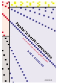
Sequential Nonrecursive Fast Fourier Transform (PSC §3.3)



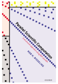
Pros and cons of recursive computations

Pros:

- display a natural splitting into subproblems, thus pointing to possible parallelism
- provide a concise formulation of the algorithm
- reduce the amount of bookkeeping

Cons:

- the corresponding computational tree is traversed sequentially, thus making parallelisation more difficult
- the corresponding tree may obscure potential shortcuts to parallelisation



Matrix decompositions

- If we decompose the matrix F_n into $F_n = A_{r-1} \cdots A_1 A_0$, where each factor A_k is an $n \times n$ matrix, we can obtain $F_n \mathbf{x}$ by repeatedly multiplying a matrix A_k and a vector:

$$F_n \mathbf{x} = A_{r-1} \cdots A_1 A_0 \mathbf{x}.$$

- Different decompositions represent different algorithms.
- Can the FFT be formulated as a matrix decomposition?
- Yes! Van Loan (Computational Frameworks for the FFT, SIAM, 1992) has formulated many variants of the FFT in terms of matrix decompositions.

Matrix and vector language for the FFT

- Define the $n \times n$ diagonal matrix

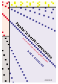
$$\Omega_n = \mathrm{diag}(1, \omega_{2n}, \omega_{2n}^2, \ldots, \omega_{2n}^{n-1}),$$

so that

$$\Omega_{n/2} = \mathrm{diag}(1, \omega_n, \omega_n^2, \ldots, \omega_n^{n/2-1}).$$

$\Omega_{n/2}$ contains exactly the powers of ω_n needed in the FFT.
- The recursive algorithm can now neatly be expressed by

$$\begin{aligned} F_n \mathbf{x} &= \begin{bmatrix} I_{n/2} & \Omega_{n/2} \\ I_{n/2} & -\Omega_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2} x(0:2:n-1) \\ F_{n/2} x(1:2:n-1) \end{bmatrix} \\ &= \begin{bmatrix} I_{n/2} & \Omega_{n/2} \\ I_{n/2} & -\Omega_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2} & 0 \\ 0 & F_{n/2} \end{bmatrix} \begin{bmatrix} x(0:2:n-1) \\ x(1:2:n-1) \end{bmatrix}. \end{aligned}$$

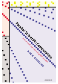


Even-odd sort matrix

The even-odd sort matrix S_n is the $n \times n$ permutation matrix containing rows $0, 2, \ldots, n-2$ of I_n followed by rows $1, 3, \ldots, n-1$,

$$S_n = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ & & \vdots & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ & & \vdots & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}.$$

Thus, $S_n\mathbf{x} = \begin{bmatrix} x(0:2:n-1) \\ x(1:2:n-1) \end{bmatrix}.$



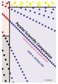
Kronecker matrix product

- Let A be a $q \times r$ matrix and B an $m \times n$ matrix. The Kronecker product (or tensor product, or direct product) of A and B is the $qm \times rn$ matrix

$$A \otimes B = \begin{bmatrix} a_{00}B & \cdots & a_{0,r-1}B \\ \vdots & & \vdots \\ a_{q-1,0}B & \cdots & a_{q-1,r-1}B \end{bmatrix}.$$

- Let $A = \begin{bmatrix} 0 & 1 \\ 2 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \end{bmatrix}$. Then

$$A \otimes B = \begin{bmatrix} 0 & B \\ 2B & 4B \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 0 & 4 & 4 & 0 & 8 \\ 0 & 2 & 0 & 0 & 4 & 0 \end{bmatrix}.$$



Useful properties

- Lemma 3.3 (Associativity) Let A, B, C be matrices. Then

$$(A \otimes B) \otimes C = A \otimes (B \otimes C).$$

- Lemma 3.4 Let A, B, C, D be matrices such that AC and BD are defined. Then

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD).$$

- Lemma 3.5 Let $m, n \in \mathbf{N}$. Then

$$I_m \otimes I_n = I_{mn}.$$

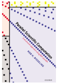
Commutativity?

- Lemma (Commutativity) Let A, B be matrices. Then

$$A \otimes B = B \otimes A.$$

Commutativity?

