

Multiplication dans $\mathbb{F}_2[X]$

R. P. Brent, P. Gaudry, [E. Thomé](#), P. Zimmermann

Plan

- 1. Introduction**
- 2. Petites tailles**
- 3. Tailles moyennes**
- 4. Grandes tailles**

1. Introduction

2. Petites tailles

3. Tailles moyennes

4. Grandes tailles

Motivations

On souhaite multiplier des polynômes binaires : dans $\mathbb{F}_2[x]$.

Opération fréquemment rencontrée.

- factorisation de polynômes, tests d'irréductibilité ;
- applications crypto (quelques-unes) ;
- moins évident : algèbre linéaire creuse sur \mathbb{F}_2 ;
- etc etc.

Représentation des données

Polynôme $x^3 + x^2 + 1$ \rightarrow entier machine $(1101)_2$ (“dense”).

- $\text{deg} \leq 63$: un **mot machine** (64-bit).
- $64 \leq \text{deg} \leq 127$: deux mots.
- ...

Aspect matériel : • add : trivial ;

• mul : facile ; beaucoup plus facile que mul entier !

• Hors sujet ici.

Aspect logiciel : • add : trivial (xor) ;

• mul : pénible (en attendant PCMULQDQ).

Objectifs

Notre contexte

- logiciel uniquement.
- vitesse partout : de 64 à 2^{32} coefficients.

Ce qui existe

On trouve en général :

- Multiplication parfois rapide pour 1, 2 . . . jusqu'à quelques mots ;
- Karatsuba au-delà ;
- et c'est tout.

Référence : [NTL](http://shoup.net/ntl) : `shoup.net/ntl`

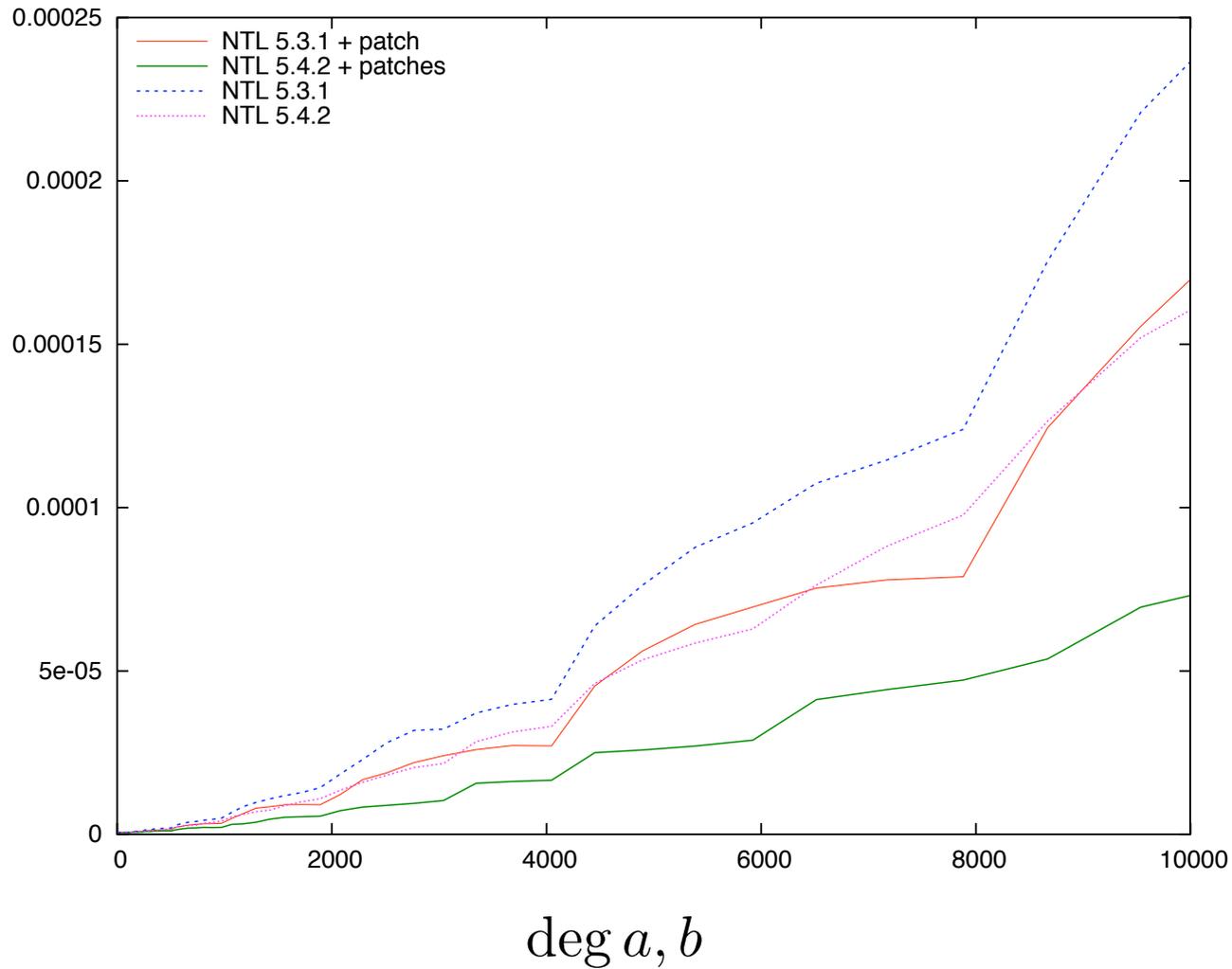
Très rarement, on peut trouver :

- Instructions CPU-spécifiques ;
- Toom-Cook ;
- Multiplication optimisée pour des opérandes déséquilibrées ;
- FFT (Schönhage ternary + Cantor additif).

gf2x fait tout cela !

