CS601: Software Development for Scientific Computing

Autumn 2022

Week7: Motifs – Sparse Matrices (contd.), Fourier Transforms

Last week..

- Matrix Multiplication
  - ijk variants and recursive matmul
- Efficiency considerations
  - Storage (e.g. cache-oblivious data storage using Z-ordering)
  - Communication cost (data movement cost)
  - Special hardware (FMA, Vector units)
- Motif: Sparse Matrices
  - Triangular Matmul (as an e.g. that exploits structure to accelerate computation)
  - Storage scheme for sparse matrices (e.g. CSR)
  - Banded matrices (y=y+Ax with banded matrix and optimized storage)

y=y+Ax with *Separable* Matrices

Refer to (Section 1 only):

https://www.math.uci.edu/~chenlong/MathPKU/FMMsimple.pdf

Matrix Algebra and Efficient Computation

- **Pic source: the Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)**

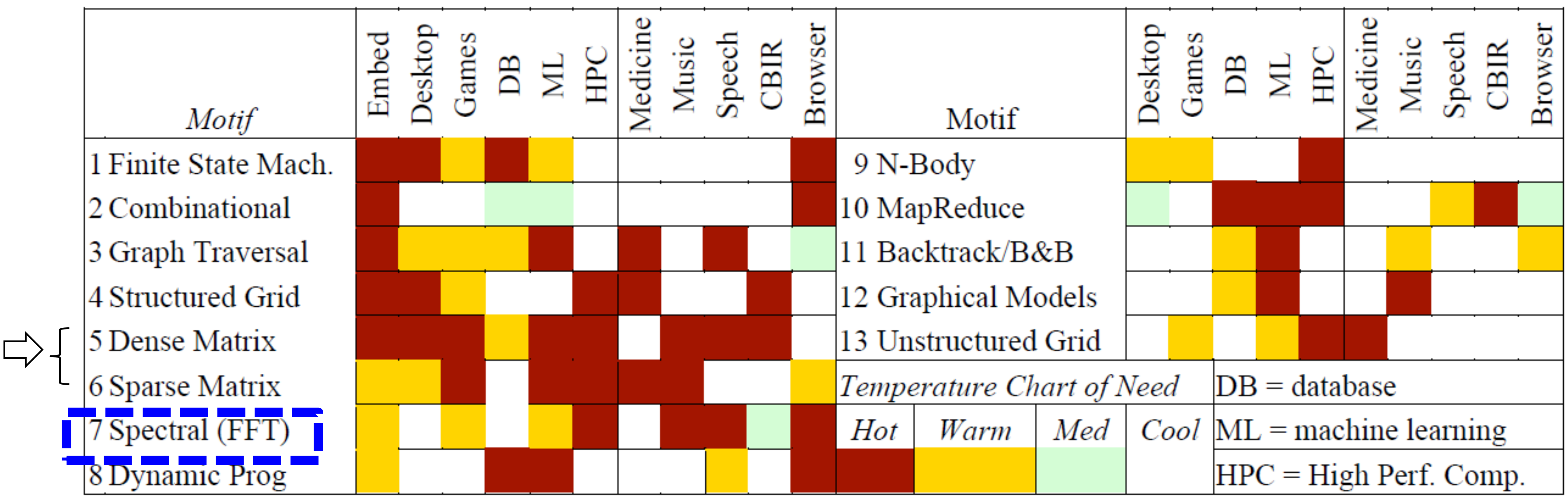


Figure 4. Temperature Chart of the 13 Motifs. It shows their importance to each of the original six application areas and then how important each one is to the five compelling applications of Section 3.1. More details on the motifs can be found in (Asanovic, Bodik et al. 2006).

