La transformée de Fourier en algorithmique : discrète et efficace

Jean-Louis Roch

Grenoble-INP,
LIG / INRIA, Grenoble, France

"Once the [FFT] method was established, it became clear that it had a long and interesting prehistory going back as far as Gauss. But until the advent of computing machines it was a solution looking for a problem."

T. W. Körner, Fourier Analysis (1988)

Quelques applications de la transformée de Fourier discrète

"Life as we know it would be very different without the FFT. » *Charles Van Loan*

- Traitement du signal: équivalent logiciel d'un analyseur de spectre
- Son et images numériques: Analyse et filtrage
  - synthèse sonore.
  - déterminer la fréquence d'une note dans une musique enregistrée, identifier des oiseaux, …
  - images numériques
- Applications scientifiques ou statistiques
  - analyse des informations sismographiques,
  - fluctuations dans les prix du marché, les populations animales, analyse des séquences ADN
- Communication numérique :
  - Sécurité: intégrité, confidentialité

- Outil de base en informatique: « Compter efficacement »

« The Top 10 Algorithms of the 20th»

[J. Dongarra, F. Sullivan editors, Computing in Science and Engineering, Jan./Feb. 2000]
[Science page 799, February 4, 2000]

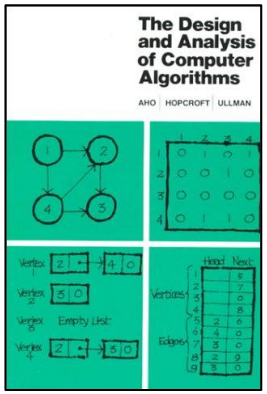
- 1946: The Metropolis Algorithm for Monte Carlo.
- 1947: Simplex Method for Linear Programming.
- 1950: Krylov Subspace Iteration Method.
- 1951: The Decompositional Approach to Matrix Computations.
- 1957: The Fortran Optimizing Compiler.
- 1959: QR Algorithm for Computing Eigenvalues.
- 1962: Quicksort Algorithms for Sorting.

- **1965: Fast Fourier Transform**. ***« An algorithm the whole family can use »***
  - « (…) *the most ubiquitous algorithm in use today to analyze and manipulate digital or discrete data. The FFT takes the operation count for discrete Fourier transform from* $O(N^2)$ *to* $O(N \log N)$. »

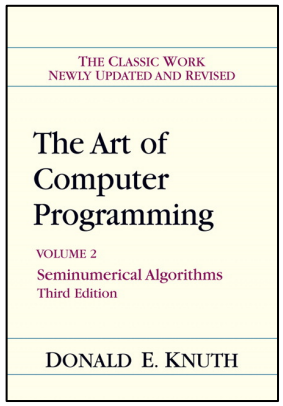
- 1977: Integer Relation Detection.
- 1987: Fast Multipole Method.

La transformée de Fourier en algorithmique

- Algorithmique: maitriser le coût = la complexité
  - espace mémoire, nombre d'opérations, temps parallèle, défauts de cache, surface×temps^2, …

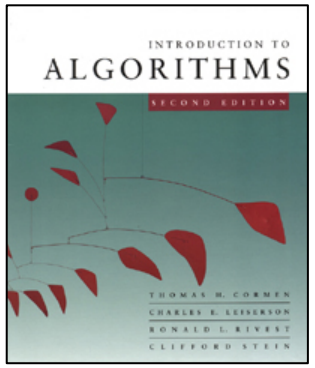


2nd ed. 1974
Chap. 7.
The Fast Fourier Transform
And its Applications



3rd ed. 1997
Section 4.3.3.C:
Discrete Fourier transforms

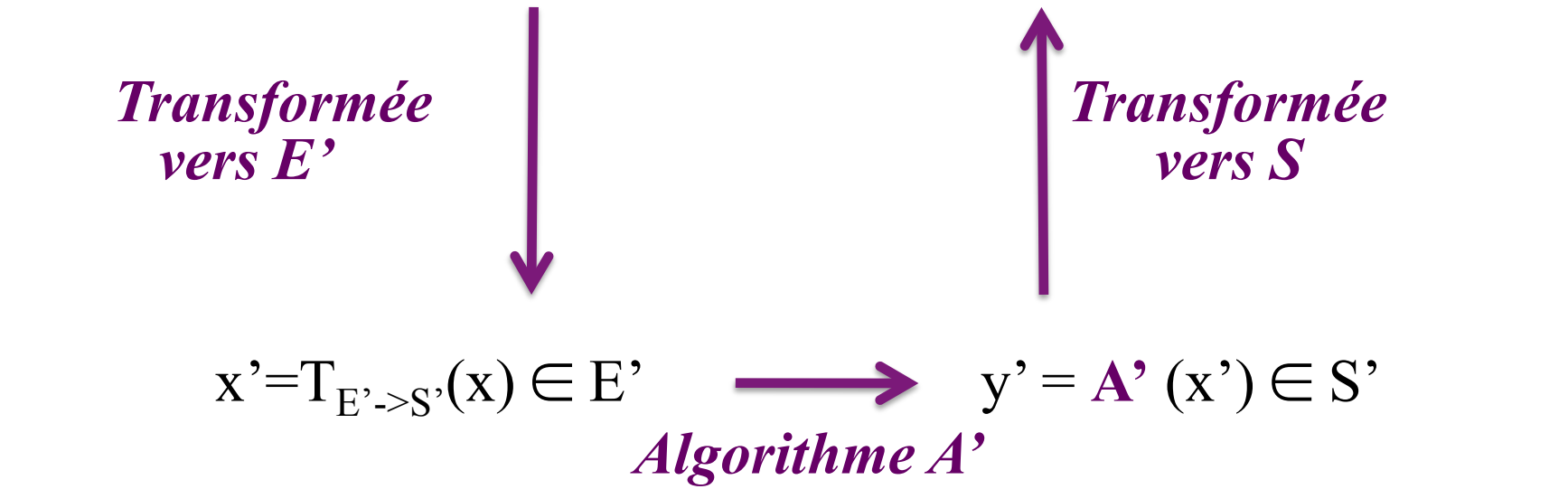
2nd ed. 1992
Chap. 12.
Fast Fourier Transform



2nd ed. 2002
Chap. 30.
Polynomials and the FFT

Conception d'algorithmes

- Pour améliorer le coût, trouver une bonne représentation :



- Objectif : améliorer le coût
  - coûts transformée(x') + cout A'(x') + coût transformée inverse(y')

Exemple: multiplier deux entiers

- **Entrée**: a, un entier **Sortie**: $a^2 = a \times a$
  (la multiplication se réduit au carré: $(a \times b) = [(a + b)^2 - (a - b)^2] / 4$

- **Algorithmes et représentations d'un entier « a »**
  - **unaire**: « $1 \ldots 1$ »$_{[a \text{ fois}]}$
    - Multiplication en O(a) opérations
  - **en base 10** (ou 2^{64}) : « $[a_{n-1} \ldots a_0]$ »
    - Multiplication « classique » en $O(\log^2 a)$ opérations = $O(n^2)$
      ou « diviser pour règner » en $O(\log^{1+\varepsilon} a)$ = $O(n^{1+\varepsilon})$
  - **modulaire**: « $[a \bmod p_1, \ldots, a \bmod p_n]$ = O(n)
    - Multiplication composante par composante en $O(\log a)$ « temps réel »
    - Mais taille limitée; et coûts transformation et comparaison

- Complexité pratique [Logiciel GNU Multiple-Precision GMP, …]
- et aussi théorique :
  - « Meilleur » minorant du nombre d'opérations = $n \cdot \log_2 n$
  - Meilleur majorant connu [Fürer, 2007] = $n \cdot \log_2 n \cdot 2^{O(\log^* a)}$

Plan de l'exposé

I. DFT et FFT

II. Améliorations algorithmiques de la DFT

III. Améliorations algorithmiques reposant sur la DFT

Transformée de Fourier discrète (DFT)

Soit $(A, +, \times, 0, 1)$ un anneau commutatif tel que

- $n = 1+ \ldots + 1$ (n fois) est inversible, d'inverse n^{-1},
- il existe dans A un élément ω qui est une racine n-ième primitive de l'unité:

$$\omega^0 = \omega^n = 1 \quad \text{et} \quad \text{pour } 0 < j < n : \omega^j \neq 1.$$

Définition : Soit $\boldsymbol{u} = [u_0, \ldots, u_{n-1}]$ un vecteur de A^n.

La transformée de Fourier discrète $\hat{\boldsymbol{u}}$ *de* $\boldsymbol{u}$ *par rapport à* ω est

$$\mathbf{DFT}_\omega(u) = \hat{\boldsymbol{u}} = [\hat{u}_0, \ldots, \hat{u}_{n-1}] \text{ avec } \hat{u}_j = \Sigma_{0 \le k < n}\, u_k . \omega^{k.j} .$$

Remarques:

– La valeur de chaque $\hat{u}_k$ dépend des n entrées $u_0, \ldots, u_{n-1}$

– « ***Transformée*** » car $\mathbf{DFT}_\omega$ est inversible :

$$\mathbf{DFT}_\omega(u) = \hat{\boldsymbol{u}} \quad \Leftrightarrow \quad u = n^{-1} . \mathbf{DFT}_{\omega^{-1}}(\hat{\boldsymbol{u}})$$

« *transformée de Fourier inverse* »

DFT, une transformée pour les convolutions

- Définition: convolution $u*v$ de deux vecteurs u et v de A^n

$$(u * v)_j = \Sigma_{0 \le k < j} \; u_k \cdot v_{j-k}$$

- Propriété :

$$\mathrm{DFT}(u * v) = \mathrm{DFT}(u) \cdot \mathrm{DFT}(v)$$

- Coût des algorithmes naïfs de convolution :
  - Domaine temps : n^2 multiplications dans A
  - Domaine fréquences : n multiplications dans A !

DFT et calcul « continu » (flottants)

- Dans $\mathbb{C}$ = corps des nombres complexes: $\omega = e^{-2i\pi/n}$

- Exemple: DFT sur n = 100 points

  Pour j=0 .. n-1

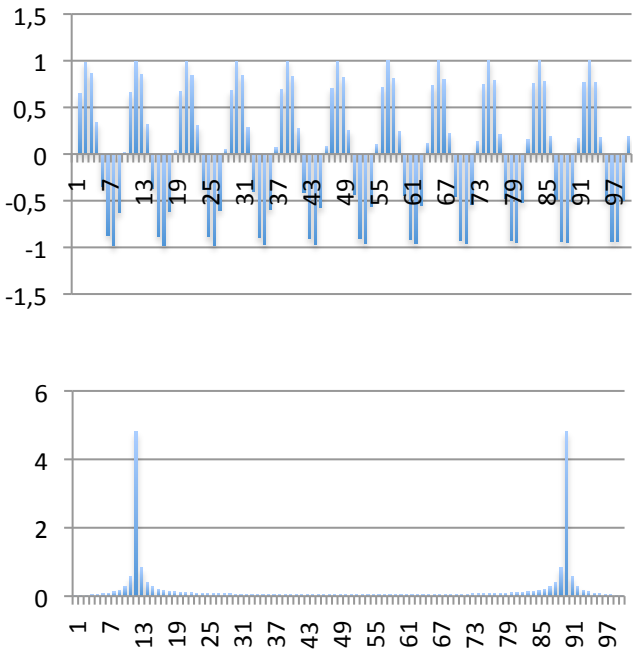
  t = i. 10^{-5} s

  $u_j = \sin(\ 0{,}7\ j)$

  ⇕

  Pour j=0 .. n-1

  $\hat{u}_j = \Sigma_{0 \le k < n}\ u_k . \omega^{k.j}$



- Remarque:
  - transformée de Fourier continue (transformée en cosinus)

$$\mathcal{F}(f) : \nu \mapsto \hat{f}(\nu) = \int_{-\infty}^{+\infty} f(t)\, e^{-i2\pi\nu t}\, dt$$

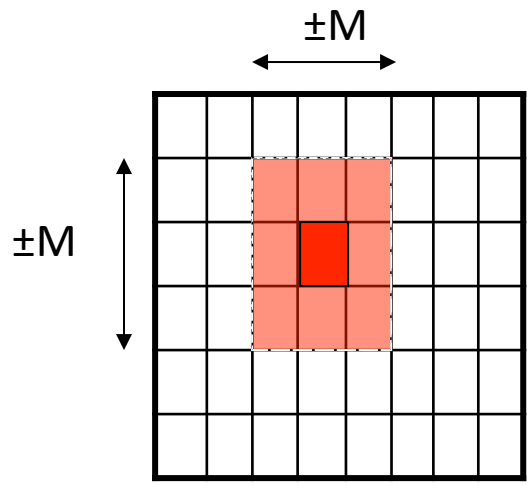
Illustration : Filtrage d'images

- Donnée: image=f(x,y) (x,y)=pixel
- Filtrer ⇔ chaque pixel = somme pondérée de ses voisins

$$g(x,y) = \Sigma_{-M \le i \le M} \Sigma_{-M \le j \le M} \; \boldsymbol{h}(i,j) \, . \, f(x+i, y+j)$$

filtre

- Interprétation géométrique



Coût calcul naïf: $O(M^2)$ par pixel
$\Rightarrow O(M^2N^2)=O(N^4)$ par image NxN