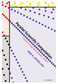
- Lemma (Commutativity) Let A, B be matrices. Then
$$A \otimes B = B \otimes A.$$
This lemma is not very useful, because it is false.
- Let $A = \begin{bmatrix} 2 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Then
$$\begin{aligned} A \otimes B &= \begin{bmatrix} 2B & 4B \end{bmatrix} = \begin{bmatrix} 2 & 0 & 4 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix}, \\ B \otimes A &= \begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix} = \begin{bmatrix} 2 & 4 & 0 & 0 \\ 0 & 0 & 2 & 4 \end{bmatrix}. \end{aligned}$$
Thus,
$$A \otimes B \neq B \otimes A.$$



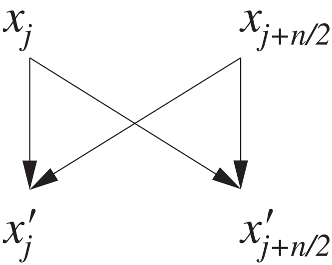
Use of Kronecker product for FFT

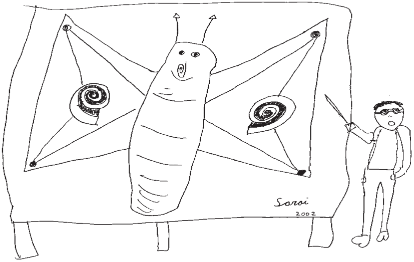
- Matrix notation and Kronecker products are powerful tools in modern Fourier transform research.
- Here, we use these tools to derive a nonrecursive variant of the FFT.
- Concise notation:

$$I_2 \otimes F_{n/2} = \begin{bmatrix} F_{n/2} & 0 \\ 0 & F_{n/2} \end{bmatrix}.$$



Butterfly operation





©Sarai Bisseling, 2002

$$
\begin{aligned}
x'_j &:= x_j + \omega_n^j x_{j+n/2};\\
x'_{j+n/2} &:= x_j - \omega_n^j x_{j+n/2};
\end{aligned}
$$

Butterfly matrix

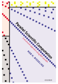
- The $n \times n$ butterfly matrix is

$$B_n = \begin{bmatrix} I_{n/2} & \Omega_{n/2} \\ I_{n/2} & -\Omega_{n/2} \end{bmatrix}.$$

- B_4 involves Ω_2, which contains powers of $\omega_4 = e^{-2\pi i/4} = -i$:

$$B_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -i \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & i \end{bmatrix}.$$

- The butterfly matrix is sparse since it has only $2n$ nonzeros out of n^2 elements.



T-shirt formula

Using the new notation gives

$$F_n\mathbf{x} = B_n(I_2 \otimes F_{n/2})S_n\mathbf{x}.$$

Since this holds for all vectors $\mathbf{x}$, we obtain an important T-shirt formula:

$$F_n = B_n(I_2 \otimes F_{n/2})S_n$$

Size reduction of the Fourier matrix

We try to reduce the size of the remaining Fourier matrix $F_{n/2}$. Thus we manipulate the factor $I_2 \otimes F_{n/2}$, or more in general, $I_k \otimes F_{n/k}$.

$$\begin{aligned}
I_k \otimes F_{n/k} &= [I_k I_k I_k] \otimes \left[B_{n/k}(I_2 \otimes F_{n/(2k)}) S_{n/k} \right] \\
&= (I_k \otimes B_{n/k})([I_k I_k] \otimes \left[(I_2 \otimes F_{n/(2k)}) S_{n/k} \right]) \\
&= (I_k \otimes B_{n/k})(I_k \otimes I_2 \otimes F_{n/(2k)})(I_k \otimes S_{n/k}) \\
&= (I_k \otimes B_{n/k})(I_{2k} \otimes F_{n/(2k)})(I_k \otimes S_{n/k}).
\end{aligned}$$

Burn at both ends

Repeatedly applying the factorisation of $I_k \otimes F_{n/k}$:

$$I_k \otimes F_{n/k} = (I_k \otimes B_{n/k})(I_{2k} \otimes F_{n/(2k)})(I_k \otimes S_{n/k}) =$$

$$(I_k \otimes B_{n/k})(I_{2k} \otimes B_{n/(2k)})(I_{4k} \otimes F_{n/(4k)})(I_{2k} \otimes S_{n/(2k)})(I_k \otimes S_{n/k}) = \cdots$$

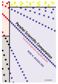
This ends when $I_n \otimes F_{n/n} = I_n \otimes F_1 = I_n \otimes I_1 = I_n$ is reached.