Seen earlier

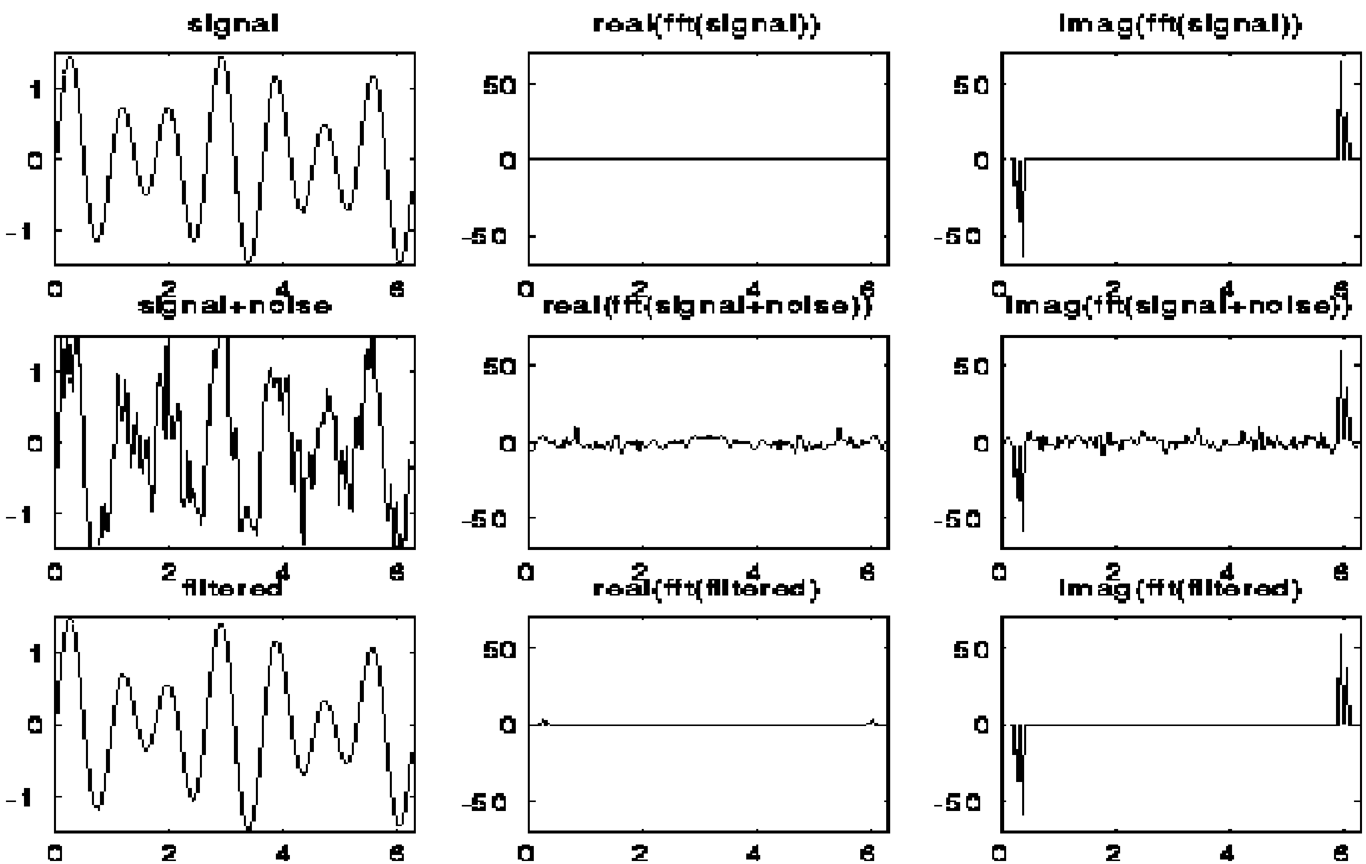
Next..

Faster y=Ax: Discrete Fourier Transforms (DFT)

- Very widely used
  - Image compression (jpeg)
  - Signal processing
  - Solving Poisson's Equation
- Represent A with F, a *Fourier Matrix* that has the following (remarkable) properties:
  - F^{-1} is easy to compute
  - Multiplications by F and F^{-1} is fast. (need to do Fx=y and x= F^{-1} y quickly)
- F has complex numbers in its entries.
  - Every entry is a power of a single number w such that $w^n=1$
  - Any entry of a Fourier matrix can be written using $f_{ij} = w^{ij}$ (row and col indices start from 0)

