PiFM/QRPi

• PI FM:
  http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter#Steps_to_play_sound:

• RPiTX:
  https://github.com/F5OEO/rpitx

• WsprryPi:
  https://github.com/JamesP6000/WsprryPi

• QRPi:
  http://rfsparkling.com/qrpi/

• qtcsdr:
  https://github.com/ha7ilm/qtcsdr
stations that around the world that spotted a beacon from a raspberry pi at my home in the last 24 hours. The beacon transmits a 2min message every 10min putting out only 0.1 Watt of power!
Last Time

- Discrete Fourier Transform
  - Linear convolution through circular
  - Linear convolutions through DFT
    - Overlap and add
    - Overlap and save

Today

- The Fast Fourier Transform
Circular convolution:

\[ h[n] \oplus x[n] = \begin{bmatrix} h[0] & h[N-1] & \cdots & h[1] \\ h[1] & h[0] & \ddots & h[2] \\ \vdots & \ddots & \ddots & \vdots \\ h[N-1] & h[N-2] & \cdots & h[0] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} = H_c x \]

- \( H_c \) is a circulant matrix
- The columns of the DFT matrix are Eigen vectors of circulant matrices.
- Eigen vectors are DFT coefficients. How can you show?

Proof in HW
Circular Convolution as Matrix Operation

- Diagonalize:
  \[ W_N H_c W_n^{-1} = \begin{bmatrix}
  H[0] & 0 & \cdots & 0 \\
  0 & H[1] & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & H[N-1]
  \end{bmatrix} \]

- Right-multiply by \( W_N \)
  \[ W_N H_c = \begin{bmatrix}
  H[0] & 0 & \cdots & 0 \\
  0 & H[1] & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & H[N-1]
  \end{bmatrix} W_N \]

- Multiply both sides by \( x \)
  \[ W_N H_c x = \begin{bmatrix}
  H[0] & 0 & \cdots & 0 \\
  0 & H[1] & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & H[N-1]
  \end{bmatrix} W_N x \]
We are interested in efficient computing methods for the DFT and inverse DFT:

\[
X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0, \ldots, N - 1
\]

\[
x[n] = \sum_{k=0}^{N-1} X[k] W_N^{-kn}, \quad n = 0, \ldots, N - 1
\]

where

\[
W_N = e^{-j\left(\frac{2\pi}{N}\right)}.
\]
Recall that we can use the DFT to compute the inverse DFT:

\[ \mathcal{DFT}^{-1}\{X[k]\} = \frac{1}{N} (\mathcal{DFT}\{X^*[k]\})^* \]

Hence, we can just focus on efficient computation of the DFT.

Straightforward computation of an \( N \)-point DFT (or inverse DFT) requires \( N^2 \) complex multiplications.
- **Fast Fourier transform algorithms** enable computation of an $N$-point DFT (or inverse DFT) with the order of just $N \cdot \log_2 N$ complex multiplications. This can represent a huge reduction in computational load, especially for large $N$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$N^2$</th>
<th>$N \cdot \log_2 N$</th>
<th>$\frac{N^2}{N \cdot \log_2 N}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>256</td>
<td>64</td>
<td>4.0</td>
</tr>
<tr>
<td>128</td>
<td>16,384</td>
<td>896</td>
<td>18.3</td>
</tr>
<tr>
<td>1,024</td>
<td>1,048,576</td>
<td>10,240</td>
<td>102.4</td>
</tr>
<tr>
<td>8,192</td>
<td>67,108,864</td>
<td>106,496</td>
<td>630.2</td>
</tr>
<tr>
<td>$6 \times 10^6$</td>
<td>$36 \times 10^{12}$</td>
<td>$135 \times 10^6$</td>
<td>$2.67 \times 10^5$</td>
</tr>
</tbody>
</table>

* 6Mp image size
Most FFT algorithms exploit the following properties of $W^k_n$:

- **Conjugate Symmetry**
  \[ W^k_{N}^{(N-n)} = W^{-kn}_N = (W^k_N)^* \]

- **Periodicity in $n$ and $k$**:
  \[ W^k_n = W^{k(n+N)}_N = W^{(k+N)n}_N \]

- **Power**:
  \[ W^2_N = W^{N/2}_N \]
Most FFT algorithms decompose the computation of a DFT into successively smaller DFT computations.