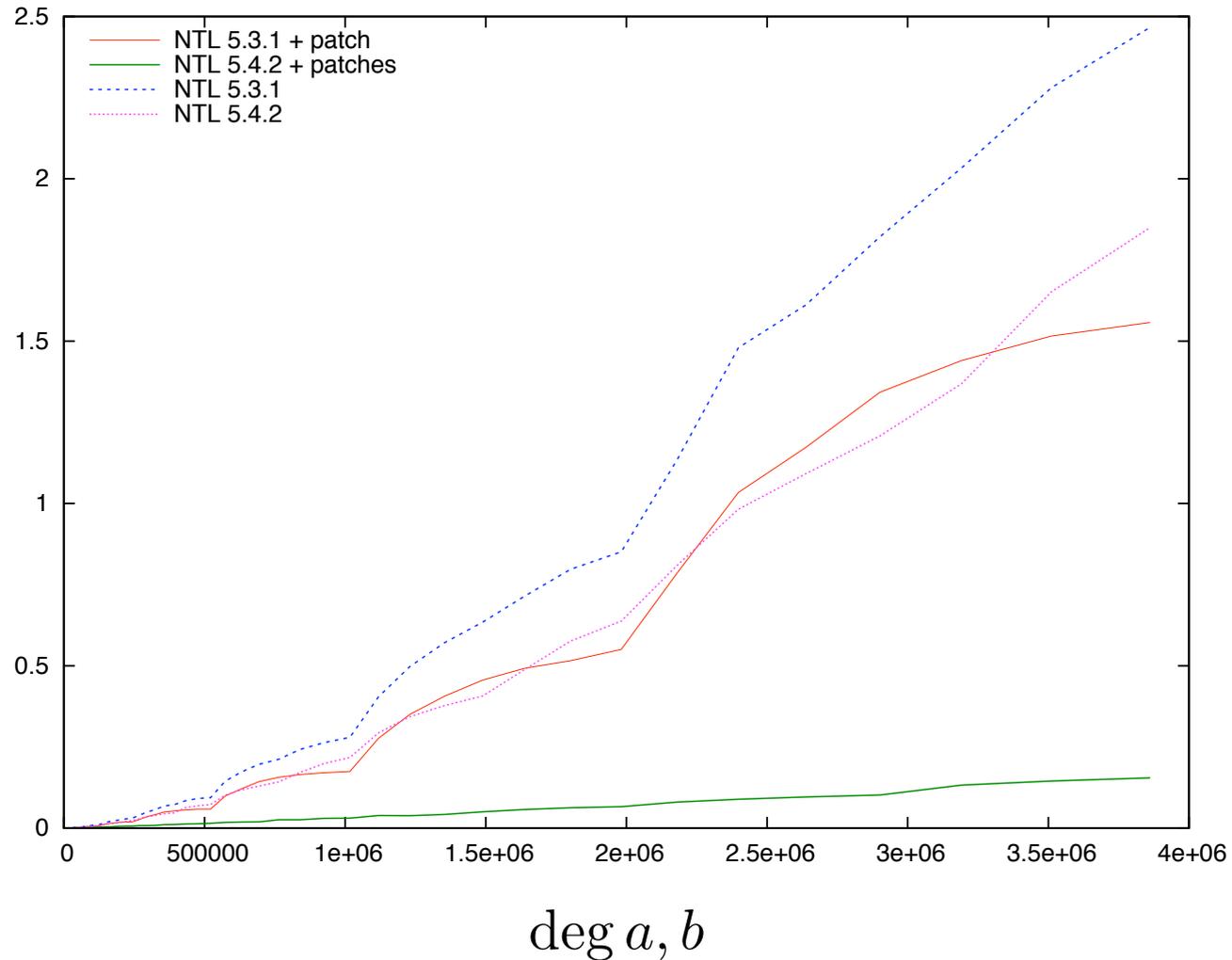
Et ça va plus vite !

Temps en secondes, Core2 2.4GHz



Et ça va plus vite !

Temps en secondes, Core2 2.4GHz



Notations usuelles

- **Découpage** : d'un polynôme $a(x)$ en tranches de s bits :

$$a(x) = A(x, x^s),$$

$$A(x, t) = A_0(x) + A_1(x)t + A_2(x)t^2 + \dots$$

et $\deg A_i < s$.

- **Recollage** : à partir de $A(x, t)$, calculer $a(x) = A(x, x^s)$.

Ces deux opérations ne sont que des jeux d'écriture.

Parfois, quelques variations sur le thème.

1. Introduction

2. Petites tailles

3. Tailles moyennes

4. Grandes tailles

deg < 64 : mul1

Du classique : $c = a \times b$ est calculé par une méthode de fenêtre (fixe).

- On tabule les multiples $g \times b$, pour $\deg g < s$ ($s = \text{window size}$).
- Découpage $a = A_0 + A_1x^s + A_2x^{2s} + \dots$.
- $c = A_0 \times b + (A_1 \times b)x^s + (A_2 \times b)x^{2s} + \dots$.

Operations requises : **shifts**, **XORs**.

Pour $\deg < 64$, on travaille uniquement avec des mots machine :

-  Pour $\deg b = 63$, le calcul de $A_i \times b$ déborde !
- La « réparation » se fait assez bien (cf papier).

mul1 : suite

- Essai-erreur pour déterminer la taille de fenêtre optimale ;
- 64×64 : ~ 75 cycles Intel core2 ; ~ 85 cycles AMD k8.
- $64k \times 64$: même principe.

Travailler en SIMD

Et pour 128×128 ?

- Karatsuba \Rightarrow **trois** 64×64 .
- Schoolbook nécessite $a \times b_{\text{low}}$ et $a \times b_{\text{high}}$ \Rightarrow **deux** 128×64 .
- **MAIS** $a \times b_{\text{low}}$ and $a \times b_{\text{high}}$ peuvent être calculés en parallèle (**SIMD**).
- Les instructions SIMD `x86_64` fournissent les shifts et XORs.
 - Accessible depuis le compilateur (`gcc`, `icc`, `MSVC`).
 - L'assembleur n'est pas un passage obligé.