Using the 1D FFT for filtering

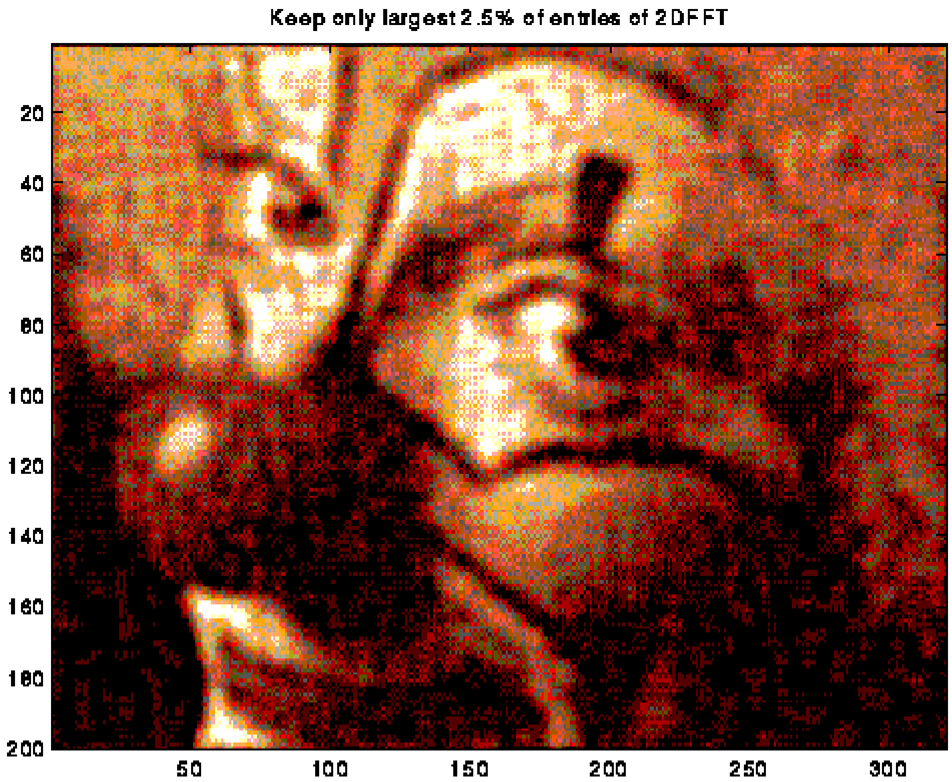
- Signal = sin(7t) + .5 sin(5t) at 128 points
- Noise = random number bounded by .75
- Filter by zeroing out FFT components < .25



Slide courtesy: James Demmel (CS267, Spring19) www.cs.berkeley.edu/~demmel

Using the 2D FFT for image compression

- Image = 200x320 matrix of values
- Compress by keeping largest 2.5% of FFT components
- Similar idea used by jpeg



Examples: Fourier Matrix

- 4x4: $\boldsymbol{F_4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & 1 & w^2 \\ 1 & w^3 & w^2 & w^1 \end{bmatrix}, i = \sqrt{-1}$
  - Here, $w = i$ (also denoted as w_4=i). $w^4 = 1$ => i is a root.

- 8x8: $\boldsymbol{F_8} =$

  Here, $w = \frac{1+i}{\sqrt{2}}$ (= sqrt of i)

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & w^8 & w^{10} & w^{12} & w^{14} \\ 1 & w^3 & w^6 & w^9 & w^{12} & w^{15} & w^{18} & w^{21} \\ 1 & w^4 & w^8 & w^{12} & w^{16} & w^{20} & w^{24} & w^{28} \\ 1 & w^5 & w^{10} & w^{15} & w^{20} & w^{25} & w^{30} & w^{35} \\ 1 & w^6 & w^{12} & w^{18} & w^{24} & w^{30} & w^{36} & w^{42} \\ 1 & w^7 & w^{14} & w^{21} & w^{28} & w^{35} & w^{42} & w^{49} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix}$$

Example: Faster y=Fx

$$
\begin{array}{c}
\begin{array}{cccccccc} \text{Column: } 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\
1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\
1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\
1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\
1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\
1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\
1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1
\end{bmatrix}
\end{array}
=
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & \omega^2 & \omega^4 & \omega^6 & \omega & \omega^3 & \omega^5 & \omega^7 \\
1 & \omega^4 & 1 & \omega^4 & \omega^2 & \omega^6 & \omega^2 & \omega^6 \\
1 & \omega^6 & \omega^4 & \omega^2 & \omega^3 & \omega & \omega^7 & \omega^5 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & \omega^2 & \omega^4 & \omega^6 & -\omega & -\omega^3 & -\omega^5 & -\omega^7 \\
1 & \omega^4 & 1 & \omega^4 & -\omega^2 & -\omega^6 & -\omega^2 & -\omega^6 \\
1 & \omega^6 & \omega^4 & \omega^2 & -\omega^3 & -\omega & -\omega^7 & -\omega^5
\end{bmatrix}
$$