Filtrage d'images par DFT

- On définit la transformée de Fourier bi-dimensionnelle (N=2M) de l'image f :

$$\hat{f}(u,v)=DFT(f)(u,v)=\Sigma_{-M\leq k\leq M}\Sigma_{-M\leq l\leq M}\ f(k,l)\ .\ e^{-2i.\pi/n\ (\ k.u\ +\ l.v)}$$

- Idem pour le filtre : $\hat{h}$ = DFT (h) et l'image filtrée $\hat{g}$=DFT(g). On a alors :

$$\hat{g}(u,v) = \hat{h}(u,v).\hat{f}(u,v)$$

Filtre stocké

- Algorithme
  - 1. $\hat{f}$:= DFT(f)
  - 2. $\hat{g}(u,v) := \hat{h}(u,v).\hat{f}(u,v)$ pour tout u,v : $O(N^2)$
  - 3. g := $DFT^{-1}(\hat{g})$

Filtrage d'images par DFT

- On définit la transformée de Fourier bi-dimensionnelle (N=2M) de l'image f :

$$\hat{f}(u,v)=DFT(f)(u,v)=\Sigma_{-M\le k\le M}\,\Sigma_{-M\le l\le M}\; f(k,l)\,.\,e^{-2i.\pi/n\,(\,k.u\,+\,l.v)}$$

- Idem pour le filtre : $\hat{h}$ = DFT (h) et l'image filtrée $\hat{g}$=DFT(g). On a alors :

$$\hat{g}(u,v) = \hat{h}(u,v).\hat{f}(u,v)$$

Filtre stocké

- Algorithme
  - 1. $\hat{f}$:= DFT(f) **$O(N^2.\log N)$**
  - 2. $\hat{g}(u,v) := \hat{h}(u,v).\hat{f}(u,v)$ pour tout u,v : $O(N^2)$
  - 3. g := $DFT^{-1}(\hat{g})$ **$O(N^2.\log N)$**

Illustration

Image initiale

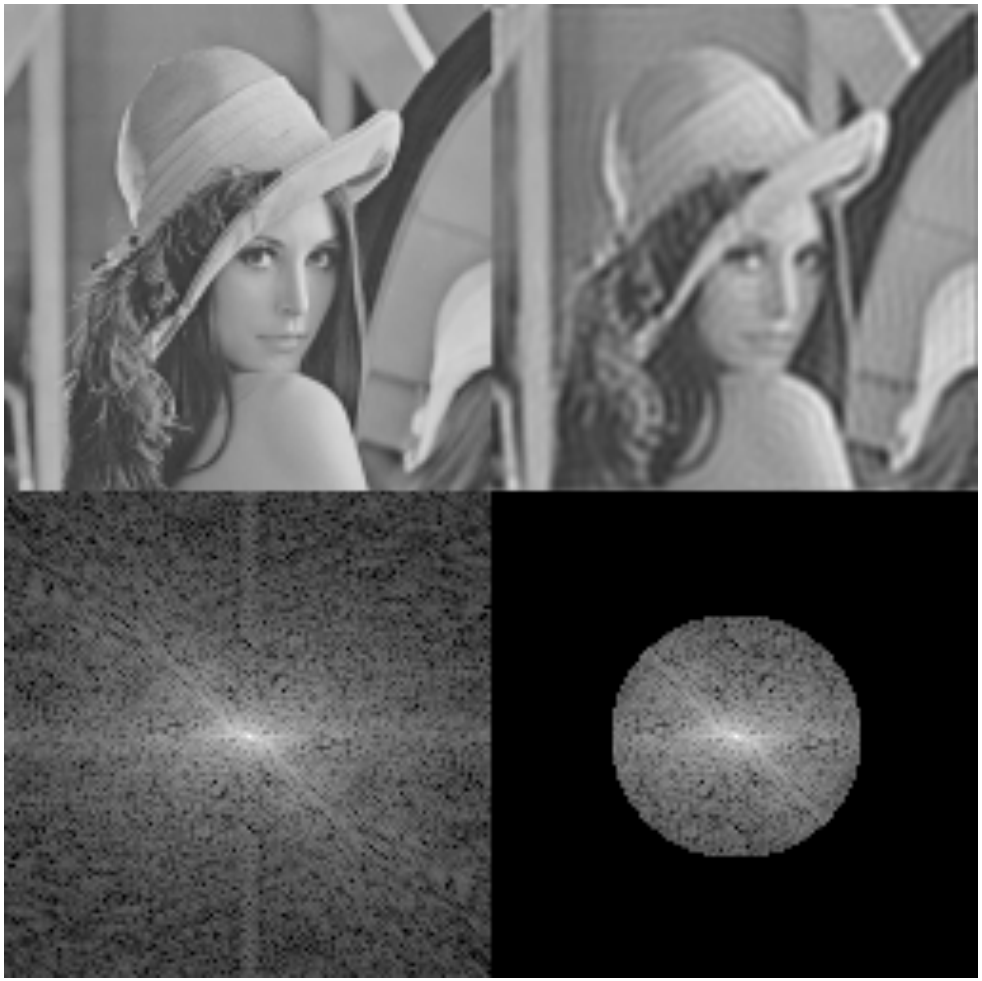


Image filtrée

DFT de l'image

Application du filtre en fréquences

Algorithme de la DFT et réorganisation des calculs

Algorithme FFT « Radix 2 » [Cooley-Tuckey 1965]

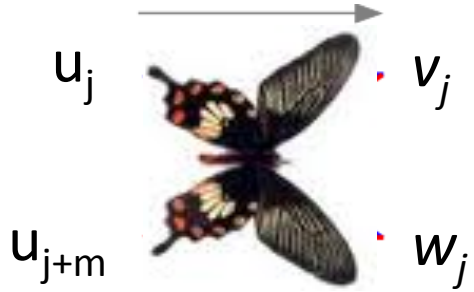
- On pose $n = 2.m$: $\omega^m = -1$;
  - $\hat{u}_{2k} = \Sigma_{0 \le j < m-1} (u_j + u_{j+m}) . \omega^{2.k..j}$
  - $\hat{u}_{2k+1} = \Sigma_{0 \le j < m-1} (u_j - u_{j+m}) . \omega^j . \omega^{2.k..j}$

- D'où, en posant :
  - $\theta = \omega^2$ est une racine $m^{ième}$ de l'unité
  - $v_j = (u_j + u_{j+m}) \Rightarrow \hat{u}_{2k} = \Sigma_{0 \le j < m-1} v_j . \theta^{k..j}$
  - $w_j = (u_j - u_{j+m}) . \omega^j \Rightarrow \hat{u}_{2k+1} = \Sigma_{0 \le j < m-1} w_j . \theta^{k..j}$

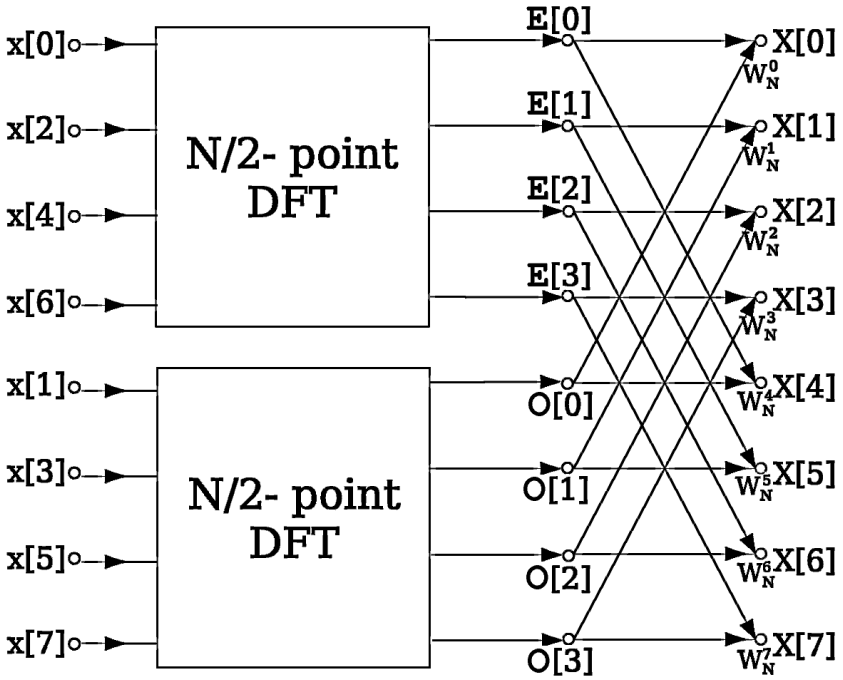
« **Twiddle factor** »

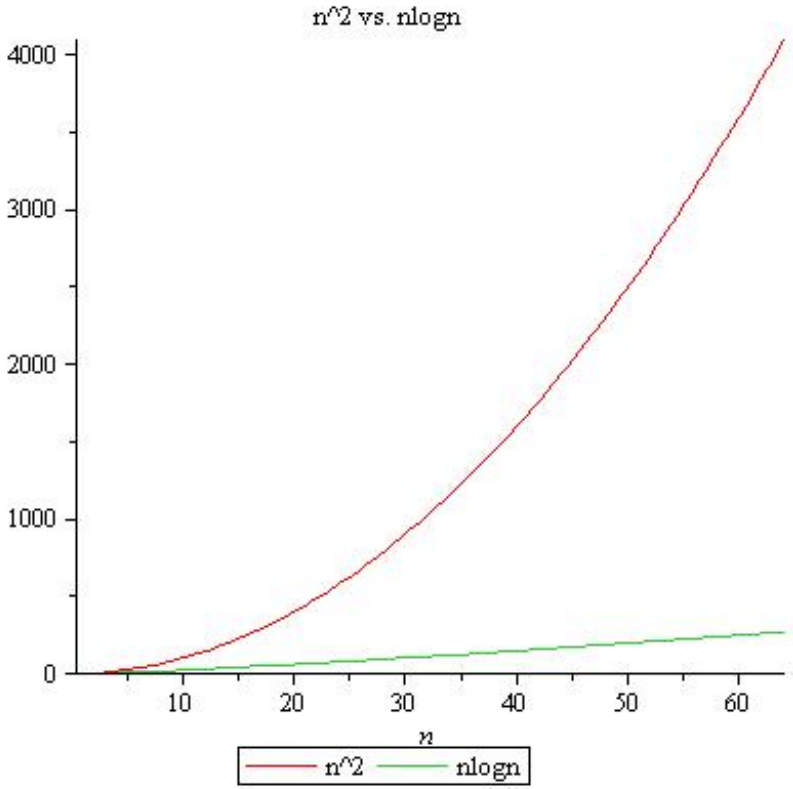
FFT de taille m=n/2

FFT de taille m=n/2



Algorithme FFT « Radix 2 » [Cooley-Tuckey 1965]





- $\text{Coût}_{FFT}(n) = 2.\ \text{Coût}_{FFT}(n/2) + \Theta(n) = \Theta(\ \boldsymbol{n.\log_2 n}\ )$ opérations.

between the publication of Gauss' algorithm and the modern rediscovery of this approach by Cooley & Tukey.

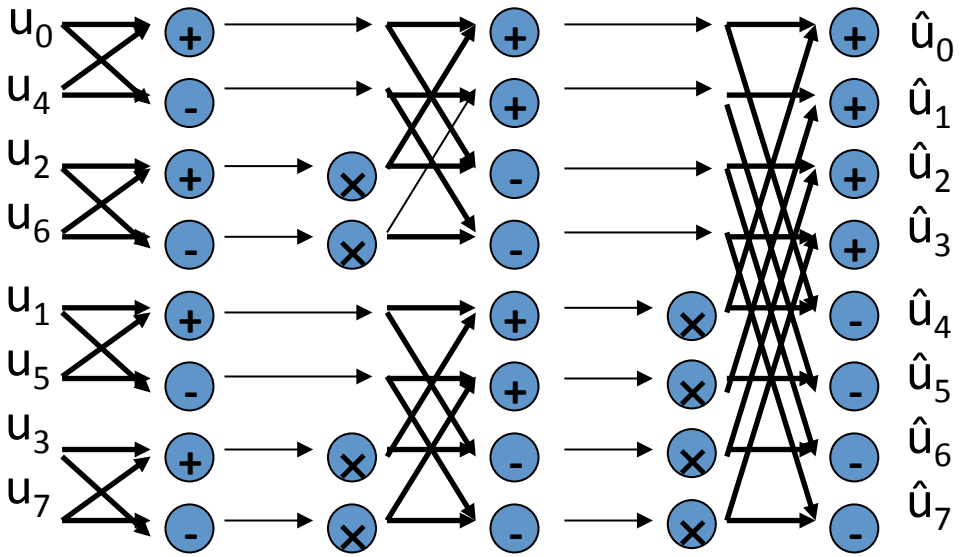
Principal Discoveries of Efficient Methods of Computing the DFT

Researcher(s)	Date	Lengths of Sequence	Number of DFT Values	Application
C. F. Gauss [10]	1805	Any composite integer	All	Interpolation of orbits of celestial bodies
F. Carlini [28]	1828	12	7	Harmonic analysis of barometric pressure variations
A. Smith [25]	1846	4, 8, 16, 32	5 or 9	Correcting deviations in compasses on ships
J. D. Everett [23]	1860	12	5	Modeling underground temperature deviations
C. Runge [7]	1903	2^nK	All	Harmonic analysis of functions
K. Stumpff [16]	1939	2^nK, 3^nK	All	Harmonic analysis of functions
Danielson & Lanczos [5]	1942	2^n	All	X-ray diffraction in crystals
L. H. Thomas [13]	1948	Any integer with relatively prime factors	All	Harmonic analysis of functions
I. J. Good [3]	1958	Any integer with relatively prime factors	All	Harmonic analysis of functions
Cooley & Tukey [1]	1965	Any composite integer	All	Harmonic analysis of functions
S. Winograd [14]	1976	Any integer with relatively prime factors	All	Use of complexity theory for harmonic analysis