128×128 : • ~ 129 cycles Intel `core2` ;

• ~ 226 cycles AMD `k8`.

• Plus rapide que Karatsuba.

1. Introduction
2. Petites tailles
3. **Tailles moyennes**
4. Grandes tailles

Tailles moyennes

Encore du classique : de 2 à 9 mots, [Karatsuba](#) codé en dur.

⇒ Pas de [branchement](#).

Exemple pour `mul4` :

```
mul2 (c, a, b);
mul2 (c + 4, a + 2, b + 2);
aa[0] = a[0] ^ a[2]; aa[1] = a[1] ^ a[3];
bb[0] = b[0] ^ b[2]; bb[1] = b[1] ^ b[3];
c24 = c[2] ^ c[4];
c35 = c[3] ^ c[5];
mul2 (ab, aa, bb);
c[2] = ab[0] ^ c[0] ^ c24; c[3] = ab[1] ^ c[1] ^ c35;
c[4] = ab[2] ^ c[6] ^ c24; c[5] = ab[3] ^ c[7] ^ c35;
```

Tailles moyennes

Encore du classique : de 2 à 9 mots, [Karatsuba](#) codé en dur.

⇒ Pas de [branchement](#).

Cycles, Intel core2.

deg	NTL	LIDIA	ZEN	nous
63	99	117	158	75
127	368	317	480	132
191	703	787	1 005	364
255	1 130	988	1 703	410
319	1 787	1 926	2 629	806
383	2 182	2 416	3 677	850
447	3 070	2 849	4 960	1 242
511	3 517	3 019	6 433	1 287

Et ensuite ?

Toom-3 : $\deg a < 3k$, write $a = A(x, x^k)$, $A(x, t) = a_0(x) + a_1(x)t + a_2(x)t^2$.

- **Évaluer** $(A(x, x_i))_{i=0,1,2,3,4}$ et $(B(x, x_i))_{i=0,1,2,3,4}$
- **Multiplier** : $C(x, x_i) = A(x, x_i)B(x, x_i)$.
- **Interpoler** : retrouver $C(x, t)$ à partir de $(C(x, x_i))_{i=0,1,2,3,4}$

Idée reçue : • Seulement pour $\#K \geq 4 \dots$

• car il faut 5 points d'évaluation (dans $\mathbb{P}^1(K)$).

- On choisit : $0, 1, \infty, x, x^{-1}$.
- Mieux : $0, 1, \infty, x^{64}, x^{-64}$: pas de shifts.
- Les degrés dans les appels récursifs augmentent un peu.

Timings

1 + deg	NTL	gf2x	method
1 536	0.011	0.01	TC3
4 096	0.053	0.039	K2
8 000	0.16	0.11	TC3W
10 240	0.26	0.19	TC3W
16 384	0.48	0.33	TC3W
24 576	0.93	0.59	TC3W
32 768	1.4	0.93	TC4
57 344	3.8	2.4	TC4
65 536	4.3	2.6	TC4
131 072	13	7.2	TC4

$\times 10^6$ cycles, Intel Core2.

1. Introduction
2. Petites tailles
3. Tailles moyennes
4. **Grandes tailles**

Grandes tailles

On veut aussi s'occuper de la zone FFT.

Plusieurs options :

- FFT entière et (beaucoup de) padding (Kronecker-Schönhage).
- FFT additive (Cantor).
- FFT ternaire (Schönhage).

Padding = mauvaise idée

Supposons qu'on sait **multiplier** (vite) dans \mathbb{Z} .

On s'en sert pour multiplier dans $\mathbb{F}_p[x]$.

Si $a(x) = a_0 + a_1x + \cdots + a_{N-1}x^{N-1}$, et $b(x)$ idem :

- On choisit $B = 2^k$ tel que $Np^2 < 2^k$.
- On calcule l'entier $\tilde{a} = a_0 + a_1B + \cdots$. Idem \tilde{b} .

$$\tilde{a} \cdot \tilde{b} = \sum_{\ell} \underbrace{\sum_{i+j=\ell} a_i b_j}_{\tilde{c}_\ell < Np^2 < B} B^\ell.$$

- On retrouve les coefficients de $c = a \times b$ par $c_i = \tilde{c}_i \bmod p$.

Pour $p = 2$, on perd un facteur $\log_2 N$, c'est **beaucoup**.

FFT additive

Supposons qu'on sait **multiplier** (vite) dans $F_k = \mathbb{F}_{2^{2^k}} = \mathbb{F}_2[\gamma]$.

On s'en sert pour multiplier dans $\mathbb{F}_2[x]$.

- Séparer les coefficients de a et b en **blocs de 2^{k-1} coefficients** :
 - Écrire $a = A(x, x^{2^{k-1}})$, $A(x, t) = a_0(x) + a_1(x)t + a_2(x)t^2 + \dots$.
 - $\tilde{a} = A(\gamma, t) \in F_k[t]$.
- $\tilde{a} \times \tilde{b} \in F_k[t]$.
- Puisque $\deg a_i b_j < 2^k$, alors $c = a \times b$ est tel que $\tilde{c} = \tilde{a} \times \tilde{b}$.

Multiplication dans F_k

Soit $s_1(x) = x^2 + x$, and $s_i(x) = \underbrace{s_1(s_1(\cdots s_1(x) \cdots))}_{i \text{ times}}$.

s_i satisfait de nombreuses propriétés :

- s_i est creux ; s_i est linéaire ; $s_{2^k} = x^{2^{2^k}} + x$.
- Soit $2^k \geq i$ et $W_i = \{\alpha \in \mathbb{F}_{2^{2^k}} \mid s_i(\alpha) = 0\} = \text{Ker } s_i$.
 W_i s.e.v. de $\mathbb{F}_{2^{2^k}}$; $\dim W_i = i$.