(Writing columns 1,3,5,7 first and then columns 2,4,6,8)

Example: Faster y=Fx

- $\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix}$ $= \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega & \omega^3 & \omega^5 & \omega^7 \\ 1 & \omega^4 & 1 & \omega^4 & \omega^2 & \omega^6 & \omega^2 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & \omega^3 & \omega & \omega^7 & \omega^5 \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & \omega^2 & \omega^4 & \omega^6 & -\omega & -\omega^3 & -\omega^5 & -\omega^7 \\ 1 & \omega^4 & 1 & \omega^4 & -\omega^2 & -\omega^6 & -\omega^2 & -\omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & -\omega^3 & -\omega & -\omega^7 & -\omega^5 \end{array}\right]$

(Partitioning into 4 matrix blocks of size 4x4.)

Recall: $F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix}$, where $w = i = w_4$

Note: in F_8, w $= \frac{1+i}{\sqrt{2}} = w_8$

therefore, $w_8^2 = w_4$

Example: Faster y=Fx

- $$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix} = \left[\begin{array}{c|c} F_4 & \Omega_4 F_4 \\ \hline F_4 & -\Omega_4 F_4 \end{array}\right]$$

(because $w^2 = w_4$)

$$\Omega_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & w & 0 & 0 \\ 0 & 0 & w^2 & 0 \\ 0 & 0 & 0 & w^3 \end{bmatrix}$$ (note: $w = \frac{1+i}{\sqrt{2}} = w_8$)

- So, $F_8 = \begin{bmatrix} F_4 & \Omega_4 F_4 \\ F_4 & -\Omega_4 F_4 \end{bmatrix}$

FFT

We can obtain 8-point DFT from 4-point DFT. But how do we obtain the result of $y=F_8x$, from $y_{top} = \boxed{F_4x_{odd}}$ and $y_{bottom} = \boxed{F_4x_{even}}$?

$$\overset{F_8}{\begin{bmatrix} F_4 & \Omega_4 F_4 \\ \hline F_4 & -\Omega_4 F_4 \end{bmatrix}} \overset{x}{\begin{bmatrix} x1 \\ x3 \\ x5 \\ x7 \\ \hline x2 \\ x4 \\ x6 \\ x8 \end{bmatrix}} = \begin{bmatrix} I_4 & \Omega_4 \\ \hline I_4 & -\Omega_4 \end{bmatrix} \begin{bmatrix} \boxed{\begin{bmatrix} F_4 \end{bmatrix} \begin{bmatrix} x1 \\ x3 \\ x5 \\ x7 \end{bmatrix}} \\ \hline \boxed{\begin{bmatrix} F_4 \end{bmatrix} \begin{bmatrix} x2 \\ x4 \\ x6 \\ x8 \end{bmatrix}} \end{bmatrix} = \overset{y}{\begin{bmatrix} y1 \\ y3 \\ y5 \\ y7 \\ \hline y2 \\ y4 \\ y6 \\ y8 \end{bmatrix}}$$

Note: can be done with 4 multiplications

$y1$ to $y4 = y_{top} + \Omega_4 * y_{bottom}$

$y5$ to $y8 = y_{top} - \Omega_4 * y_{bottom}$

$y_{top} = F_4x_{odd}$

$y_{bottom} = F_4x_{even}$

(x_{odd}= elements at odd numbered indices of vector x)

(x_{even}= elements at even numbered indices of vector x)

Divide-and-Conquer FFT (D&C FFT)

FFT(v, ϖ, m) … assume m is a power of 2

if m = 1 return v[0]

else

v_{even} = FFT(v[0:2:m-2], ϖ^2, m/2)

v_{odd} = FFT(v[1:2:m-1], ϖ^2, m/2)

