

Sequential Nonrecursive FFT

Section 3.3 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

Utrecht University



Pros and cons of recursive computations

Pros. Recursive computations

- ▶ display a **natural splitting** into subproblems, thus pointing to possible parallelism;
- ▶ provide a **concise formulation** of the algorithm;
- ▶ reduce the amount of **bookkeeping**;
- ▶ are **cache-friendly**.

Cons. They

- ▶ traverse the corresponding computational tree **sequentially**, thus making parallelization more difficult.



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! Charles 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

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

so that

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

$\Omega_{n/2}$ contains exactly the powers of ω_n needed in the FFT.

- ▶ The recursive algorithm can now neatly be expressed by

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



Even-odd sort matrix

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

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

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



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 of the Kronecker product

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

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

- ▶ Lemma 3.4 (**Mixed products**) 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 (**Identities**) 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**.
- ▶ **Counterexample:** Let $A = [2 \ 4]$ and $B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Then

$$A \otimes B = [2B \ 4B] = \begin{bmatrix} 2 & 0 & 4 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix},$$

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

Thus,

$$A \otimes B \neq B \otimes A.$$



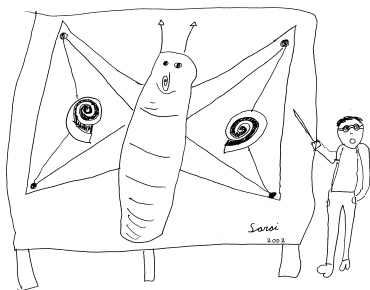
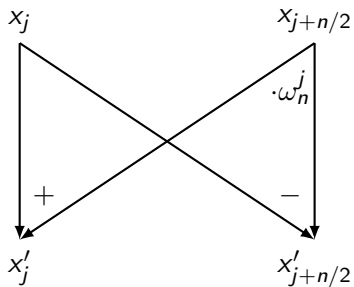
Use of Kronecker product for FFT

- ▶ 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.
- ▶ We benefit from a **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

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

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 get:

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



Size reduction of the Fourier matrix

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

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



Burn at both ends

- ▶ We repeatedly reduce the size:

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

- ▶ We start with

$$F_n = I_1 \otimes F_n.$$

- ▶ We end when the middle factor equals

$$I_n \otimes F_{n/n} = I_n \otimes F_1 = I_n \otimes I_1 = I_n.$$



Cooley–Tukey theorem (1965)

The Cooley–Tukey decomposition of F_n is

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

where R_n is the bit-reversal permutation matrix,

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



Unordered FFT based on Cooley–Tukey theorem

input: \mathbf{x} : vector of length $n = 2^m$, $m \geq 1$, $\mathbf{x} = \mathbf{x}_0$.

output: $\mathbf{x} = (I_1 \otimes B_n)(I_2 \otimes B_{n/2})(I_4 \otimes B_{n/4}) \cdots (I_{n/2} \otimes B_2)\mathbf{x}_0$.

function UFFT(\mathbf{x} , n)

$k := 2;$

while $k \leq n$ **do**

 { $\mathbf{x} := (I_{n/k} \otimes B_k)\mathbf{x}$ }

for $r := 0$ **to** $\frac{n}{k} - 1$ **do**

 { $x(rk : rk + k - 1) := B_k x(rk : rk + k - 1)$ }

for $j := 0$ **to** $\frac{k}{2} - 1$ **do**

 { Compute $x_{rk+j} \pm \omega_k^j x_{rk+j+k/2}$ }

$\tau := \omega_k^j x_{rk+j+k/2};$

$x_{rk+j+k/2} := x_{rk+j} - \tau;$

$x_{rk+j} := x_{rk+j} + \tau;$

$k := 2k;$



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 k th 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** $\rho_n : \{0, \dots, n-1\} \rightarrow \{0, \dots, n-1\}$ 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