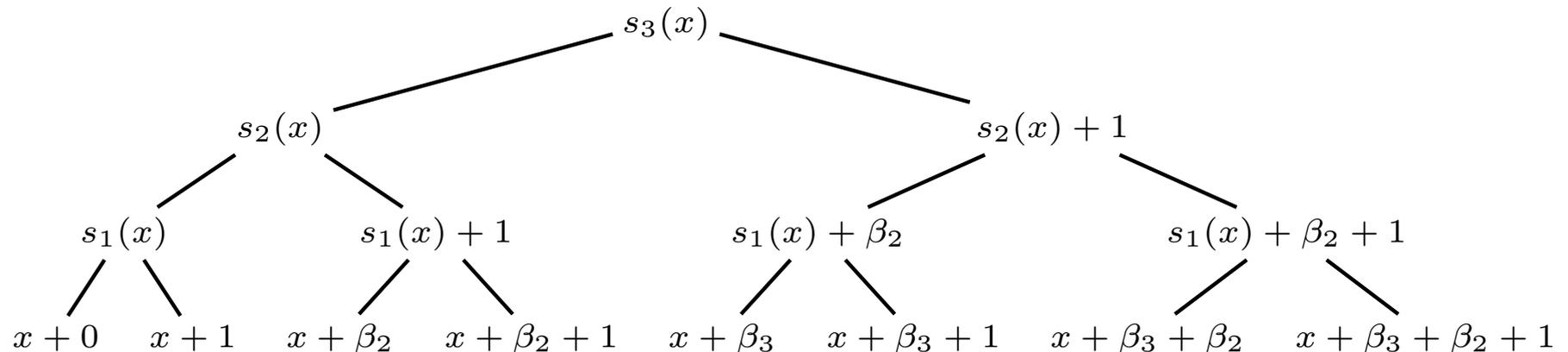
Comment multiplier $h = f \times g$ dans $F_k[x]$?

- Evaluer f and g en les points d'un W_i .
- multiplier point par point pour obtenir $\{f(\alpha) \times g(\alpha), \alpha \in W_i\}$.
- Interpoler : retrouver h à partir de $\{f(\alpha) \times g(\alpha), \alpha \in W_i\}$.

Multiplication dans F_k (2)

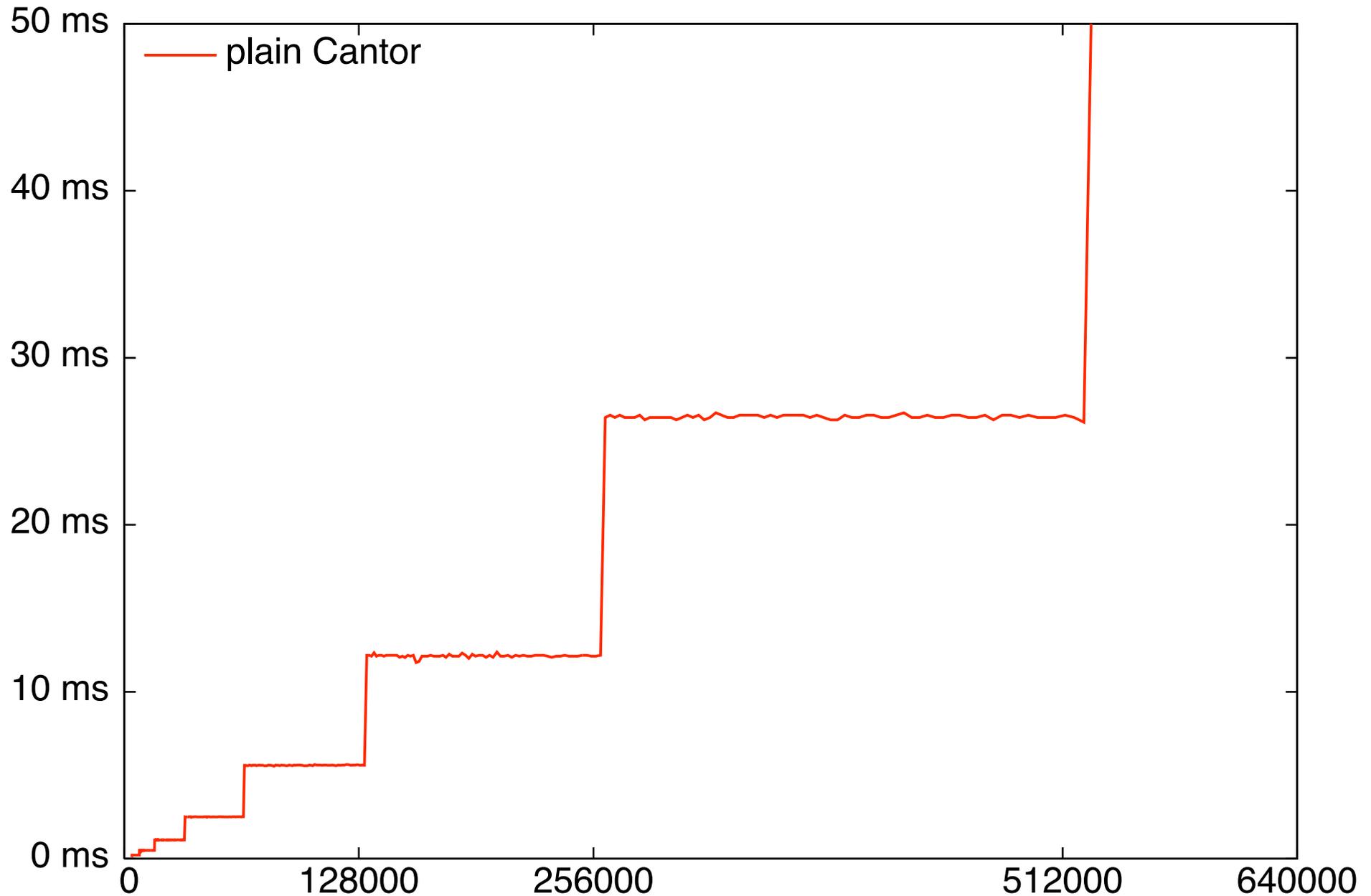
Multi-évaluation en W_i : sub-product tree :