- *Decimation-in-time* algorithms decompose $x[n]$ into successively smaller subsequences.
- *Decimation-in-frequency* algorithms decompose $X[k]$ into successively smaller subsequences.

We mostly discuss *decimation-in-time* algorithms here.

Assume length of $x[n]$ is power of 2 ($N = 2^\nu$). If smaller zero-pad to closest power.
Decimation-in-Time Fast Fourier Transform

- We start with the DFT

\[ X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0, \ldots, N - 1 \]

- Separate the sum into even and odd terms:

\[ X[k] = \sum_{n \text{ even}} x[n] W_N^{kn} + \sum_{n \text{ odd}} x[n] W_N^{kn} \]

These are two DFT’s, each with half of the samples.
Decimation-in-Time Fast Fourier Transform

Let \( n = 2r \) (\( n \) even) and \( n = 2r + 1 \) (\( n \) odd):

\[
X[k] = \sum_{r=0}^{(N/2)-1} x[2r]W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r + 1]W_N^{(2r+1)k}
\]

\[
= \sum_{r=0}^{(N/2)-1} x[2r]W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r + 1]W_N^{2rk}
\]

- Note that:

\[
W_N^{2rk} = e^{-j\left(\frac{2\pi}{N}\right)(2rk)} = e^{-j\left(\frac{2\pi}{N/2}\right)rk} = W_{N/2}^r
\]

Remember this trick, it will turn up often.
Decimation-in-Time Fast Fourier Transform

Hence:

\[
X[k] = \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^r + W_N^k \sum_{r=0}^{(N/2)-1} x[2r + 1] W_{N/2}^r
\]

\[\triangleq G[k] + W_N^k H[k], \quad k = 0, \ldots, N - 1\]

where we have defined:

\[
G[k] \triangleq \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^r \quad \Rightarrow \text{DFT of even idx}
\]

\[
H[k] \triangleq \sum_{r=0}^{(N/2)-1} x[2r + 1] W_{N/2}^r \quad \Rightarrow \text{DFT of odd idx}
\]
Decimation-in-Time Fast Fourier Transform

An 8 sample DFT can then be diagrammed as

Even Samples
- x[0]
- x[2]
- x[4]
- x[6]

Odd Samples
- x[1]
- x[3]
- x[5]
- x[7]

N/2 - Point DFT

G[0]
G[1]
G[2]
G[3]

H[0]
H[1]
H[2]
H[3]

W_N^0
W_N^1
W_N^2
W_N^3
W_N^4
W_N^5
W_N^6
W_N^7

X[0]
X[1]
X[2]
X[3]
X[4]
X[5]
X[6]
X[7]

Miki Lustig UCB. Based on Course Notes by J.M Kahn EE123 Digital Signal Processing
Both \( G[k] \) and \( H[k] \) are periodic, with period \( N/2 \). For example

\[
G[k + N/2] = \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{r(k+N/2)}
\]

\[
= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} W_{N/2}^{r(N/2)}
\]

\[
= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk}
\]

\[
= G[k]
\]

So

\[
G[k + (N/2)] = G[k]
\]

\[
H[k + (N/2)] = H[k]
\]
The periodicity of $G[k]$ and $H[k]$ allows us to further simplify. For the first $N/2$ points we calculate $G[k]$ and $W_N^k H[k]$, and then compute the sum

$$X[k] = G[k] + W_N^k H[k] \quad \forall \{k : 0 \leq k < \frac{N}{2}\}.$$ 

How does periodicity help for $\frac{N}{2} \leq k < N$?
\[ X[k] = G[k] + W_N^k H[k] \quad \forall \{k : 0 \leq k < \frac{N}{2}\}. \]

- for \( \frac{N}{2} \leq k < N \):

\[ W_N^{k+(N/2)} = ? \]

\[ X[k + (N/2)] = ? \]
Decimation-in-Time Fast Fourier Transform

\[ X[k + (N/2)] = G[k] - W_N^k H[k] \]

We previously calculated \( G[k] \) and \( W_N^k H[k] \).