Starting with $F_n = I_1 \otimes F_n$ gives the Cooley-Tukey theorem (1965):

$$F_n = (I_1 \otimes B_n)(I_2 \otimes B_{n/2})(I_4 \otimes B_{n/4}) \cdots (I_{n/2} \otimes B_2) R_n,$$

where

$$R_n = (I_{n/2} \otimes S_2) \cdots (I_4 \otimes S_{n/4})(I_2 \otimes S_{n/2})(I_1 \otimes S_n).$$



Binary digits

- We can write an index j, $0 \leq j < n$, as

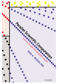
$$j = \sum_{k=0}^{m-1} b_k 2^k,$$

where $b_k \in \{0, 1\}$ is the kth bit and $n = 2^m$.

- b_0 is the least significant bit; b_{m-1} the most significant bit.
- We use the notation

$$(b_{m-1} \cdots b_1 b_0)_2 = \sum_{k=0}^{m-1} b_k 2^k.$$

- Example: $(10100101)_2 = 2^7 + 2^5 + 2^2 + 2^0 = 165$.



Bit-reversal permutation

Let $n = 2^m$, with $m \geq 1$. The bit-reversal permutation $\rho_n : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ is defined by

$$\rho_n((b_{m-1} \cdots b_0)_2) = (b_0 \cdots b_{m-1})_2.$$

For $n = 8$:

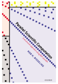
j	$(b_2 b_1 b_0)_2$	$(b_0 b_1 b_2)_2$	$\rho_8(j)$
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Bit-reversal algorithm

input: $\mathbf{x}$: vector of length $n = 2^m$, $m \geq 1$, $\mathbf{x} = \mathbf{x}_0$.
output: $\mathbf{x}$: vector of length n, such that $\mathbf{x} = R_n \mathbf{x}_0$.
call: bitrev($\mathbf{x}, n$).

for $j := 0$ **to** $n - 1$ **do**
  { Compute $r := \rho_n(j)$ }
  $q := j$;
  $r := 0$;
  for $k := 0$ **to** $\log_2 n - 1$ **do**
    $b_k := q \bmod 2$;
    $q := q \operatorname{div} 2$;
    $r := 2r + b_k$;
  if $j < r$ **then** swap(x_j, x_r);

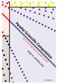
Based on Theorem 3.10: $R_n = P_{\rho_n}$ = permutation matrix corresponding to ρ_n. For a proof, see pp. 110–111.



Unordered FFT

input: $\mathbf{x}$: vector of length $n = 2^m$, $m \geq 1$, $\mathbf{x} = \mathbf{x}_0$.
output: $\mathbf{x}$: vector of length n, such that $\mathbf{x} = F_n R_n \mathbf{x}_0$.
call: UFFT$(\mathbf{x}, n)$.

$k := 2$;
while $k \leq n$ **do**
  $\{$ Compute $\mathbf{x} := (I_{n/k} \otimes B_k)\mathbf{x}$ $\}$
  for $r := 0$ **to** $\frac{n}{k} - 1$ **do**
    $\{$ Compute $x(rk: rk + k - 1) := B_k x(rk: rk + k - 1)$ $\}$
    for $j := 0$ **to** $\frac{k}{2} - 1$ **do**
      $\{$ Compute $x_{rk+j} \pm \omega_k^j x_{rk+j+k/2}$ $\}$
      $\tau := \omega_k^j x_{rk+j+k/2}$;
      $x_{rk+j+k/2} := x_{rk+j} - \tau$;
      $x_{rk+j} := x_{rk+j} + \tau$;
  $k := 2k$;



Summary

- We have derived a nonrecursive fast Fourier transform (FFT) by using matrix notation and the Kronecker matrix product.
- The result is the Cooley-Tukey Decimation In Time (DIT) formula

$$F_n = (I_1 \otimes B_n)(I_2 \otimes B_{n/2})(I_4 \otimes B_{n/4}) \cdots (I_{n/2} \otimes B_2) R_n.$$

- R_n is the permutation matrix that corresponds to the bit-reversal permutation ρ_n.
- Each of the $\log_2 n$ matrix factors $I_k \otimes B_{n/k}$ has $2n$ nonzero elements, and each corresponding matrix–vector multiplication requires $5n$ flops. Total number of flops: $5n \log_2 n$. Same as for the recursive FFT.
- The nonrecursive variant is a good basis for parallelisation.