$$\{f(\alpha), \alpha \in W_i\} = \{f \bmod (x + \alpha), \alpha \in W_i\}.$$



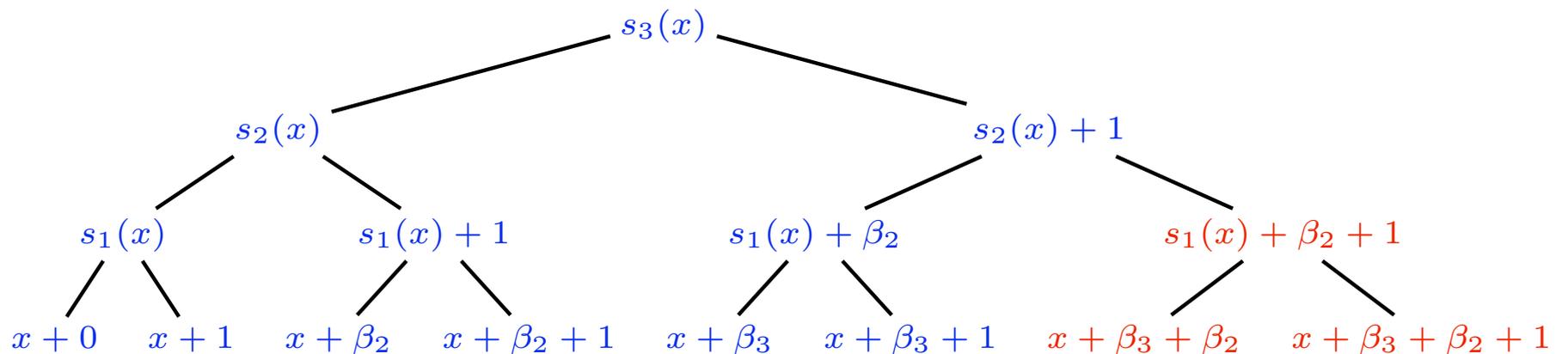
- fils-droit = 1 + fils-gauche.
- Seuls les coefficients constants vivent dans des extensions.
- s_j est creux, la réduction ne coûte pas cher.

Cantor : effet d'escalier



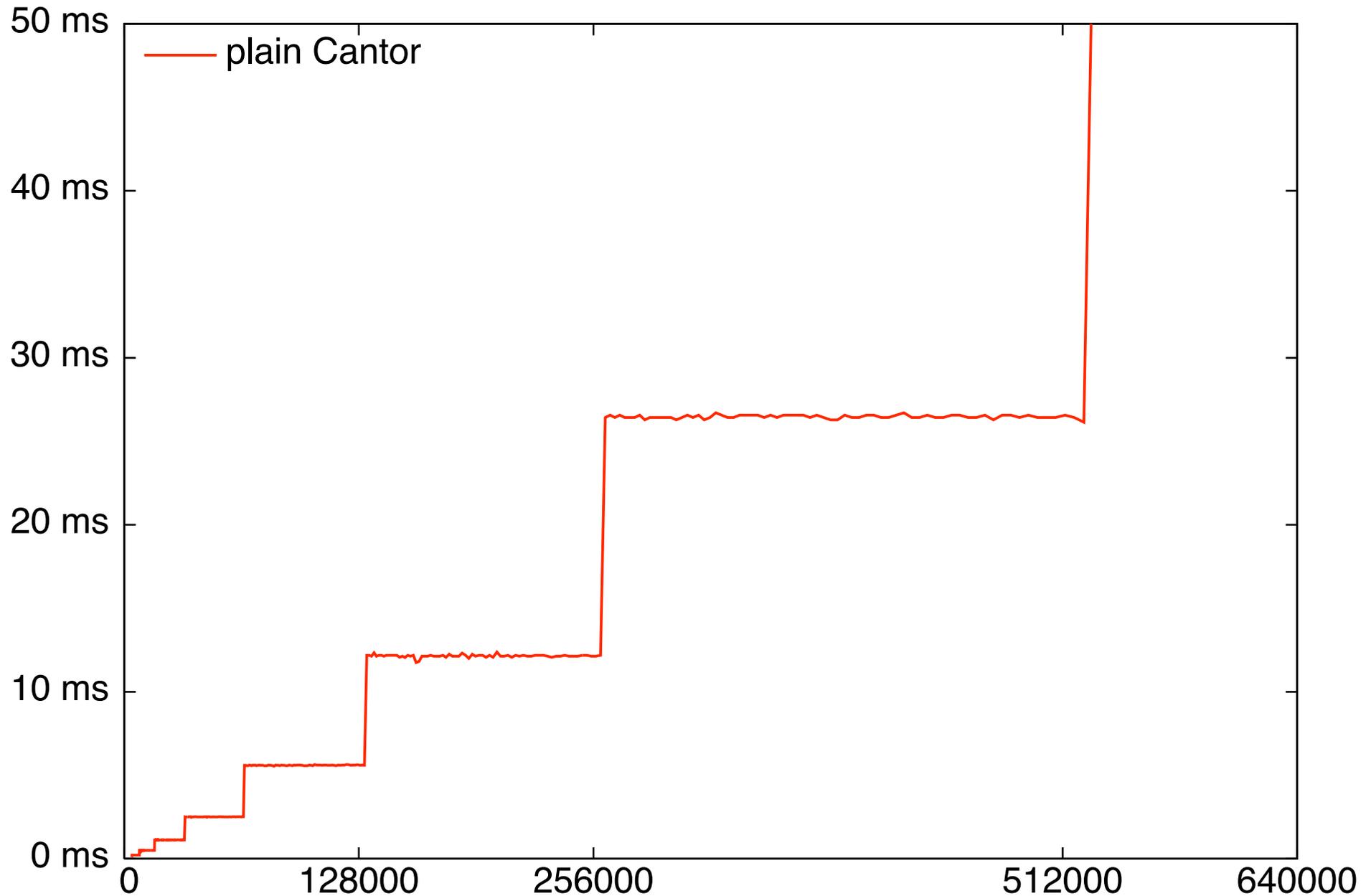
Une variante tronquée

- Ne pas évaluer en plus de points que nécessaire. Exemple pour 6 :