Recursive computation of bit-reversal permutation

- ▶ The bit-reversal permutation ρ_n is related to a bit-reversal permutation of **half its length** by the property

$$\rho_n(j) = \begin{cases} 2\rho_{n/2}(j) & \text{for } 0 \leq j < n/2, \\ 2\rho_{n/2}(j - n/2) + 1 & \text{for } n/2 \leq j < n. \end{cases}$$

- ▶ **Proof** for $j < n/2$: write $j = (0b_{m-2} \cdots b_0)_2 = (b_{m-2} \cdots b_0)_2$. Then

$$\rho_{n/2}(j) = (b_0 \cdots b_{m-2})_2,$$

so that

$$2\rho_{n/2}(j) = (b_0 \cdots b_{m-2}0)_2 = (b_0 \cdots b_{m-2}b_{m-1})_2 = \rho_n(j).$$

- ▶ **Proof** for $n/2 \leq j < n$: similar.



Cost of computing the bit-reversal permutation

- ▶ Define $T(n)$ as the **time in ops** (operations) needed for computing $\rho_n(0: n - 1)$.

- ▶ Then

$$T(n) = T\left(\frac{n}{2}\right) + 2\frac{n}{2},$$

because the computation of ρ_n uses the $n/2$ computed values of $\rho_{n/2}$ and it performs a **multiplication by 2** and an **increment by 1** for each of these values.

- ▶ Repeated application leads to a **linear cost**:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + n = T\left(\frac{n}{4}\right) + \frac{n}{2} + n = \dots \\ &= T(1) + 2 + 4 + \dots + n = 2n - 2. \end{aligned}$$

- ▶ The ops are cheap, as they can be done efficiently using **bit operations**.



Connection between bit-reversal permutation and matrix

- ▶ R_n is called the **bit-reversal matrix**. It is defined by

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

- ▶ Multiplying a vector by R_n starts by splitting the vector into a subvector of components $x_{(b_{m-1} \cdots b_0)_2}$ with $b_0 = 0$ and a subvector with $b_0 = 1$.
- ▶ The most significant bit of the new position of a component is b_0 .
- ▶ Each subvector is then split according to bit b_1 , and so on.
- ▶ The **final position** of the vector component with index $j = (b_{m-1} \cdots b_0)_2$ is $\rho_n(j) = (b_0 \cdots b_{m-1})_2$.
- ▶ Therefore $(R_n \mathbf{x})_{\rho_n(j)} = x_j$.



Theorem 3.11: $R_n = P_{\rho_n}$

- ▶ Lemma 2.5 gives the **connection** between a permutation σ and a permutation matrix P_σ :

$$(P_\sigma \mathbf{x})_j = x_{\sigma^{-1}(j)}.$$

- ▶ We have just seen that $(R_n \mathbf{x})_{\rho_n(j)} = x_j$. Since ρ_n is its own inverse, we can also write this as

$$(R_n \mathbf{x})_j = x_{\rho_n(j)}.$$

- ▶ Comparing the two equations above, we see that $\sigma = \rho_n^{-1} = \rho_n$, and hence we have shown informally that

$$R_n = P_{\rho_n}.$$

- ▶ A more formal proof based on induction is given in the book.



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 $\mathbf{x} = R_n \mathbf{x}_0$.

function BITREV(\mathbf{x} , n)

$\rho_1(0) := 0$; $k := 2$;

while $k \leq n$ **do**

 { Compute $\rho_k(0: k - 1)$ }

for $j := 0$ **to** $k/2 - 1$ **do**

$\rho_k(j) := 2\rho_{k/2}(j)$;

$\rho_k(j + k/2) := 2\rho_{k/2}(j) + 1$;

$k := 2k$;

 { Swap components of \mathbf{x} based on ρ_n }

for $j := 0$ **to** $n - 1$ **do**

if $j < \rho_n(j)$ **then**

 swap ($x_j, x_{\rho_n(j)}$);



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.
- ▶ The total number of flops is $5n \log_2 n$, the same as for the recursive FFT. The bit reversal costs an additional n ops.
- ▶ The nonrecursive variant is a good basis for **parallelization**.