M. T. Heideman, D. H. Johnson, C. S. Burrus, *Gauss and the history of the fast Fourier transform*, *IEEE ASSP Magazine 1 (4), 14–21 (1984).*

Programmation (radix 2) : schéma papillon

- Calcul en place, en colonne : « butterfly »



- Parallélisme potentiel:
  - $\text{Log}_2\, n$ étapes, n opérations indépendantes à chaque étape

- Implémentations matérielles

Stabilité numérique

- La FFT est peu sensible aux erreurs d'arrondi :
  - Si $\hat{u}_f$ est le résultat du calcul flottant avec précision υ :

$$\frac{\| \hat{u} - \hat{u}_f \|_2}{\| \hat{u} \|_2} \leq \sqrt{n} . \log n . \; c\upsilon$$ avec $c\upsilon \sim 10^{-15}$ en double précision

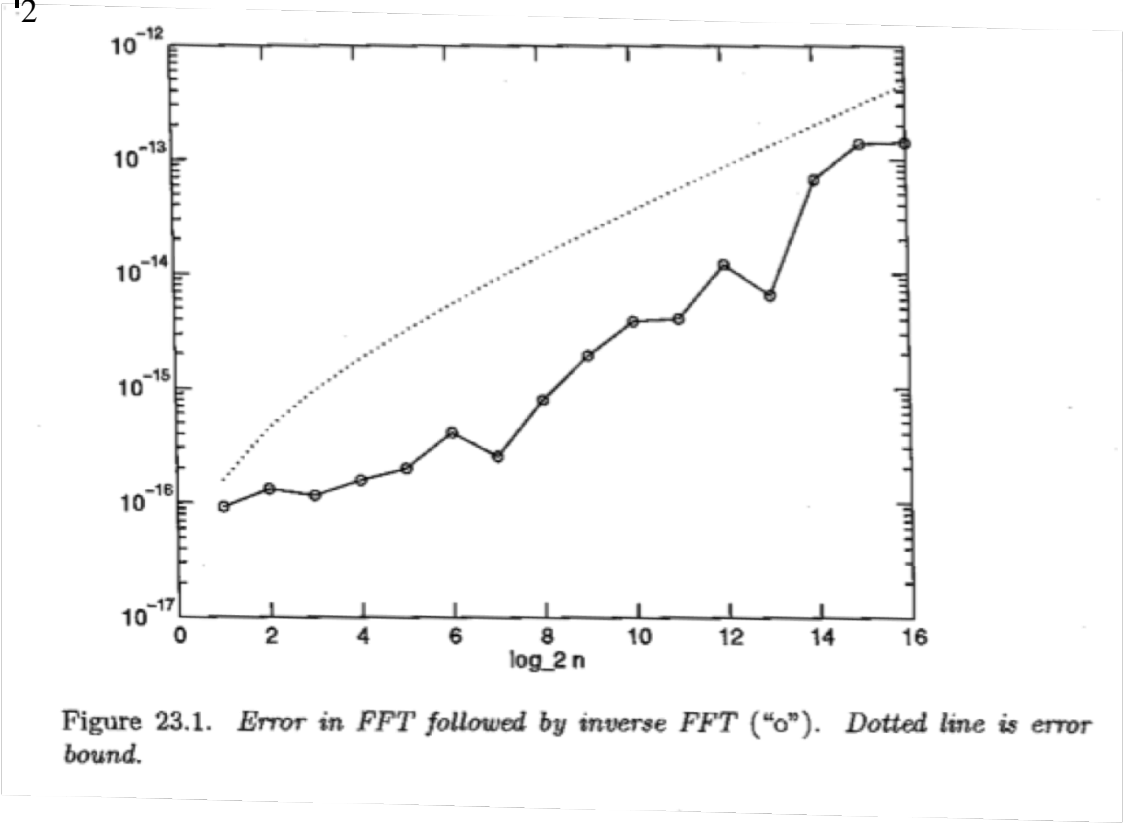


Figure 23.1. *Error in FFT followed by inverse FFT* ("o"). *Dotted line is error bound.*

[Higham 2002]

DFT: Interprétation matricielle

- La transformée de Fourier (directe ou inverse) est un produit matrice-vecteur (matrice de Vandermonde) :

$$\mathbf{DFT}_\omega(u) = \hat{u} = [n^{-1}.]\ \boldsymbol{\Omega.u}$$

$$\begin{pmatrix} \hat{u}_0 \\ . \\ . \\ . \\ \hat{u}_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & & \omega^{n-1} \\ . & \omega^2 & \omega^4 & & . \\ . & & & \dots & . \\ . & & \omega^{k.j} & & . \\ . & & & & . \\ 1 & \omega^{n-1} & & \dots & \omega^{(n-1)^2} \end{pmatrix} \begin{pmatrix} u_0 \\ . \\ . \\ . \\ u_{n-1} \end{pmatrix}$$

= **évaluation du polynôme** $\Sigma_{0\le k<n} u_k.X^k$ **en** $1, \omega, \omega^2, \ \omega^{n-1}$

- La valeur de chaque $\hat{u}_k$ dépend des n entrées $u_0, \dots, u_{n-1}$

DFT et polynômes (si ω et n^{-1})

- DFT_ω (coefficients d'un polynôme de degré < n) :
  - évaluation du polynôme aux n abscisses ω^i.
  - $DFT_\omega^{-1} = n^{-1}.DFT_{\omega^{-1}}$: Interpolation du polynôme

- Multiplication de polynômes de degrés $m<n/2$ en $O(n.\log n)$ opérations, asymptotiquement optimal

  - Entrée: $A= \Sigma_{0\le i<m}\ a_i X^i$ et $B=\Sigma_{0\le i<m}\ b_i X^i$
  - Sortie: $R = A.B = \Sigma_{0\le i<2m}\ r_i X^i$

    1. Calcul $A' := DFT_\omega([0, \ldots,0, a_{m-1}, \ldots, a_0])$
       et $B' := DFT_\omega([0, \ldots,0, b_{m-1}, \ldots, b_0])$

    2. Pour $0 \le k < n$: $R'_k := A'_k \times B'_k$

    3. Calcul $[r_{n-1}, \ldots, r_0] := n^{-1}.DFT_{\omega^{-1}}(R')$

DFT et arithmétique polynomiale

- Sur un anneau où n est inversible et avec une racine $n^{\text{ième}}$ primitive de l'unité,

- Se ramènent à des multiplications :
  - Division euclidienne : $O(n \log n)$
  - PGCD : $O(n \log^2 n)$
    - Sous-résultants (coefficients de Bezout)

  - Arithmétique $(+, -, \times, ^{-1})$ dans les corps finis

Vision de la DFT comme Evaluation de polynômes aux ω^i

- Algorithme Diviser pour régner (*n* puissance de 2)
  - Le polynôme $P = u_0 + u_1.X^1 + u_2.X^2 + \ldots + u_{n-1}.X^{n-1}$ est divisé en deux polynômes:
    - Coefficients pairs: $P_0 = u_0 + u_2.X^1 + u_4.X^2 + \ldots + u_{n/2-1}.X^{n/2-1}$
    - Coefficients impairs: $P_1 = u_1 + u_3.X^1 + u_5.X^2 + \ldots + u_{n/2-2}.X^{n/2-1}$
  - Alors $P(X) = P_0(X^2) + X.P_1(X^2)$

- Or $(\omega^{n/2+i})^2 = (\omega^i)^2$: il suffit d'évaluer P_0 et P_1 en n/2 valeurs :
  - 1 évaluation (*n*) = 2 évaluations(n/2) + O(n)
  - mêmes opérations que FFT Radix-2.

Evaluation de polynômes et *mixed-radix*

- La formulation $P(X) = P_0(X^2) + X.P_1(X^2)$
  s'étend avec découpe en k (pour $n \leq k.m$).
-

  Soit $P_i = u_i + u_{i+k}.X^1 + u_{i+2k}.X^2 + \ldots + u_{i+(m-1).k}.X^{m-1}$ $(i=0..k-1)$

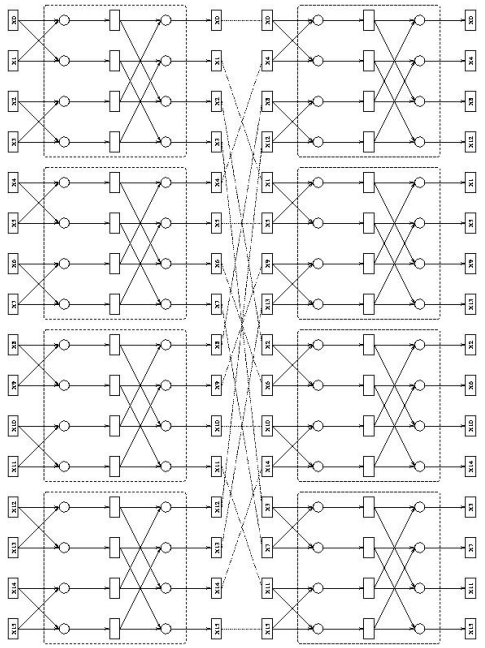
  Alors $P(X) = P_0(X^k) + X.P_1(X^k) + \ldots + X.P_{k-1}(X^k)$

- D'où réduction FFT de taille ***k.m*** à :

  k FFT de taille ***m***

  \+ transposition ***m*** × ***k*** → ***k*** × ***m***

  \+ ***m*** FFT de taille ***k***

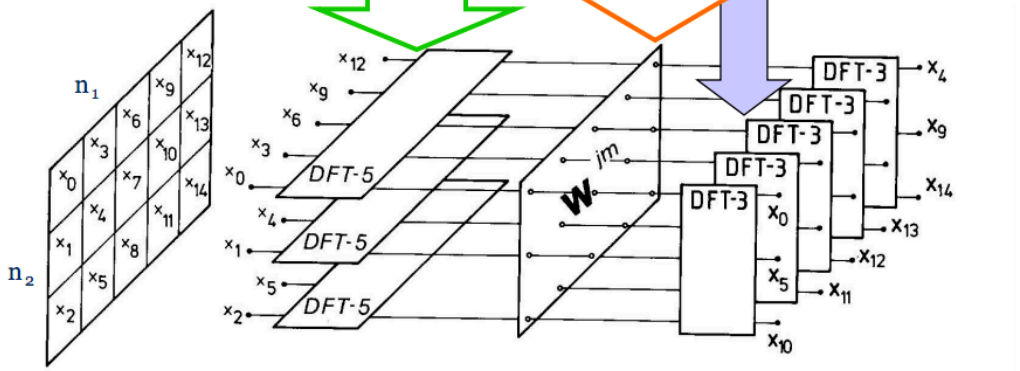


Découpes récursives de DFT(n)

- Algorithmes de Cooley-Tuckey avec $n = n_1 \cdot n_2$
  - Calculer n_2 DFT de taille n_1
  - Multiplier par les racines de l'unité (*twiddle factors*)
  - Calculer n_1 DFT de taille n_2

$$Y[i] = \sum_{j=0}^{n-1} X[j] w^{-ij}$$

$$Y[i_1 + i_2 n_1] = \sum_{j_2=0}^{n_2-1} \left[\left(\sum_{j_1=0}^{n_1-1} X[j_1 n_2 + j_2] w_{n_1}^{-i_1 j_1} \right) w_n^{-i_1 j_2} \right] w_{n_2}^{-i_2 j_2}$$



Formulation générique de la DFT (et DSPs)

$$x \mapsto y = M \cdot x$$

Vecteur entrée (signal)

Vecteur sortie(signal)

transformation = matrice

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

DFT = diagonale (*twiddles*) entrée

$$y = (\mathbf{DFT}_2 \otimes \mathrm{I}_2) \cdot \mathrm{T}^4_2 \cdot (\mathrm{I}_2 \otimes \mathbf{DFT}_2) \cdot \mathrm{L}^4_2 \, x$$