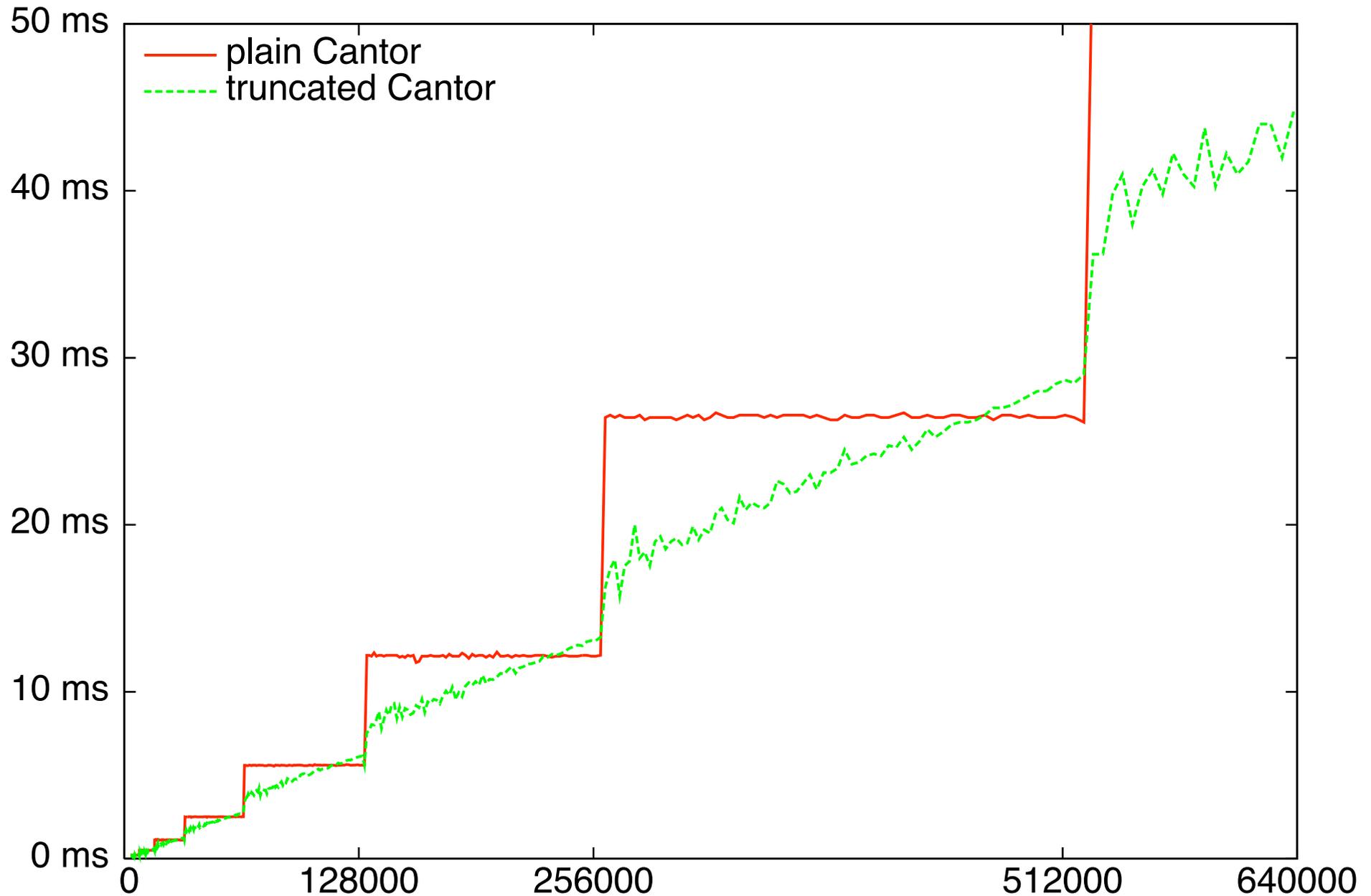


- Calcul modulo $s_2(x) (s_1(x) + \beta_2)$ au lieu de $s_3(x)$.
- Interpolation plus difficile. Utilise le sub-product tree deux fois.

