Last time

- Matrix formula language to describe structured matrices (= transform algorithms)
  basic matrices: DFT_2, I_n, $diag(..)$, L^{kn}_k
  matrix operators: $A \cdot B$, $A \oplus B$, $I_n \otimes A$, $A \otimes I_n$
- Cooley-Tukey FFT (recursive, DIT)

$$DFT_{km} = (DFT_k \otimes I_m)\, T^{km}_m\, (I_k \otimes DFT_m)\, L^{km}_k$$

  Cost: $n \log_2(n) + O(n)$ complex adds
  $(n = 2^k)$ $\frac{1}{2} n \log_2(n) + O(n)$ complex mults
  independent of chosen recursion strategy
  What about the real cost?

Complex arithmetic (transforms)

$y = Tx$, T complex $n \times n$ matrix, $x, y \in \mathbb{C}^n$

How to implement on a computer?

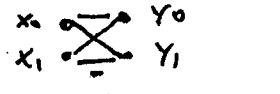
- convert complex vectors $\in \mathbb{C}^n$ into real vectors $\in \mathbb{R}^{2n}$. Two formats:
  - split complex: $(re(x_0), \ldots, re(x_{n-1}), im(x_0), \ldots, im(x_{n-1}))$
    (one pointer or two pointers for re and im)
  - interleaved complex: $(re(x_0), im(x_0), \ldots, re(x_{n-1}), im(x_{n-1}))$
    (usually better data locality)
- convert complex ops to real ops:
  $u = u_1 + j u_2$, $v = v_1 + j v_2$

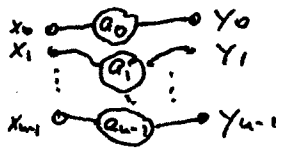
complex	real	
$u = v + w$	$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$	
1 add	2 adds	
$u = vw$	$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} v_1 & -v_2 \\ v_2 & v_1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$	v is constant in transforms
1 mult	4 mults, 2 adds = 6 ops	

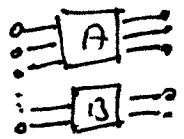
- real cost of Cooley-Tukey FFT:
  $2 n \log_2(n) + O(n)$ + $3 n \log_2(n) + O(n) = 5 n \log_2(n) + O(n)$

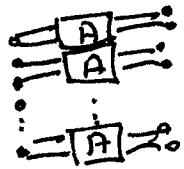
~~different views~~ Math formulas → code

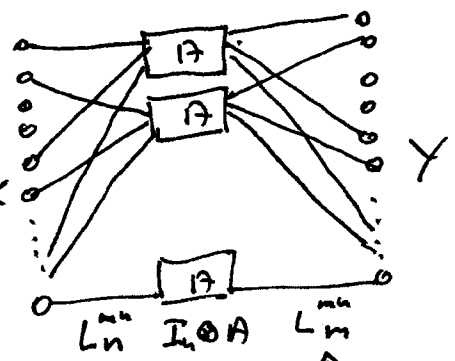
structure | matrix | dataflow | pseudo code

DFT_2 | $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ |  | $y_0 = x_0 + x_1$, $y_1 = x_0 - x_1$ — butterfly

$diag(a_0, \dots, a_{n-1})$ | $\begin{pmatrix} a_0 & & \\ & a_1 & \\ & & \ddots \end{pmatrix}$ |  | for $i = (0:1:n-1)$ $y_i = a_i x_i$ — scaling

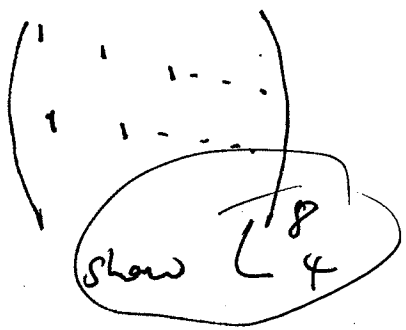
$A_{m \times m} \oplus B_{n \times n}$ | $\begin{pmatrix} A & \\ & B \end{pmatrix}$ |  | $y(0:1:m-1) = A \cdot x(0:1:m-1)$, $y(m:1:m+n-1) = B \cdot x(m:1:m+n-1)$ — parallel