Sortie — Produit Kronecker — Identité — Permutation

Plan de l'exposé

I. DFT et FFT

II. Améliorations algorithmiques de la DFT

Hiérarchie mémoire, Cache
FFTW, SPIRAL

III. Améliorations algorithmiques reposant sur la DFT

Arithmétique exacte

Programmation et performances

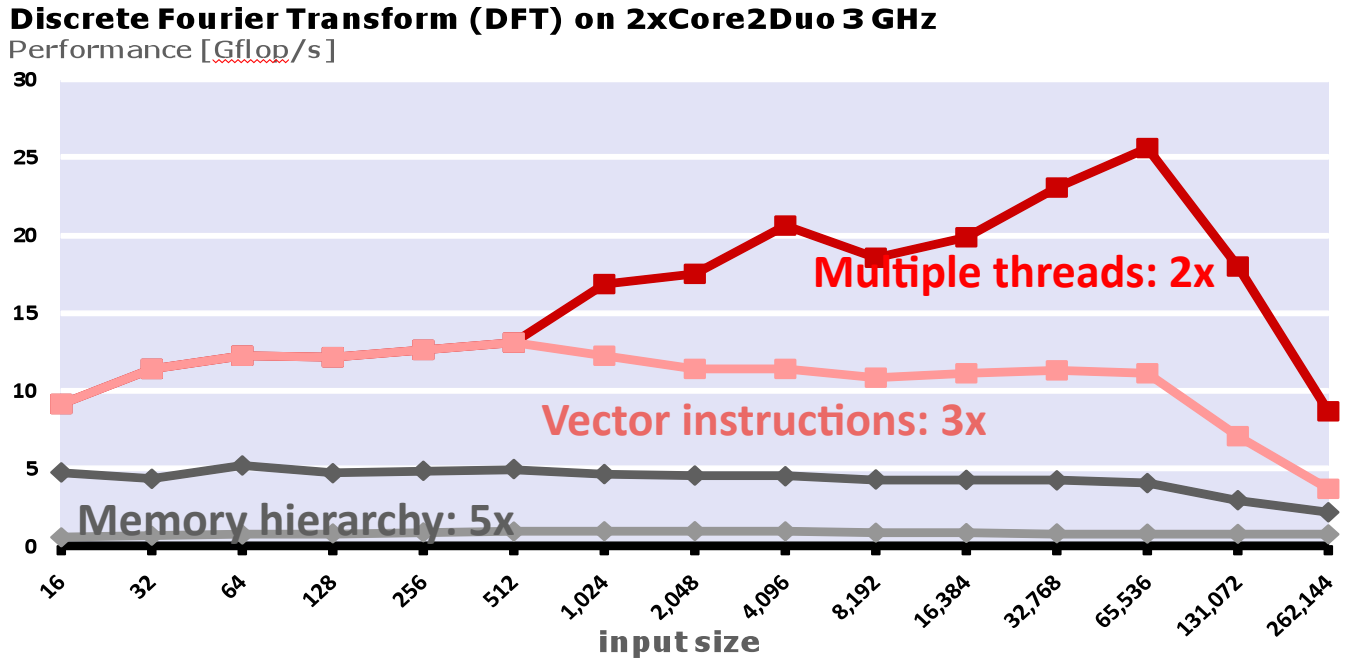
- Premiers programmes « optimisés » :
  - Dérouler la récursivité => programme itératif
  - arrêt sur seuil (FFT spécialisée pour tailles spécifiques)

- Encore et toujours des « meilleurs » algorithmes
  Par exemple, pour FFT réelle de taille 2^m
    - [Yavné, 1968] #opérations FFT « split-radix » : $4\,n\log_2 n - 6n + 8$
    - [Johnson,Frigo, 2007] : $(34/9)\,n\log_2 n - (124/27).n + \ldots$

- Les machines évoluent, les algorithmes aussi.

Programmation Haute-performance

- Algorithmes
- Optimisations de compilation
- Exploiter l'architecture
  - Parallélisme au niveau instruction (ILP)
  - Vectorisation (ex. SSE)
  - Parallélisme (multicoeurs, GPU)
  - Hiérarchie mémoire
  - Mémoire distribuée (clusters, Top'500)

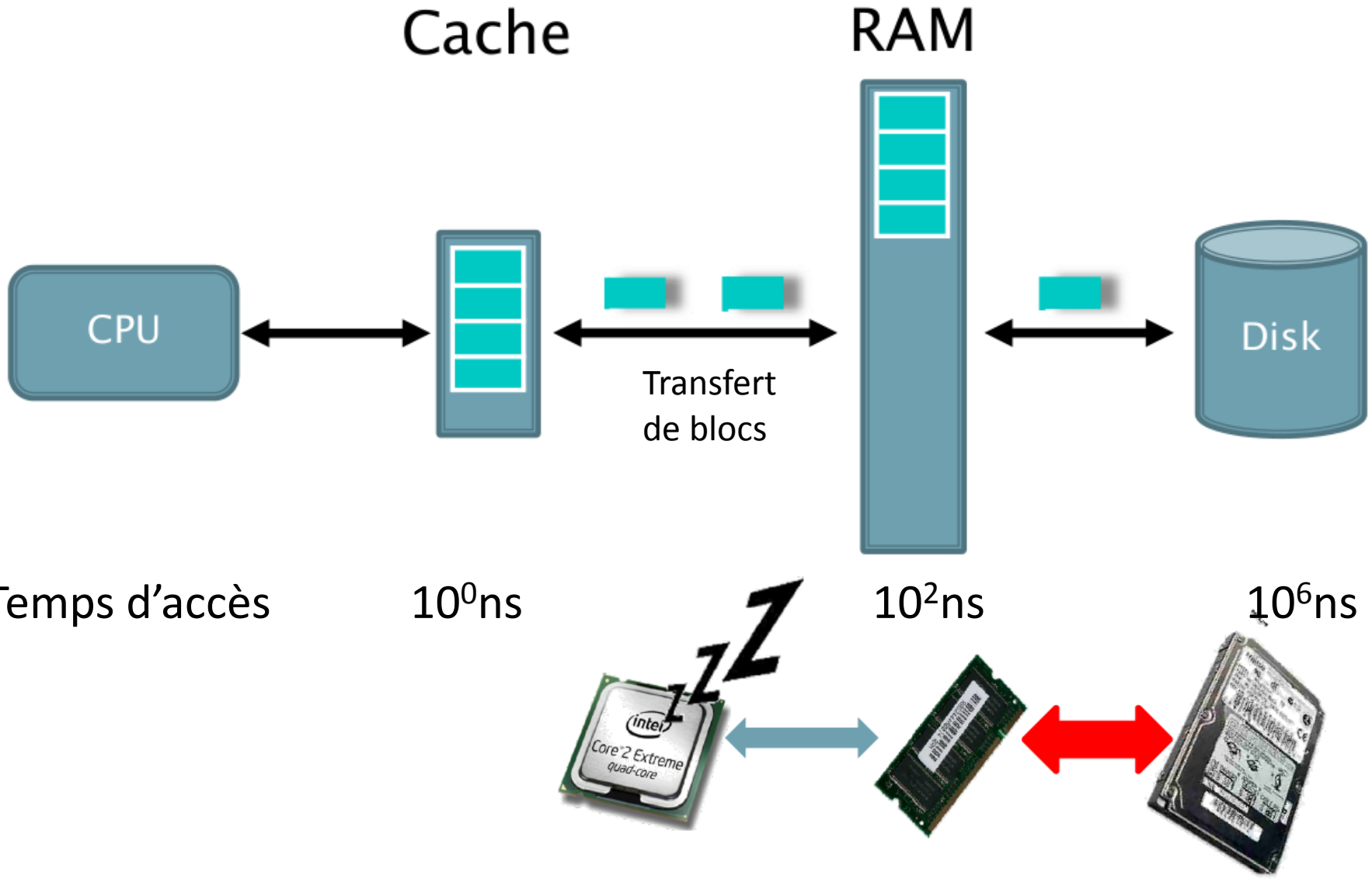


[J. Johnson, SPIRAL]

La programmation devient un cauchemar…

Cela motive de synthétiser les programmes via
- de nouveaux modèles algorithmiques,
- de nouveaux langages de plus haut niveau pour les compiler.

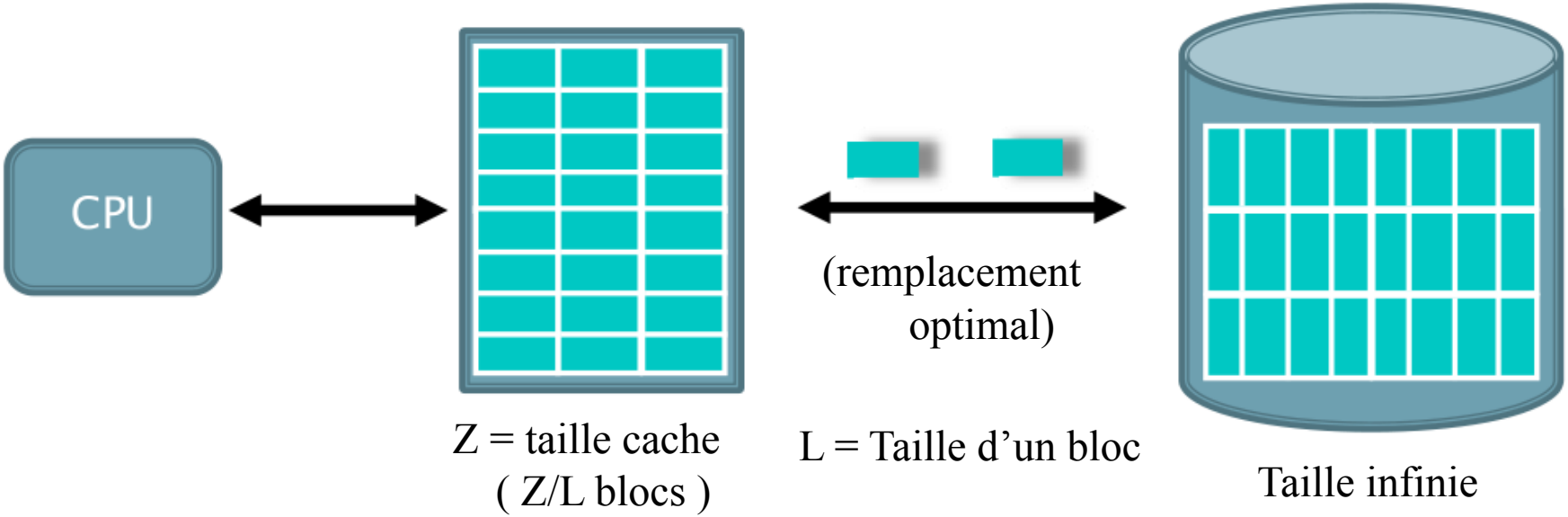
Hiérarchie mémoire et cache



[Aggarwal & Vitter 1988]
[Frigo & al 1999]

Modèle de cache

or external memory
out-of-core
disk access machine
I/O model



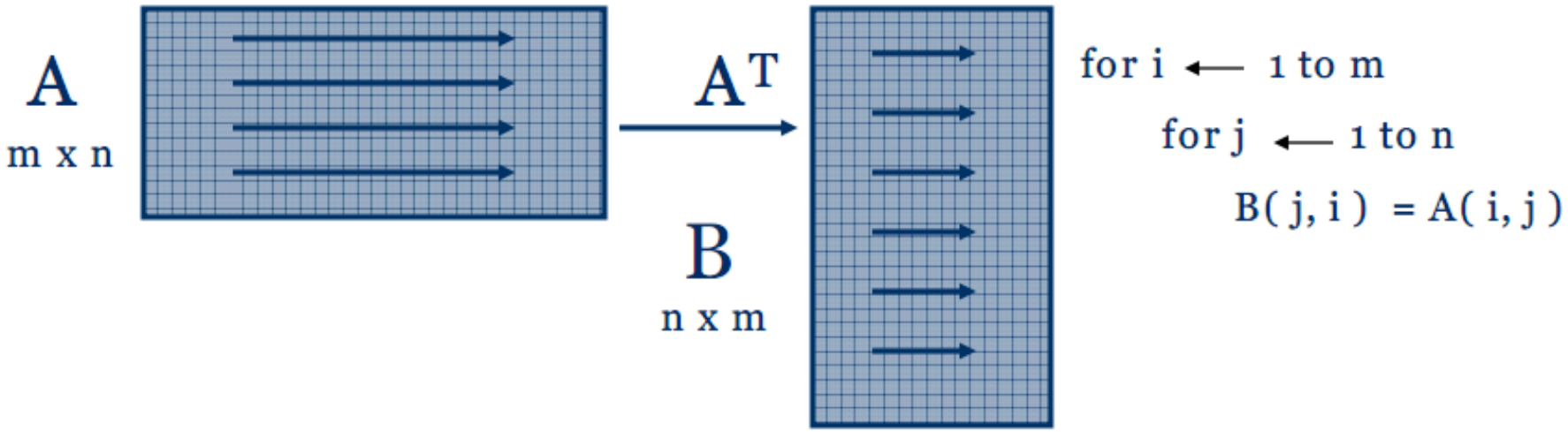
Travail (W) = #opérations

Défauts de cache: Q(n, L, Z) = #transferts de blocs

Transposition de matrice

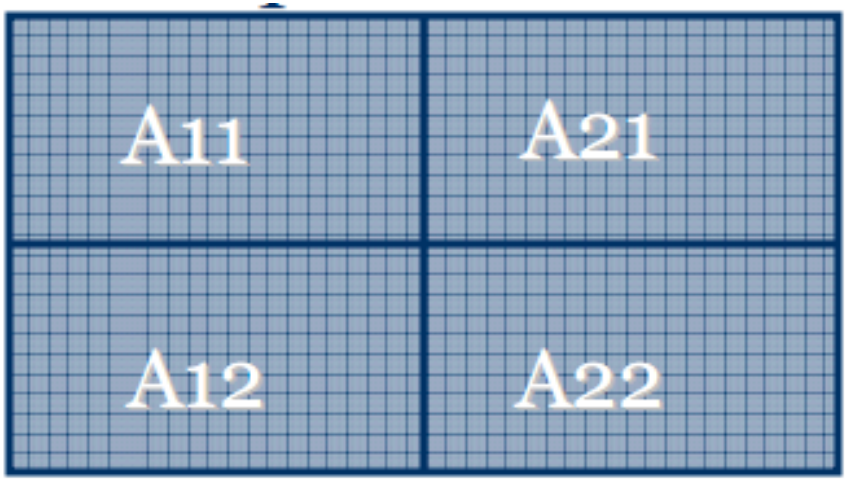
- Le disque = séquence de mots
  - Matrices stockées par lignes

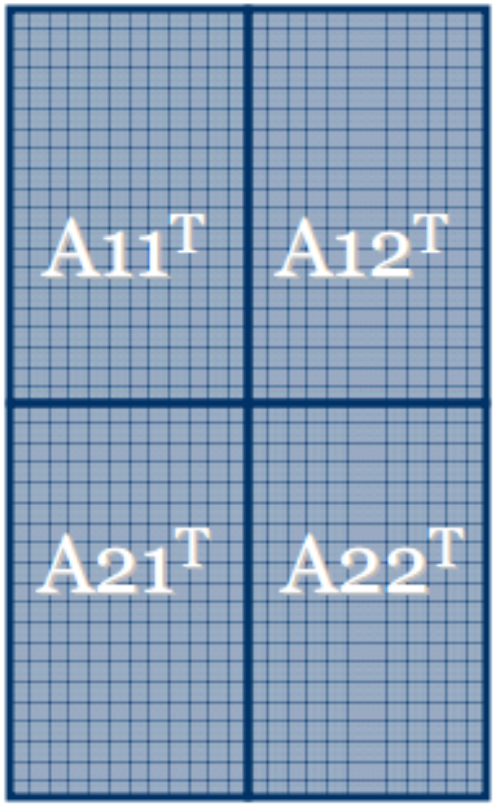
- Si n très large, l'accès de B par colonne cause un défaut de cache à chaque top



Transposition de matrice optimale

- Partitionner A selon la plus grande dimension, et récursivement transposer chaque bloc





- Q(m,n) O(1 + m.n/L) : optimal.
- « **Cache-oblivious** »: optimal sans référencer Z et L dans le programme.