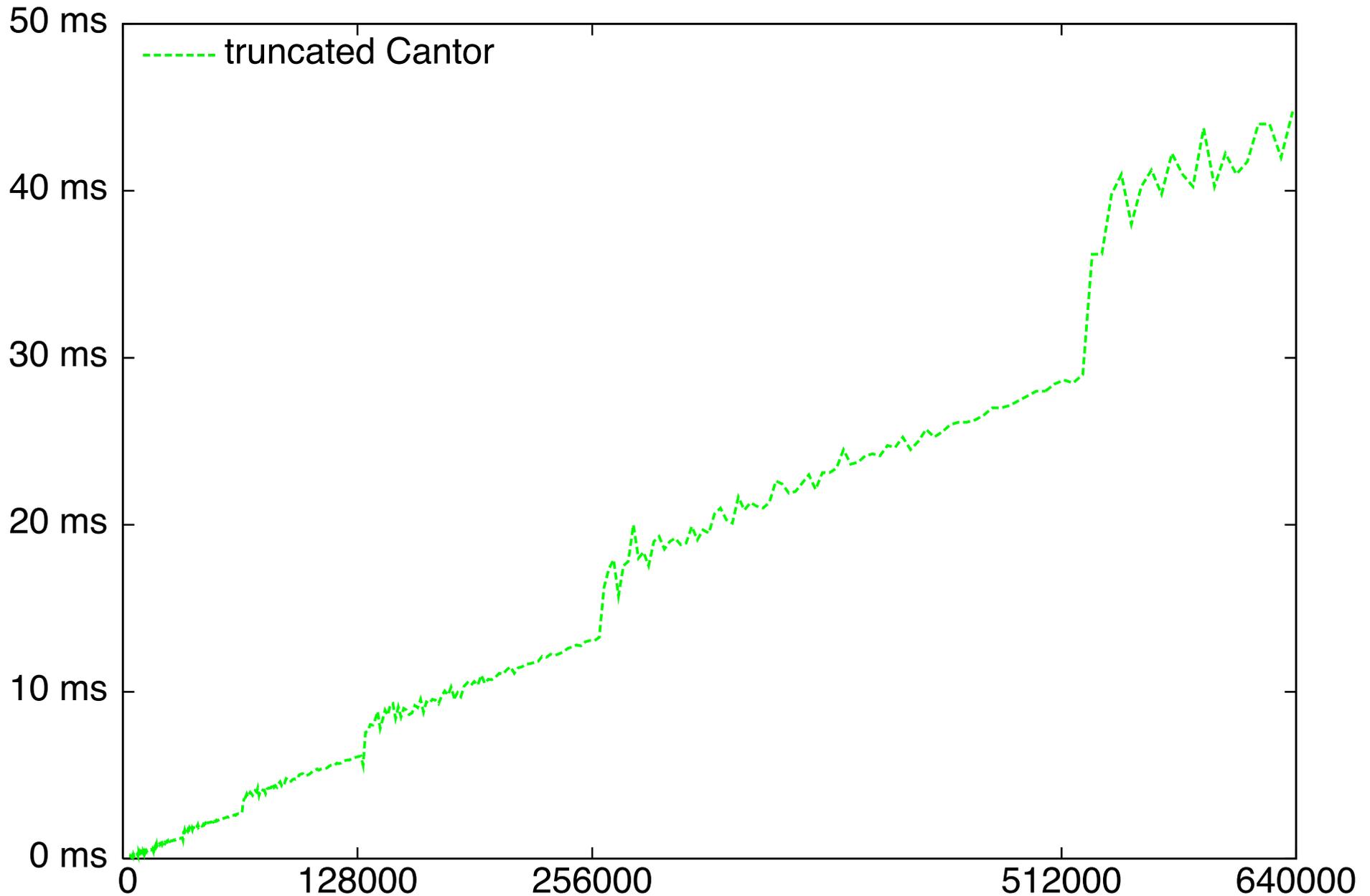
Performance de la FFT additive



Performance de la FFT additive



Performance de la FFT additive



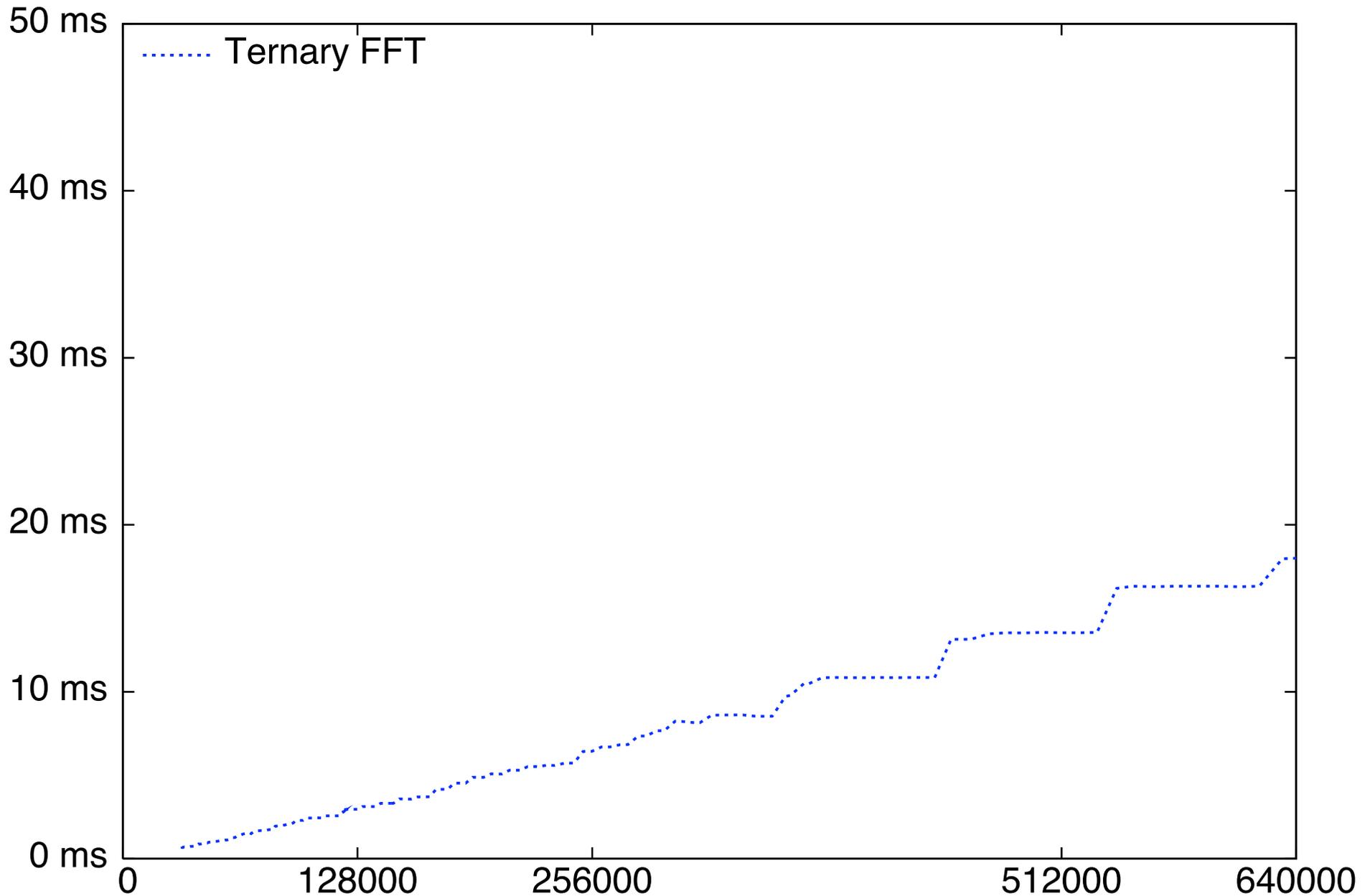
FFT ternaire – Schönhage

- FFT \rightsquigarrow racines 2^n -èmes de 1 ; pas glop pour char $K = 2$.
- Schönhage (1977) : calcul dans $R = \mathbb{F}_2[x]/x^{2L} + x^L + 1$; $L = \lambda 3^{k-1}$.
- x^λ est une racine 3^k -ème de 1 dans R .
- FFT ternaire pour multiplier des polynômes de $\deg < 3^k$ dans $R[t]$.