$I_n \otimes A_{m \times m}$ | $\begin{pmatrix} A & & \\ & A & \\ & & \ddots \end{pmatrix}$ |  | for $i = 0:1:n-1$ $y(im:1:im+m-1) = A x(im:1:im+m-1)$ — loopable, n A's in parallel

$A_{m \times m} \otimes I_n$ | $\begin{pmatrix} a_{00} I_n & a_{01} I_n & \cdots \\ a_{10} I_n & \ddots & \\ \vdots & & \end{pmatrix}$ |  | for $i = 0:1:n-1$ $y(i:n:i+(m-1)n) = A x(i:n:i+(m-1)n)$ — loopable, n A's in parallel at stride n; vectorizable

to draw dataflow use $A \otimes I_n = L^{mn}_m (I_n \otimes A) L^{mn}_n$

L^{kn}_k, $n = km$



does the same, but upper one without mod

```
for i = 0:1:k-1
    for j = 0:1:m-1
        y[i+mj] = x[jk+i]
```

or

```
for i = 0:1:n-2
    y[i] = x[ki mod n-1]
y[n-1] = x[n-1]
```

} read at stride k

cost analysis: (counting adds and mults)

$C(n) = (A(n), M(n))$

careful: for complex matrices this counts complex ops which is different from real ops

$cost(DFT_2) = (2,0)$

$cost(I_n) = (0,0)$

$cost(L^n_k) = (0,0)$

$cost(A \oplus B) = cost(A) + cost(B)$

$cost(\bigoplus_{i=0}^{n-1} (q_i)) \leq$ Computation $= (0,k)$, $k = \#$ q_i's $\neq \pm 1$

$cost(I_n \otimes A) = n \cdot cost(A)$

$A \otimes I_n$

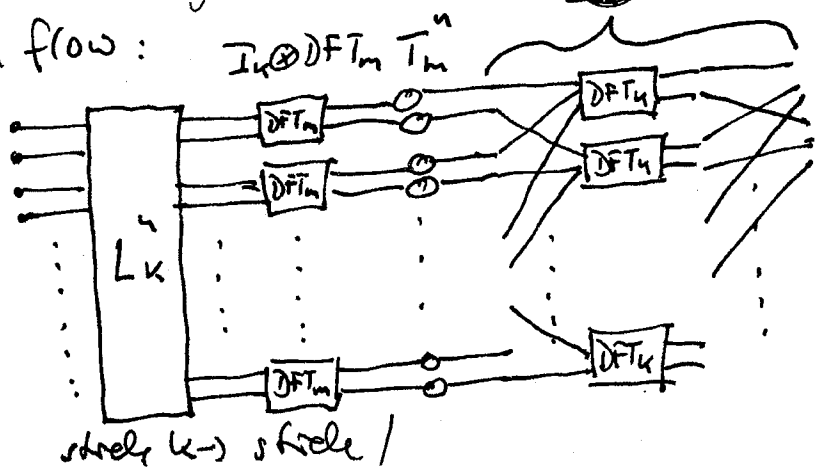
2.) Cooley-Tukey FFT ("the" FFT)

a.) $DFT_{km} = (DFT_k \otimes I_m)\, T^n_m\, (I_k \otimes DFT_m)\, L^n_k$

$T^n_m = \bigoplus_{i=0}^{n-1} diag(1, \omega_n^i, \dots, \omega_n^{(m-1)i})$ "twiddle factors"

Notes:
- the above version is called decimation-in-time (DIT)
- if recursively the same k is chosen it is called radix-k
- for a 2-power n, a.) yields an $O(n \log(n))$ algorithm