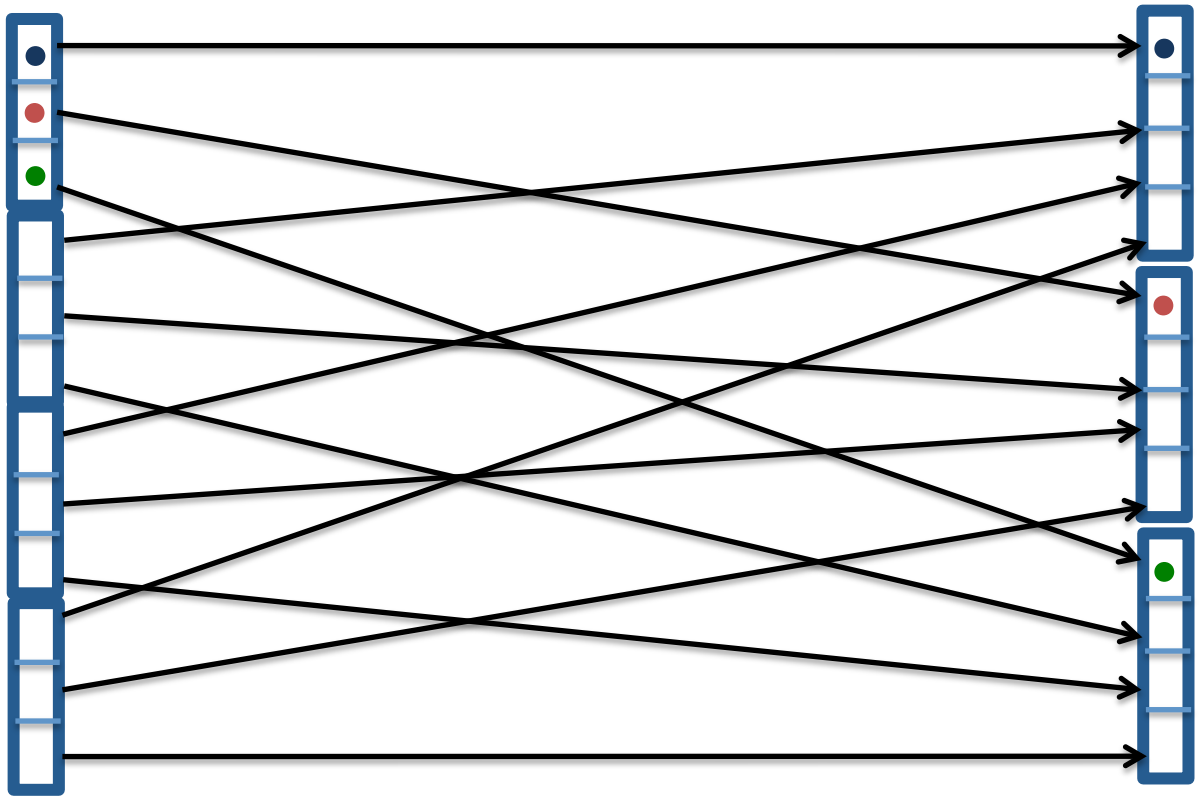
Lien transposition et FFT

- FFT (n.m) = n FFT(m) + transposition + m FFT(n)

matrice n×m twiddle factors matrice m×n

n = 4

m = 3

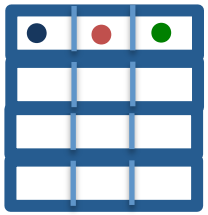


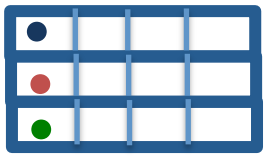
Lien transposition et FFT

- FFT (n.m) = n FFT(m) + transposition + m FFT(n)
  matrice n×m twiddle factors matrice m×n

n = 4

m = 3





FFT et cache

- Cooley-Tuckey :
  - Si programmation naïve en place,
    itératif, colonne par colonne :

    après $\log_2 Z$, plus de localité
  - $Q(n) = n \log n / Z\ L + O(n \log n)$

FFT en connaissant la taille du cache

- Découpe de n en blocs de taille Z maximisant la localité :
  - $n = Z \cdot Z \ldots Z = Z^{\log_Z n}$
- D'où $(n/Z).\log_Z n$ blocs DFT de taille Z
  - Entre deux blocs: permutation avec vidage du cache
- $Q(n) = O(n/L \cdot \log_Z L)$
  - Optimal pour un algorithme de Cooley-Tukey

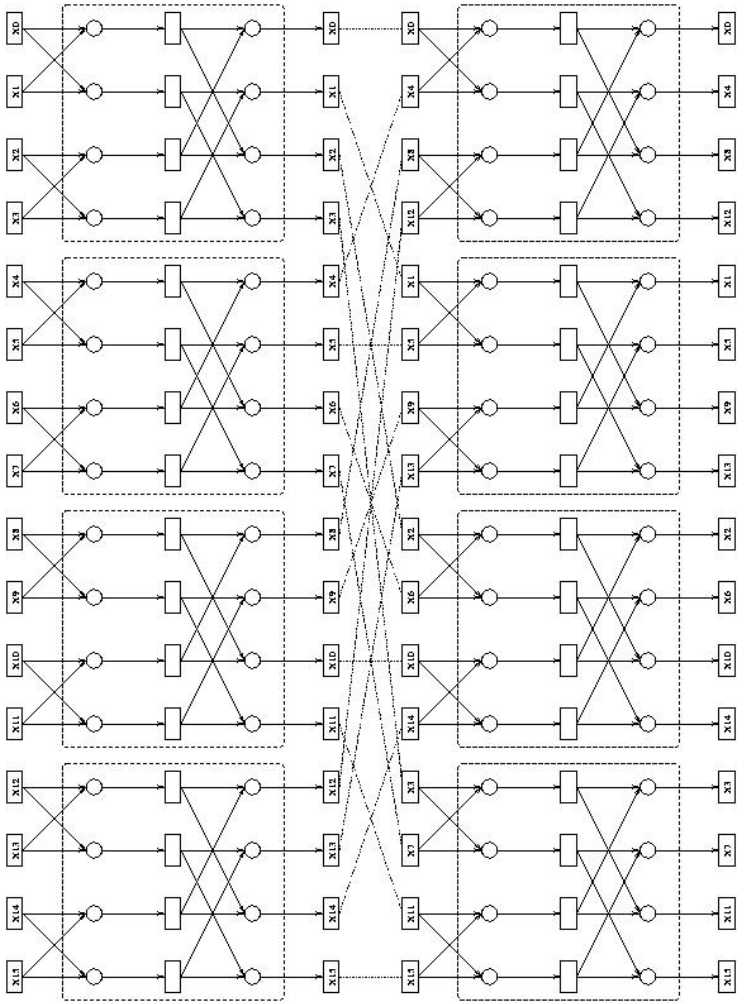
FFT « cache-oblivious »

- Avec une découpe récursive de taille √n :
  1. Calcul de √n FFTde taille √n
  2. Permutation par blocs : $V_i[j] \leftrightarrow V_j[i]$
  3. Calcule √n FFTde taille √n

- si n>Z: $Q(n) = 2\sqrt{n}.Q(\sqrt{n}) + O(n/L)$
  sinon $Q(n) = n$.

- $Q(n) = O(n/L \ . \log_Z L)$: optimal

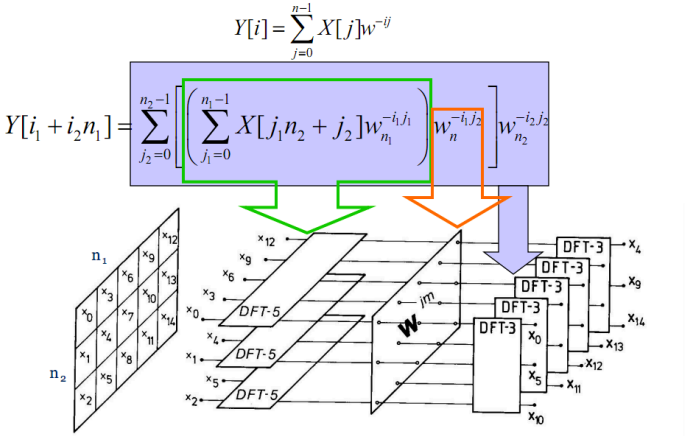
- Cache+parallélisme :
  algorithmique portable



Exemple: n= 16

FFT et réorganisations algorithmiques

- Pour une taille n fixée, choisir la meilleure organisation
  radix $n \leq n_1 \cdot n_2 \geq n$ et parenthésage



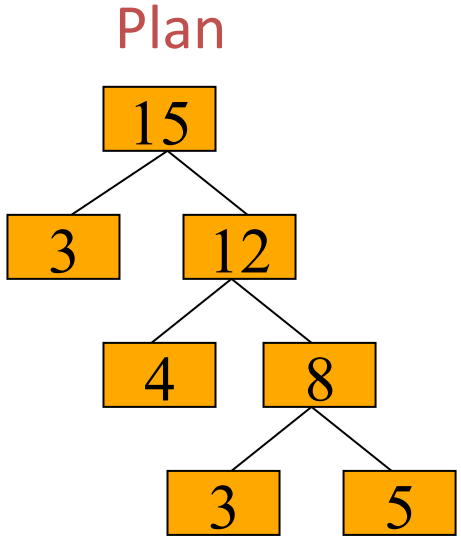
$$y = (\mathbf{DFT}_2 \otimes \mathrm{I}_2) \cdot \mathrm{T}_2^4 \cdot (\mathrm{I}_2 \otimes \mathbf{DFT}_2) \cdot \mathrm{L}_2^4 \; x$$

- Choix à chaque niveau de la récursivité :
  - Limiter la combinatoire
    - programmation dynamique (heuristique ici)
- Exemples d'infrastructures : FFTW et SPIRAL

La bibliothèque FFTW [Frigo, Johnson IEEE05]

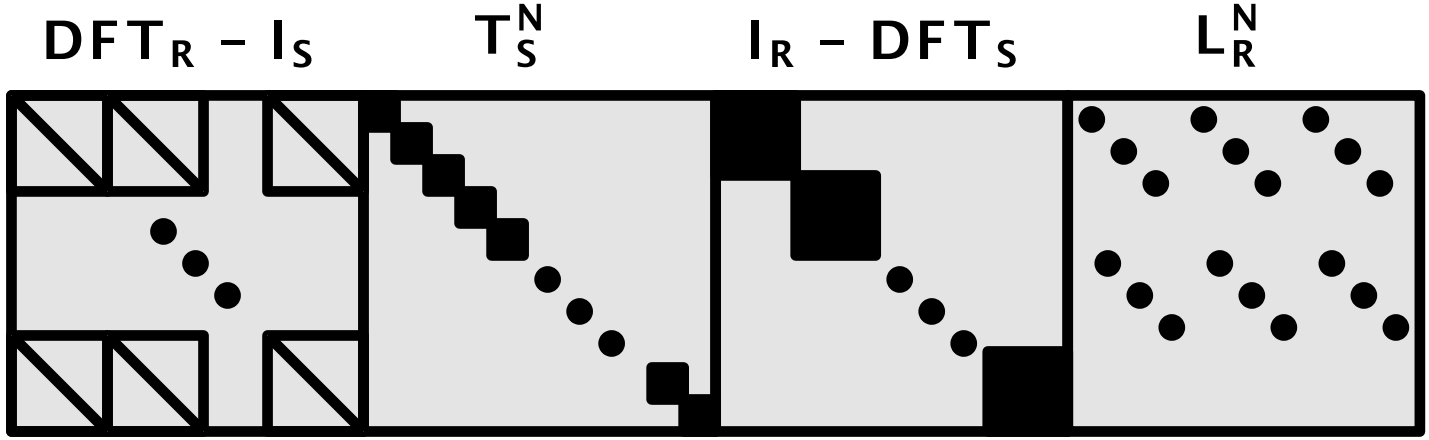
http://www.fftw.org/ [3.2.2])

- Utilise des “codelets”: code optimisé pour les FFTs de petite taille.
- Pour un *n* fixé (paramètre), combine ces codelets via par découpe récursive de la FFT (Split-radix)
  - Différentes stratégies, in-out, ...
- Utilise la programmation dynamique pour trouver une combinaison efficace
- Le résultat du calcul (choix des facteurs de découpe et facteurs “twiddle”) sont stockés dans un plan.
- L’exécution du plan est réalisée par un exécuteur.