ϖ-vec = [ϖ^0, ϖ^1, … $\varpi^{(m/2-1)}$] precomputed

return [v_{even} + (ϖ-vec .* v_{odd}),

v_{even} - (ϖ-vec .* v_{odd})]

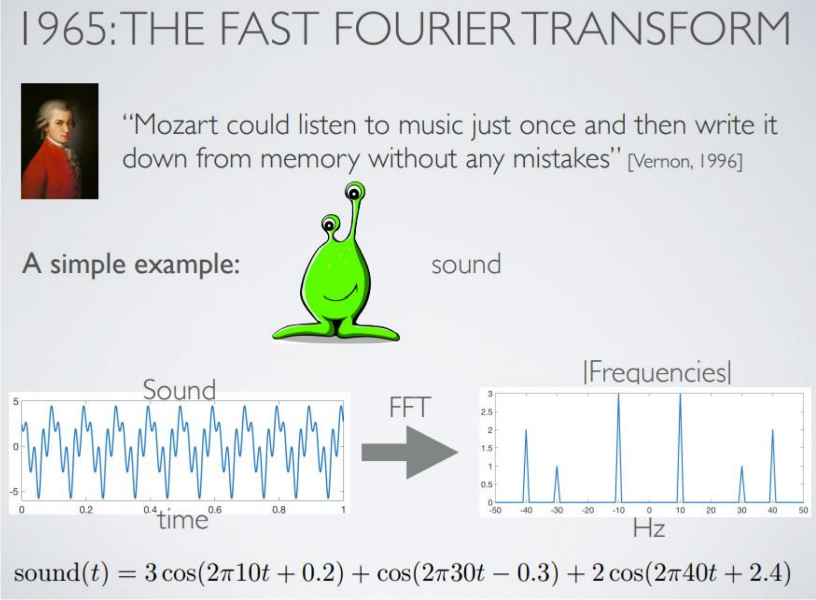
- Matlab notation: ".*" means component-wise multiply.

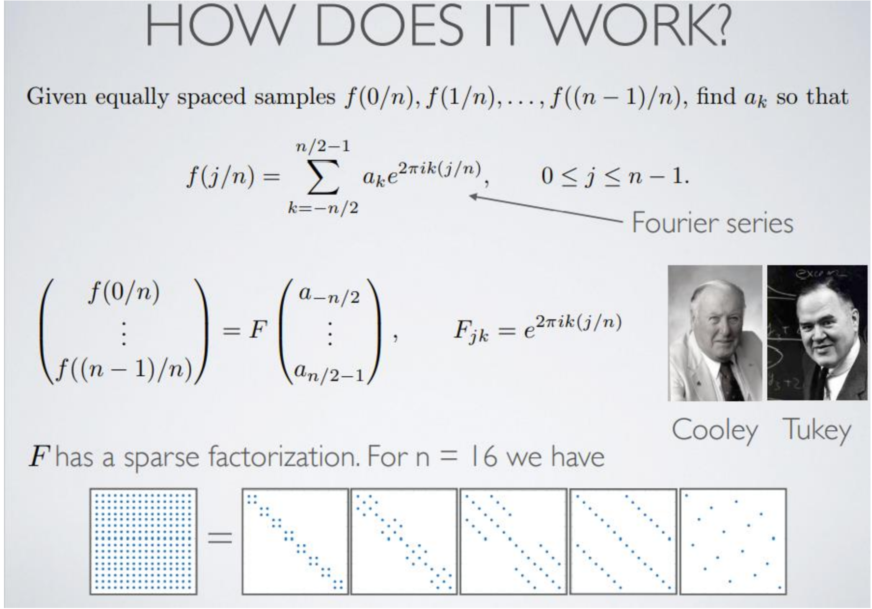
Cost: T(m) = 2T(m/2)+O(m) = O(m log m) operations.

Popularized/published by Cooley-Tuckey in 1965.

FFT - Summary

- We will revisit FFT when solving Poisson's equation
- 2-slide summary (**courtesy**: Alex Townsend, Cornell. Source)





- References:
  - Refer to Lecture 20 (Spring 2018) at https://inst.eecs.berkeley.edu/~cs267/archives.html
  - Section 1.4, Matrix Computations, 4th Ed, Golub and Van Loan
  - Section 3.5, Linear Algebra and Its Applications, 4th Ed, Gilbert Strang