Data flow: $I_k \otimes DFT_m \; T^n_m$ $\quad DFT_m \otimes I_k$



derived cost $A + M$ mult 2 for radix 2

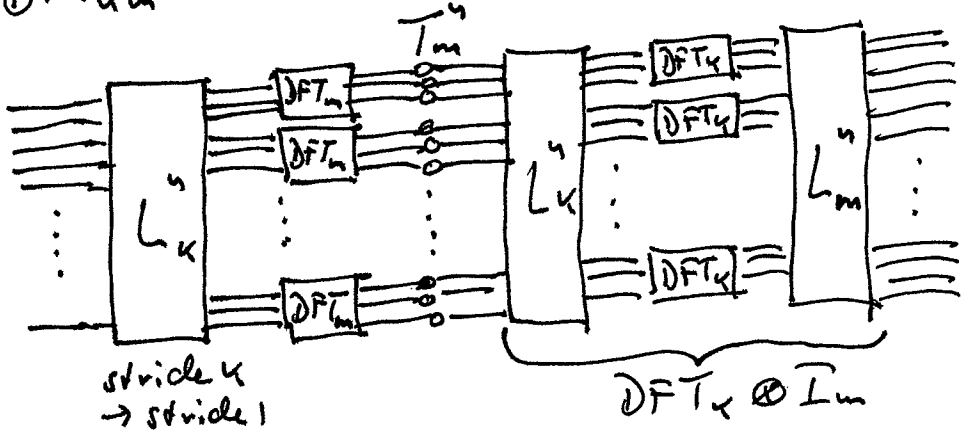
$A(n) = 2A(n/2) + n$

$M(n) = 2M(n/2) + \frac{n}{2} - 1$

Recursive vs. Iterative FFT

- recursive:

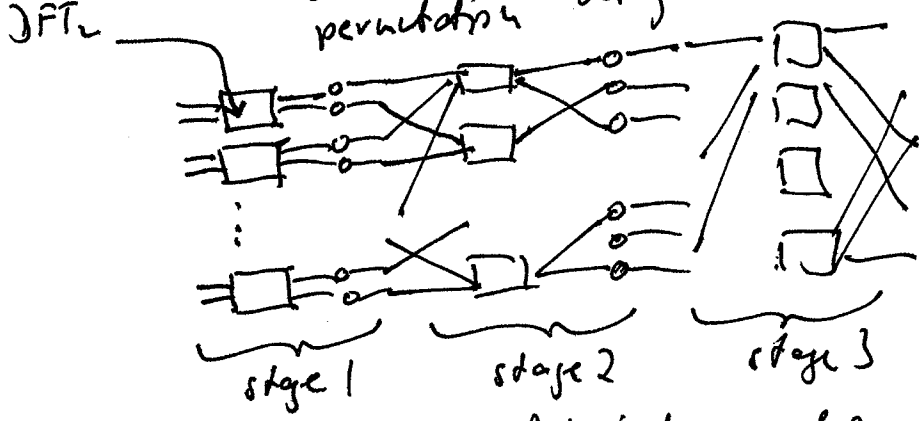
$$DFT_{km} = (I_k \otimes DFT_m)\, T^{km}_m\, (DFT_k \otimes I_m)\, L^{km}_k$$



- iterative: (that's what most people consider "the FFT")
  ($n = 2^t$)

$$DFT_{2^t} = R_{2^t} \prod_{i=1}^{t} (I_{2^{t-i}} \otimes T^{2^i}_{2^{i-1}})(I_{2^{t-i}} \otimes DFT_2 \otimes I_{2^{i-1}})$$

R_{2^t}: bit reversal permutation; $\prod_{i=1}^{t} (I_{2^{t-i}} \otimes T^{2^i}_{2^{i-1}})$: diagonal; $(I_{2^{t-i}} \otimes DFT_2 \otimes I_{2^{i-1}})$: 2^{t-1} butterflies at varying stride



$t = \log_2(n)$ stages

Assume: $\frac{n}{2}$ elements fit into cache

iterative FFT: every stage produces cache misses (since it passes through entire data)

recursive FFT: assume $k = 2$
only 2 stages produce cache misses

Analogy

MMM	DFT
by definition	iterative FFT
blocked	recursive FFT