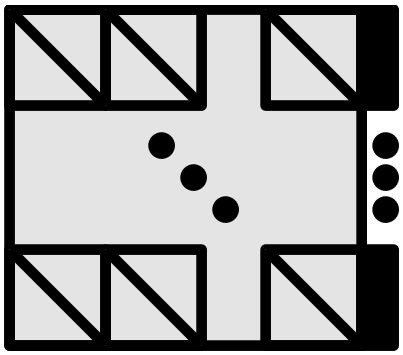


FFTW : optimisations

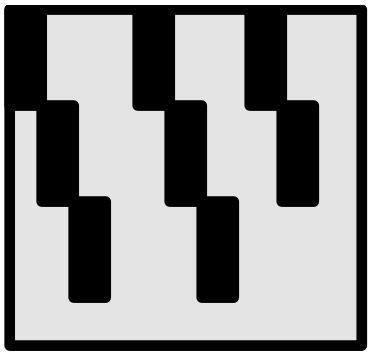
$$DFT_N = (DFT_R - I_S)T^N_S(I_R - DFT_S)T^N_S; \quad N = RS$$



$(DFT_R - I_S)T^N_S$



$(I_R - DFT_S)L^N_R$



FFTW : utilisation

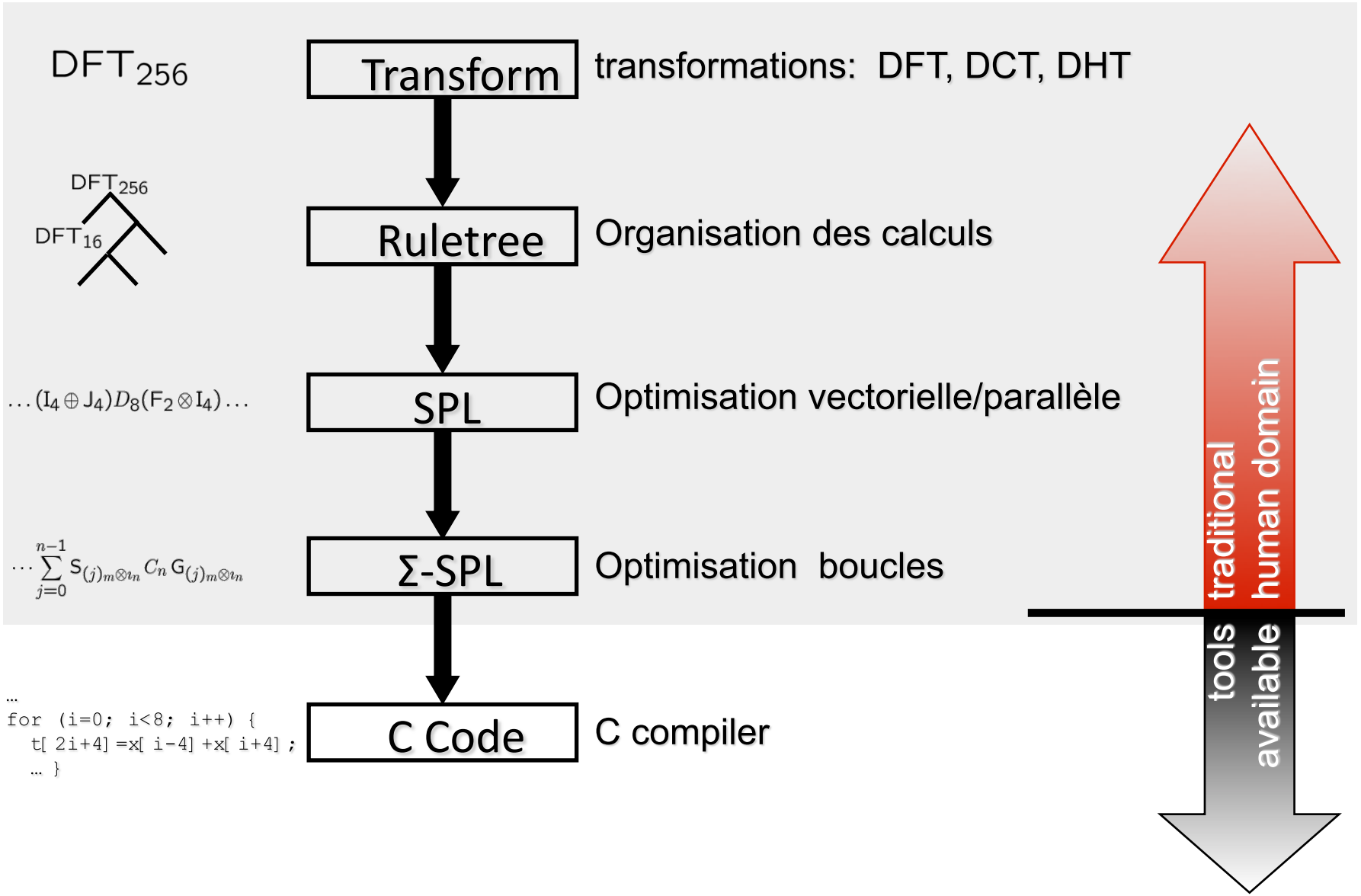
```
fftw_plan plan;
int n = 1024;
COMPLEX A[n], B[n];

/* plan the computation */
plan = fftw_create_plan(n);

/* execute the plan */
fftw(plan, A);

/* the plan can be reused */
fftw(plan, B);
```

Surcoût du calcul du plan amorti par sa réutilisation.

[Johnson&al. PLDI 05]

Synthèse par compilation: SPIRAL

www.spiral.net



SPIRAL: Génération de programme

Choix de la Transformation *(utilisateur)*

$$\mathrm{DFT}_8$$

Algorithme en SPL *choix possibles (règles réécriture)*

$$(\mathrm{DFT}_2 \otimes I_4)\, T_4^8\, (I_2 \otimes ((\mathrm{DFT}_2 \otimes I_2) \cdot T_2^4\, (I_2 \otimes \mathrm{DFT}_2)\, L_2^4))\, L_2^8$$

$\sum$-SPL: *[PLDI 05]*

$$\sum \left(S_j\, \mathrm{DFT}_2\, G_j\right) \sum \left(\sum \left(S_{k,l}\, \mathrm{diag}\left(\mathrm{t}_{k,l}\right) \mathrm{DFT}_2\, G_l\right) \sum \left(S_m\, \mathrm{diag}\left(\mathrm{t}_m\right) \mathrm{DFT}_2\, G_{k,m}\right)\right)$$

C Code:

