256 multiplications) gives 80/256 ~ 32 %, BUT would we have got to those 4-point DFTs sooner (than radix-2) in execution time, and correspondingly quicker back through the twiddles ?? This is a matter to be resolved by future testing ….

Now the question arises as to the twiddle combinations which involve the complex numbers 1, -1, + **i** and – **i**. A reasonable representation for a complex number **Z** would be Cartesian pairs ie. [Re{Z}, Im{Z}] in which case such multiplications are simple moves and/or sign changes rather than computation involving non-integral floats :

if **Z** = [Re{Z}, Im{Z}]

then **i*Z =** [-Im{Z}, Re{Z}]

and - **i*Z** = [Im{Z}, -Re{Z}]

with **1*Z =** [Re{Z}, Im{Z}] (doh !)

and -**1*Z =** [-Re{Z}, -Im{Z}] (doh-er !)

Other radices would necessarily invoke (co-)sinusoidal components for Argand plane angles not a multiple of $\pi/2$.