Now we only have to compute their difference to obtain the second half of the spectrum. No additional multiplies are required.
The $N$-point DFT has been reduced two $N/2$-point DFTs, plus $N/2$ complex multiplications. The 8 sample DFT is then:
Note that the inputs have been reordered so that the outputs come out in their proper sequence.

We can define a *butterfly operation*, e.g., the computation of $X[0]$ and $X[4]$ from $G[0]$ and $H[0]$:

\[
G[0] = G[0] + W_N^0 H[0]
\]

\[
X[0] = G[0] + W_N^0 H[0]
\]

\[
H[0] \rightarrow W_N^0 
\]

\[
X[4] = G[0] - W_N^0 H[0]
\]

This is an important operation in DSP.
Still $O(N^2)$ operations..... What shall we do?
We can use the same approach for each of the $N/2$ point DFT’s. For the $N = 8$ case, the $N/2$ DFTs look like

*Note that the inputs have been reordered again.*
At this point for the 8 sample DFT, we can replace the \( N/4 = 2 \) sample DFT’s with a single butterfly.

The coefficient is

\[
W_{N/4} = W_{8/4} = W_2 = e^{-j\pi} = -1
\]

The diagram of this stage is then

```
x[0]  \quad -1 \quad  \quad x[0] + x[4]
 x[0] - x[4]  \quad 1
```

Miki Lustig UCB. Based on Course Notes by J.M Kahn
Combining all these stages, the diagram for the 8 sample DFT is:

This is the decimation-in-time FFT algorithm.
In general, there are $\log_2 N$ stages of decimation-in-time.

Each stage requires $N/2$ complex multiplications, some of which are trivial.

The total number of complex multiplications is $(N/2) \log_2 N$.

The order of the input to the decimation-in-time FFT algorithm must be permuted.

- First stage: split into odd and even. Zero low-order bit first
- Next stage repeats with next zero-lower bit first.
- Net effect is reversing the bit order of indexes
This is illustrated in the following table for $N = 8$.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Bit-Reversed Binary</th>
<th>Bit-Reversed Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>
Decimation-in-Frequency Fast Fourier Transform

The DFT is

\[ X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \]

If we only look at the even samples of \( X[k] \), we can write \( k = 2r \),

\[ X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r)} \]

We split this into two sums, one over the first \( N/2 \) samples, and the second of the last \( N/2 \) samples.

\[ X[2r] = \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2r(n+N/2)} \]
But \( W_N^{2r(n+N/2)} = W_N^{2rn} W_N^N = W_N^{2rn} = W_{N/2}^{rn} \).

We can then write

\[
X[2r] = \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2r(n+N/2)}
\]

\[
= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2rn}
\]

\[
= \sum_{n=0}^{(N/2)-1} (x[n] + x[n + N/2]) W_{N/2}^{rn}
\]

This is the \( N/2 \)-length DFT of first and second half of \( x[n] \) summed.
Decimation-in-Frequency Fast Fourier Transform

\[
X[2r] = \text{DFT}_{N/2} \left\{ (x[n] + x[n + N/2]) \right\}
\]

\[
X[2r + 1] = \text{DFT}_{N/2} \left\{ (x[n] - x[n + N/2]) W_N^n \right\}
\]

(By a similar argument that gives the odd samples)

Continue the same approach is applied for the \(N/2\) DFTs, and the \(N/4\) DFT’s until we reach simple butterflies.
The diagram for an 8-point decimation-in-frequency DFT is as follows:

This is just the decimation-in-time algorithm reversed! The inputs are in normal order, and the outputs are bit reversed.
Non-Power-of-2 FFT’s

A similar argument applies for any length DFT, where the length \( N \) is a composite number.

For example, if \( N = 6 \), a decimation-in-time FFT could compute three 2-point DFT’s followed by two 3-point DFT’s.
Non-Power-of-2 FFT’s

Good component DFT’s are available for lengths up to 20 or so. Many of these exploit the structure for that specific length. For example, a factor of

\[ W_N^{N/4} = e^{-j \frac{2\pi}{N} (N/4)} = e^{-j \frac{\pi}{2}} = -j \]

just swaps the real and imaginary components of a complex number, and doesn’t actually require any multiplies. Hence a DFT of length 4 doesn’t require any complex multiplies. Half of the multiplies of an 8-point DFT also don’t require multiplication.