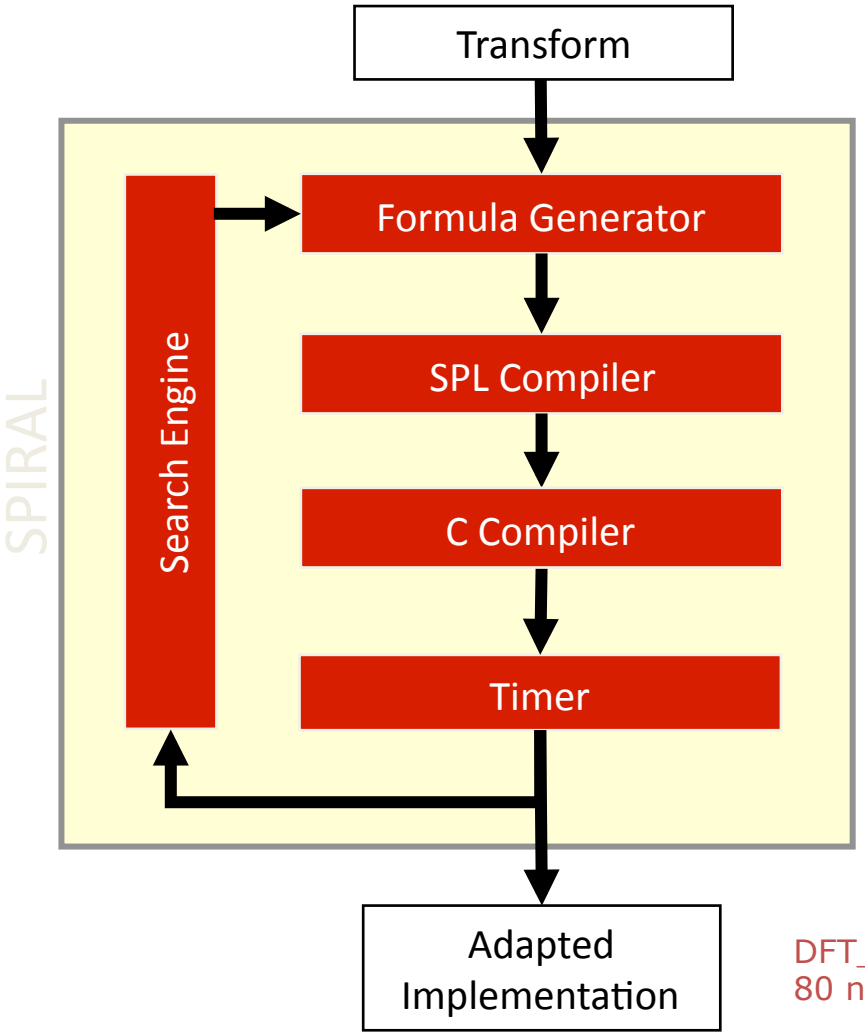
```
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```

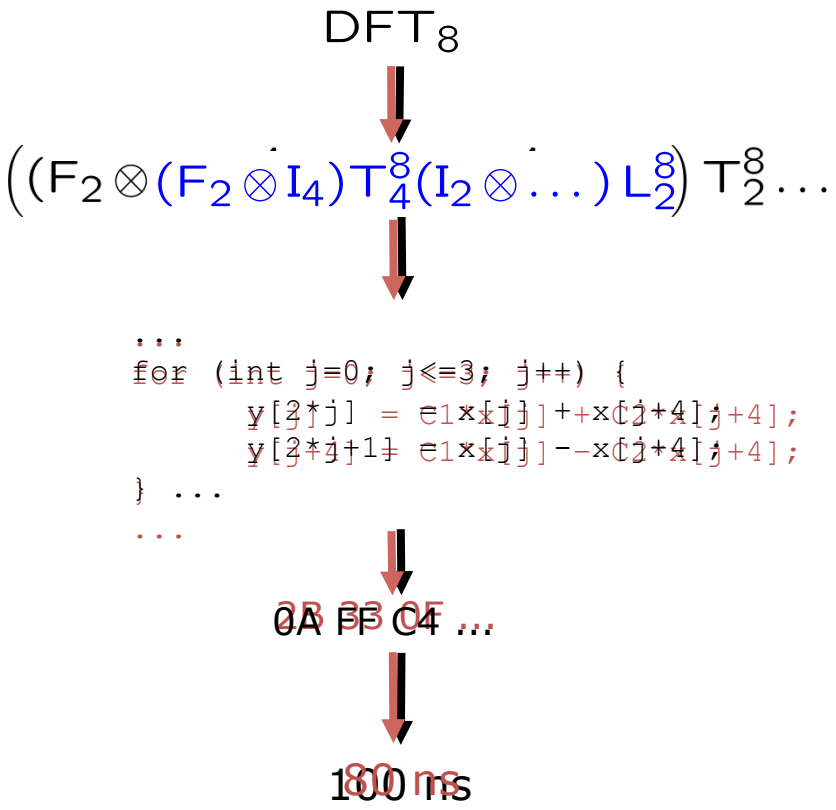
Optimisations aux différents niveaux

- parallelization vectorisation
- Optimisations boucles
- Ordonnancement instructions, Constantes, …

SPIRAL Architecture [Johnson&al IEEE 05]

Recherche empirique parmi plusieurs factorisations (auto-tuning)





DFT_8.c
80 ns

Plan de l'exposé

I. DFT et FFT

II. Améliorations algorithmiques de la DFT

Découpes récursive, algorithmes cache et processor oblivious

Synthéses optimisées : FFTW, SPIRAL

III. Améliorations algorithmiques reposant sur la DFT

Arithmétique exacte : entiers, polynômes, matrices

Codes correcteurs d'erreurs

DFT et calcul exact

- Requis pour DFT / FFT et FFT^{-1} : +, × *n* inversible ω racine primitive
- Pour tout n, il existe un entier M avec une racine primitive $n^{ème}$ de l'unité ω modulo M :
  $\omega^n = 1 \pmod M$ et pour $0 < j < n$: $\omega^j \neq 1 \pmod M$
- Exemple: dans $A = Z_{11}^*$

  et pour n=10 (n^{-1}=10)

	^2	^3	^4	^5	^6	^7	^8	^9	^10
1	1	1	1	1	1	1	1	1	1
2	4	8	5	10	9	7	3	6	1
3	9	5	4	1	3	9	5	4	1
4	5	9	3	1	4	5	9	3	1
5	3	4	9	1	5	3	4	9	1
6	3	7	9	10	5	8	4	2	1
7	5	2	3	10	4	6	9	8	1
8	9	6	4	10	3	2	5	7	1
9	4	3	5	1	9	4	3	5	1
10	1	10	1	10	1	10	1	10	1

  - Modulo 11, ω = 2 est une racine primitive $10^{ème}$ de l'unité (6, 7, 8 aussi)
    - donc $\omega^{2=}2^2=4$ est une racine $5^{ème}$ de l'unité ($6^2=3$, $7^2=4$, $8^2=9$ aussi)
    - et $2^{-4} = 2^{10-4} = 2^6 = 9$.
- Le corps fini à *q* éléments possède une racine primitive nième de l'unité si *n* divise *q*-1.

DFT sur un anneau général

- Ajout « virtuel » de racines de l'unité pour la DFT
  - Utilisation de puissance de 2 comme racine de l'unité

- Dans un anneau A avec 2 inversible :
  - Calcul dans $A[X] / (X^n + 1)$ $\Rightarrow$ $X^{2n} = 1 \pmod{X^n + 1}$
  - $\omega = X$ est une racine primitive $2n$-ième de l'unité

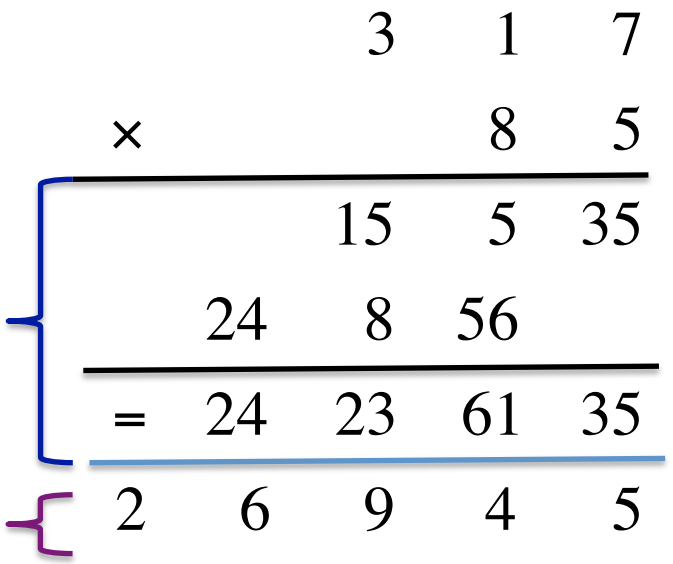
  - Calcul récursif par DFT sur des blocs de $\sqrt{n}$ chiffres
    - Multiplication O(n.log n. loglog n .Logloglog n … log*n)

DFT et multiplication d'entiers

Entiers de n chiffres en base B

Multiplication sans retenue (DFT)
calculs modulo* M = *mot machine
Il suffit $M < n \cdot B^2$

propagation de retenue en O(n)



- Choix de M : « premier de Fourier » : $p\text{-}1=u.2^k$
  - Exemple: $108.2^{57}+1$, premier et $\omega=64$ racine 2^{57}-ième de 1.
  - Avec B $=2^{20}$ (blocs de 20 bits), entiers de n $\leq 14.10^6$ blocs de 20 bits : permet de multiplier des entiers 30 Mbits = en O(n.log n)

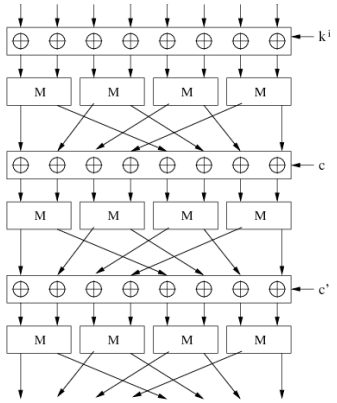
- Le surcoût lié à la redondance peut-être évité
  - Modulo 3 nombres premiers de 64 bits (« 3-primes »)

DFT et arithmétique entière

- Multiplication d'entiers de n bits
  - FFT sur des entiers de √n bits :
    coût = $n.\log n.\log\log n$ [Schönhage,Strassen 71]
  - Découpe avec ≠ décompositions mixtes $n=n_1.n_2$
    coût = $n.\log n.\ 2^{O(\log^* a)}$ [Fürer 07] [De, Saha, Kurur Saptharish 08]

- Division euclidienne, pgcd, inverse modulaire en temps quasi linéaire $\tilde{O}(n)$

- Cryptographie



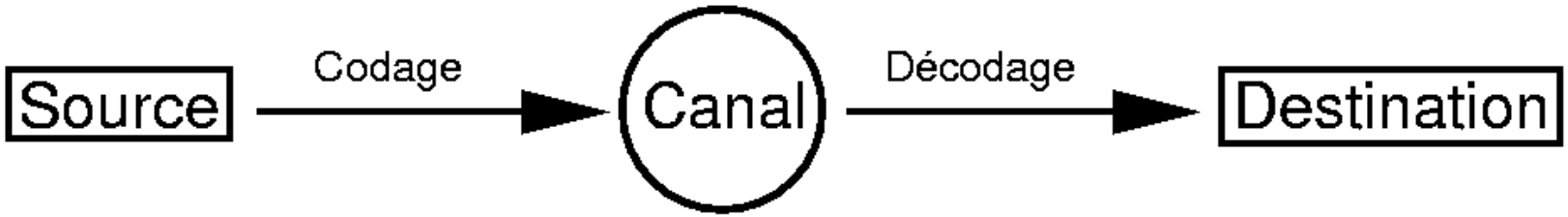
Autres algorithmes reposant sur la FFT

- Polynômes sur un anneau A[X]
  - représenter du polynôme comme un entier

[Cantor, Kaltofen 1991]

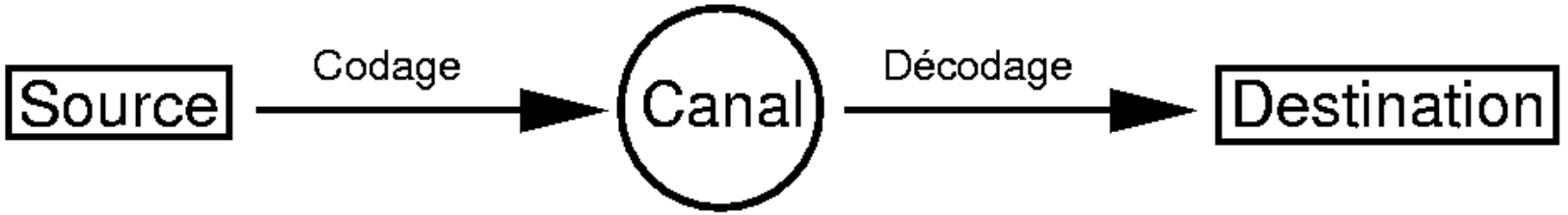
- Opérations sur certaines matrices creuses structurées