Composite length FFT’s can be very efficient for any length that factors into terms of this order.
For example $N = 693$ factors into

\[ N = (7)(9)(11) \]

each of which can be implemented efficiently. We would perform

- $9 \times 11$ DFT’s of length 7
- $7 \times 11$ DFT’s of length 9, and
- $7 \times 9$ DFT’s of length 11
Historically, the power-of-two FFTs were much faster (better written and implemented).

For non-power-of-two length, it was faster to zero pad to power of two.

Recently this has changed. The free FFTW package implements very efficient algorithms for almost any filter length. Matlab has used FFTW since version 6.
FFT computation time (Matlab FFTW) on MacBookPro

- N: Number of points
- Run time [ms]: Computation time in milliseconds

Key points:
- N = 192, Run time ≈ 0.01 ms
- N = 224, Run time ≈ 0.015 ms
- N = 256, Run time ≈ 0.02 ms
- N = 288, Run time ≈ 0.015 ms

Overall, the run time increases as the number of points (N) increases.
**FFT as Matrix Operation**

\[
\begin{pmatrix}
  x[0] \\
  \vdots \\
  x[k] \\
  \vdots \\
  x[N-1]
\end{pmatrix}
= \begin{pmatrix}
  W_N^{00} & \cdots & W_N^{0n} & \cdots & W_N^{0(N-1)} \\
  \vdots & \ddots & \vdots & \ddots & \vdots \\
  W_N^{k0} & \cdots & W_N^{kn} & \cdots & W_N^{k(N-1)} \\
  \vdots & \ddots & \vdots & \ddots & \vdots \\
  W_N^{(N-1)0} & \cdots & W_N^{(N-1)n} & \cdots & W_N^{(N-1)(N-1)}
\end{pmatrix}
\begin{pmatrix}
  x[0] \\
  \vdots \\
  x[n] \\
  \vdots \\
  x[N-1]
\end{pmatrix}
\]

- $W_N$ is fully populated $\Rightarrow N^2$ entries.
FFT as Matrix Operation

\[
\begin{pmatrix}
X[0] \\
\vdots \\
X[k] \\
\vdots \\
X[N-1]
\end{pmatrix} =
\begin{pmatrix}
W_N^{00} & \cdots & W_N^{0n} & \cdots & W_N^{0(N-1)} \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
W_N^{k0} & \cdots & W_N^{kn} & \cdots & W_N^{k(N-1)} \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
W_N^{(N-1)0} & \cdots & W_N^{(N-1)n} & \cdots & W_N^{(N-1)(N-1)}
\end{pmatrix}
\begin{pmatrix}
x[0] \\
\vdots \\
x[n] \\
\vdots \\
x[N-1]
\end{pmatrix}
\]

- $W_N$ is fully populated $\Rightarrow N^2$ entries.
- FFT is a decomposition of $W_N$ into a more sparse form:

\[
F_N = \begin{bmatrix}
I_{N/2} & D_{N/2} \\
I_{N/2} & -D_{N/2}
\end{bmatrix}
\begin{bmatrix}
W_{N/2} & 0 \\
0 & W_{N/2}
\end{bmatrix}
\begin{bmatrix}
\text{Even-Odd Perm. Matrix}
\end{bmatrix}
\]

- $I_{N/2}$ is an identity matrix. $D_{N/2}$ is a diagonal with entries $1, W_N, \ldots, W_N^{N/2-1}$

Miki Lustig UCB. Based on Course Notes by J.M Kahn

EE123 Digital Signal Processing
FFT as Matrix Operation

Example: $N = 4$

$$F_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & W_4 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -W_4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Beyond NlogN

• What if the signal $x[n]$ has a $k$ sparse frequency
  – A. Gilbert et. al, “Near-optimal sparse Fourier representations via sampling
  – H. Hassanieh et. al, “Nearly Optimal Sparse Fourier Transform”
  – Others......

• $O(K \log N)$ instead of $O(N \log N)$

From: http://groups.csail.mit.edu/netmit/sFFT/paper.html