  - Liées à de l'arithmétique polynomiale :
    - Circulantes, Toeplitz

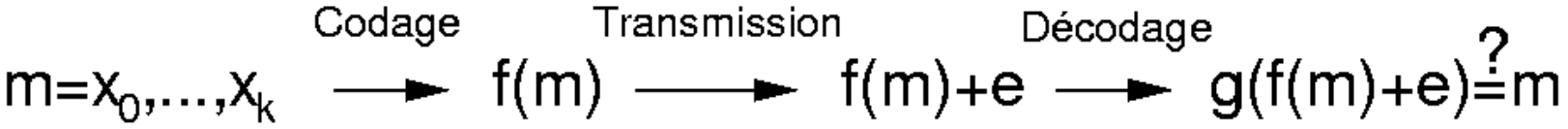
Transmission numérique



- Le code doit répondre à différents critères :
  - Rentabilité : compression des données […FFT…]
  - Sécurité de l'information : confidentialité, intégrité, authentification, […FFT…]
  - **Tolérance aux fautes** : correction/détection d'erreurs.

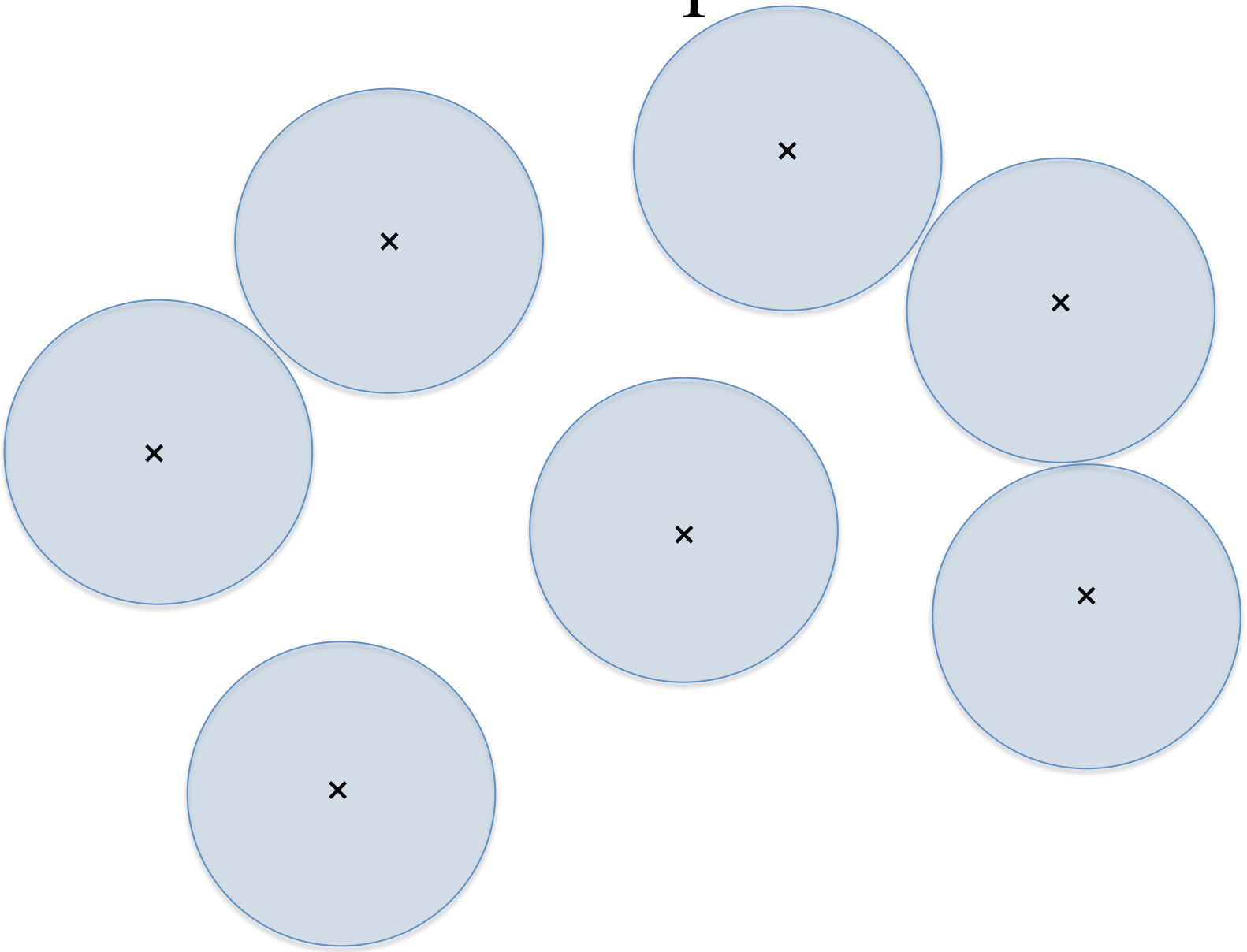
Transmission numérique



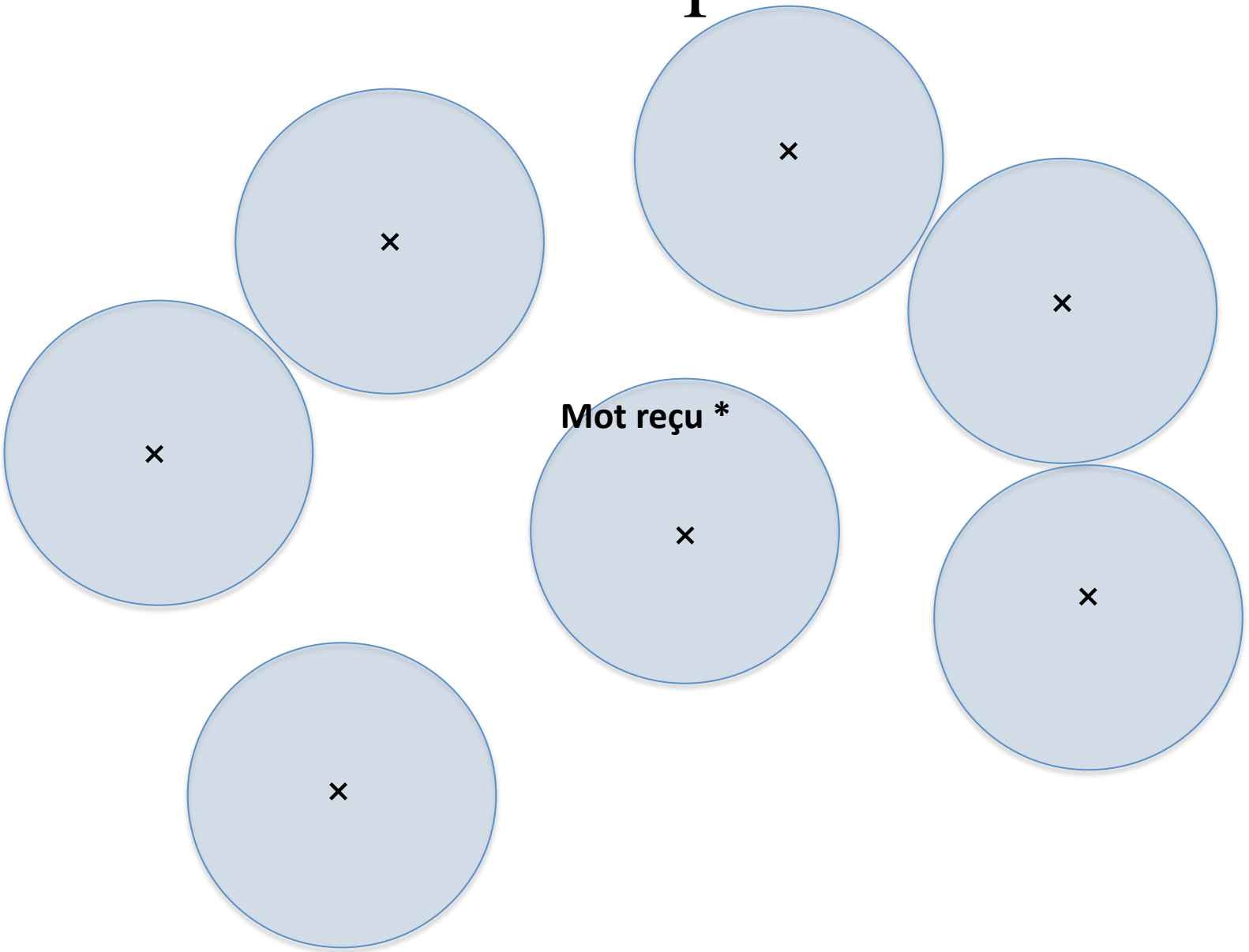


- Détection/correction d'erreurs:
  - Un mot source de *k* symboles est représenté univoquement par un mot de *n*=*k*+*r* symboles = *mot de code*
  - Déf: Distance = nombre minimum de symboles qui diffèrent entre deux mots de code

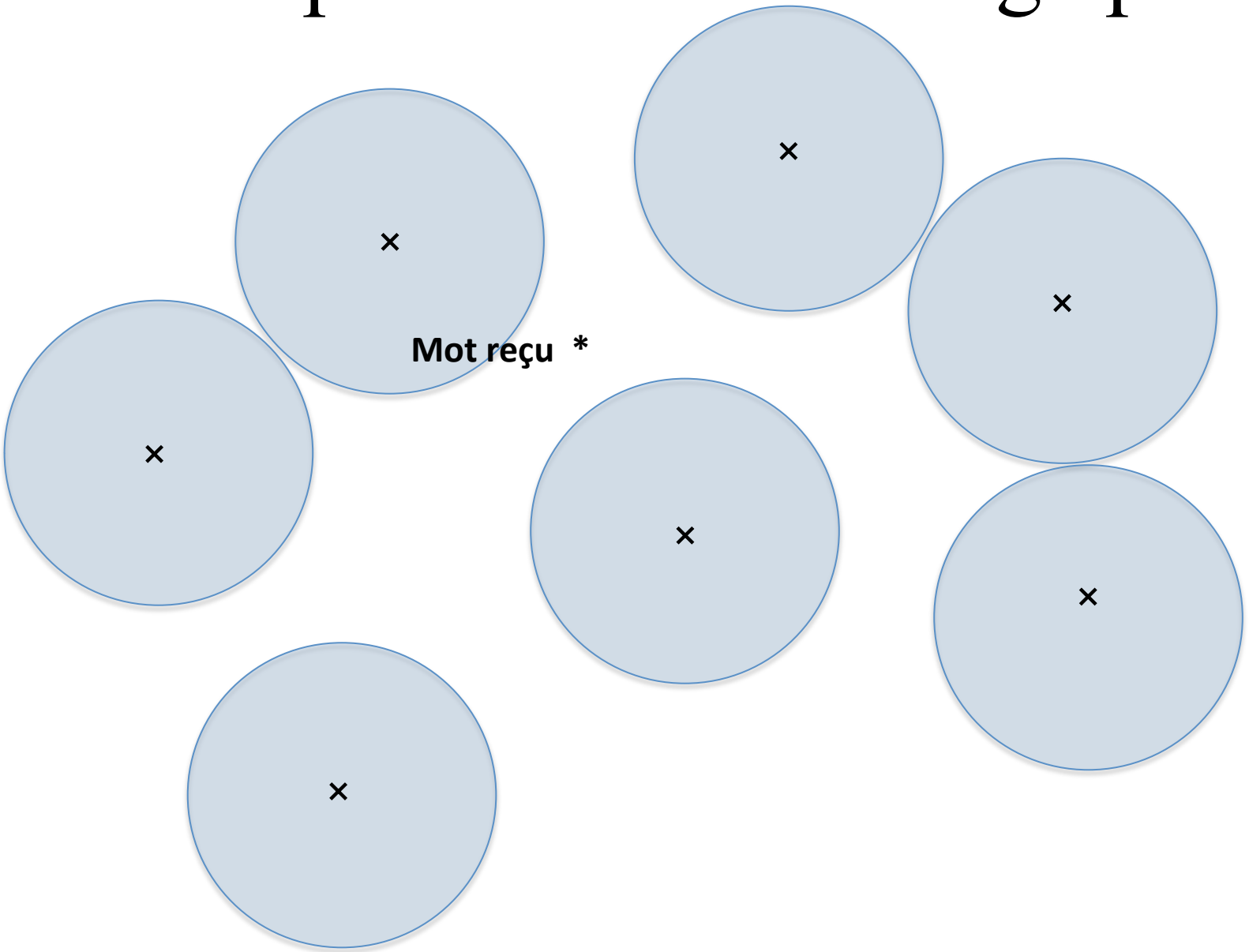
Codes par blocs



Codes par blocs



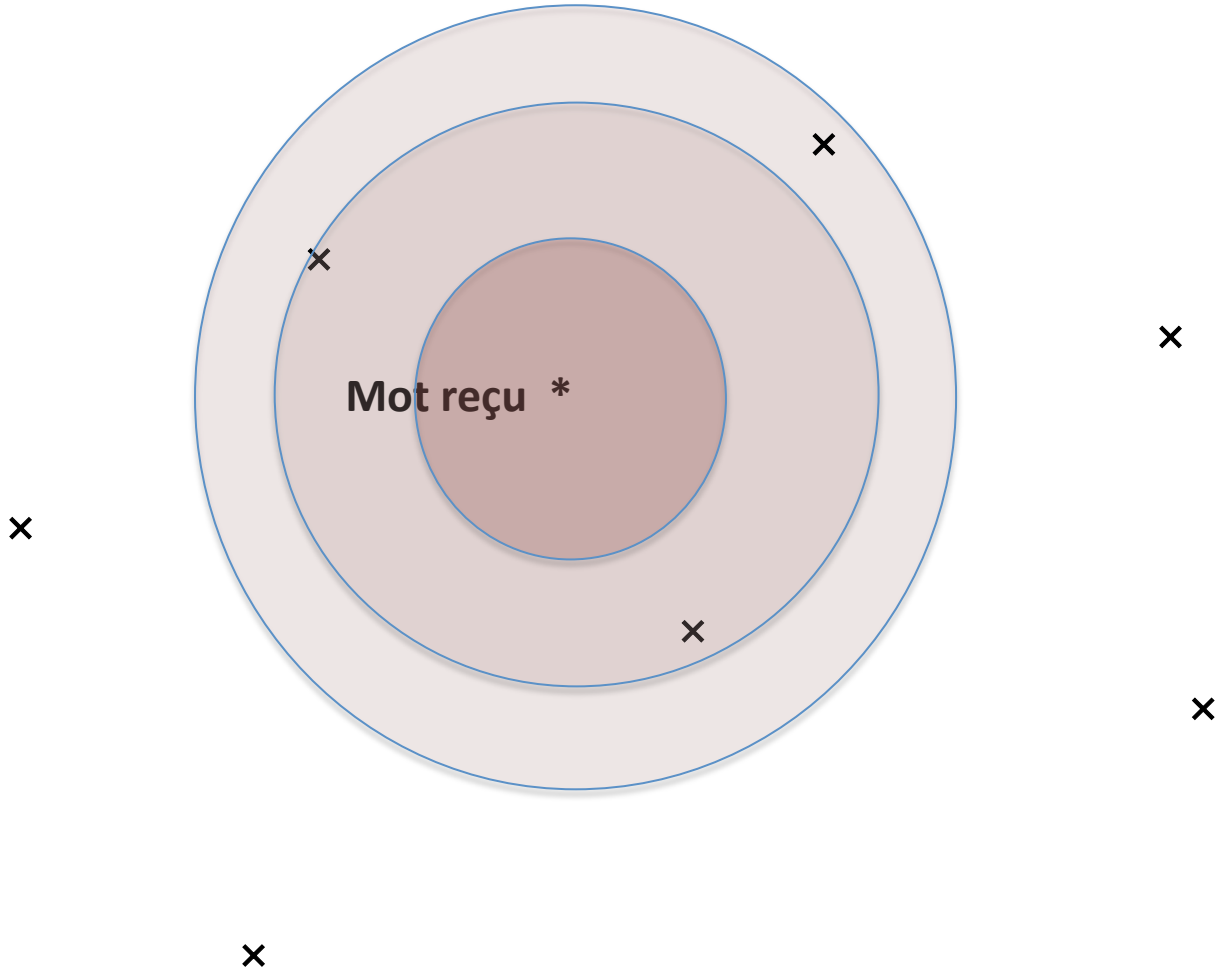
Codes par blocs – décodage par liste



Codes de Reed-Solomon et FFT

- Codage = évaluation d'un polynôme en n = k+r points (FFT)
  - Redondance de r symboles
  - Mot de code $[y_0, \ldots, x_{k-1}, y_k, \ldots, y_{n-1}] = \Omega\,[x_0, \ldots, x_{k-1}, 0, \ldots, 0]$
- Distance « maximale » (au sens classique)
  - Avec 2r symboles de redondance, on corrige r erreurs
- Correction: on reçoit $z=[z_0, \ldots, z_{k-1}, z_k, \ldots, z_{n-1}] = y+e$ avec e de poids $\leq r$
  - $\Omega^{-1}z = \Omega^{-1}y + \Omega^{-1}e = x + \hat{e} = [x_0+\hat{e}_0, \ldots, x_{k-1}+\hat{e}_{k-1}, \mathbf{\hat{e}_k, \ldots, \hat{e}_{n-1}}]$
  - les *r* symboles de $\mathbf{\hat{e}_k, \ldots, \hat{e}_{n-1}}$ engendrent linéairement $\hat{e}$ donc de calculer e

Codes par blocs – décodage par liste



Conclusion

- La DFT est une transformée « non triviale » quasi-linéaire ($n \log n$)
- Complexité algorithmique « multi-critère »
  - Nombre d'opérations, défauts cache, profondeur parallèle, …
  - Vers de nouvelles analyses des algorithmes:
    - « Variante » simultanément optimale: #opérations, cache, profondeur parallèle
- Synthèse d'algorithme (« algorithmes adaptatifs »)
  - Nombreuses variantes, dépendants de l'architecture (ex. parallèles, GPUs)
- Importance de la DFT (via FFT et ses variantes)
  - en algorithmique « exacte » aussi
- Impact important aussi sur le calcul
  - Algorithmique quantique

Références

- **Remerciements**: Marc Tchiboukdjian, Jeremy R. Johnson

- M. T. Heideman, D. H. Johnson, C. S. Burrus, *Gauss and the history of the fast Fourier transform*, *IEEE ASSP Magazine 1 (4), 14–21(1984).*

- Steven G. Johnson and Matteo Frigo, "A modified split-radix FFT with fewer arithmetic operations", *IEEE Trans. Signal Processing* **55** (1), 111–119 (2007).
- Matteo Frigo and Steven G. Johnson, "IThe Design and Implementation of FFTW3," *Proceedings of the IEEE* **93** (2), 216–231 (2005).
- Steven G. Johnson and Matteo Frigo,, « Implementing FFT in practice » *cnx.org/content/m16336/ , 5/2011*

- Franz Franchetti and Markus Püschel **Fast Fourier Transform** in Encyclopedia of Parallel Computing, Eds. David Padua, Springer 2011
- Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson and Nicholas Rizzolo
  SPIRAL: Code Generation for DSP Transforms Proceedings of the IEEE, ``Program Generation, Optimization, and Adaptation", Vol. 93, No. 2, pp. 232- 275, 2005

- Peter Burgiser, Michael Clausen, M. Amin Shokrollahi, Algebraic complexity theory » Springer Verlag, 1997
- Martin Fürer, "Faster integer multiplication", STOC 2007 Proceedings, pp. 57–66.

?