 - **Evaluer** $\hat{f} = \{f(x^{\lambda i}), 0 \leq i < 3^k\}$. (idem \hat{g}).
 - Multiplier **point par point** $\rightsquigarrow \widehat{fg}$; **multiplications dans R : récursif**.
 - **Interpoler** pour retrouver fg ; FFT encore car $\hat{\hat{f}} = f$.

Même technique de découpage qu'avant \Rightarrow multiplication dans $\mathbb{F}_2[x]$.

- En pratique, on travaille modulo $x^N + 1$ (pour un $N > \deg ab$).
- Cf papier.

FFT ternaire – Schönhage



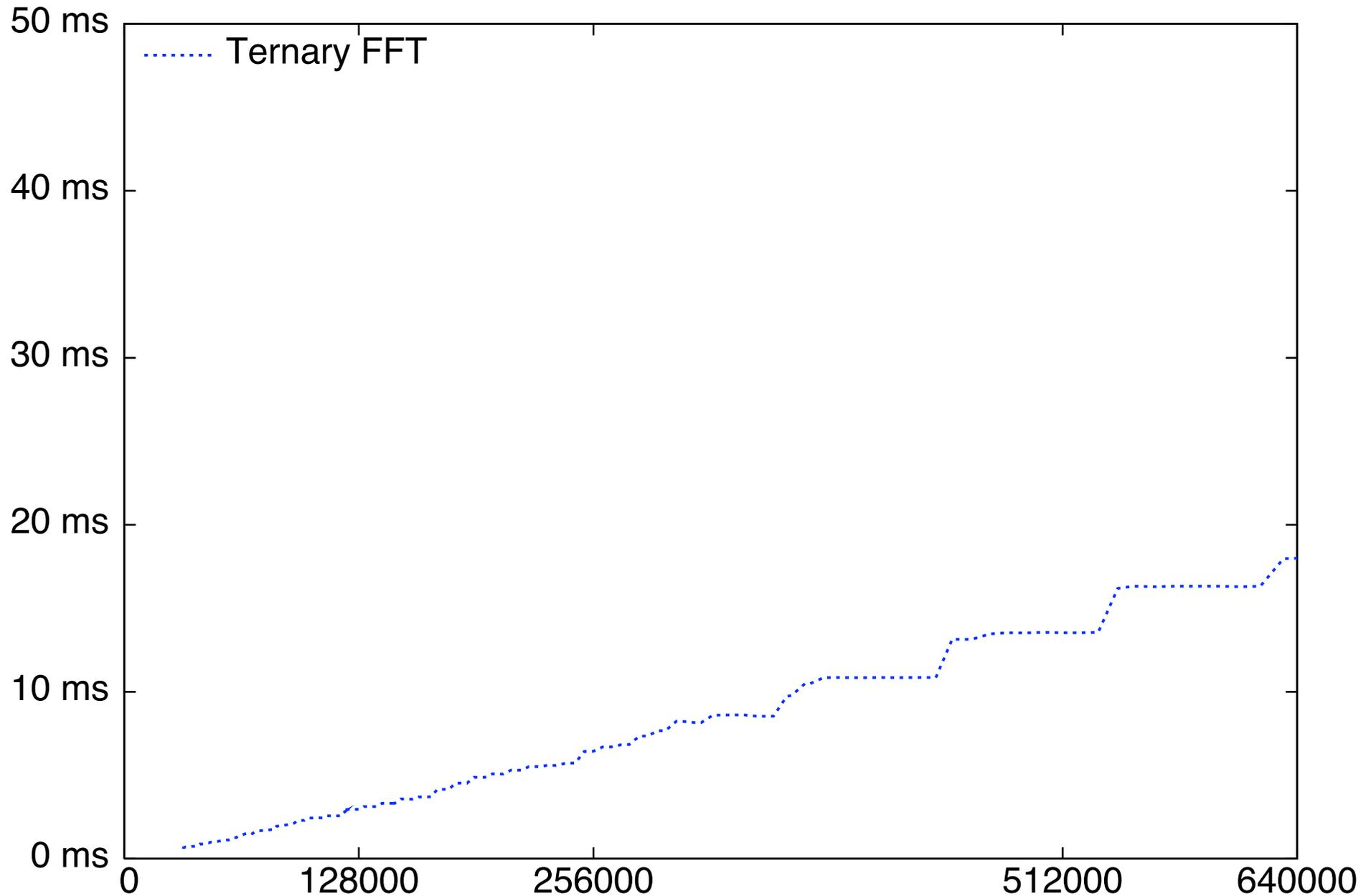
Couper en deux la FFT ternaire

Il y a un léger effet d'escalier.

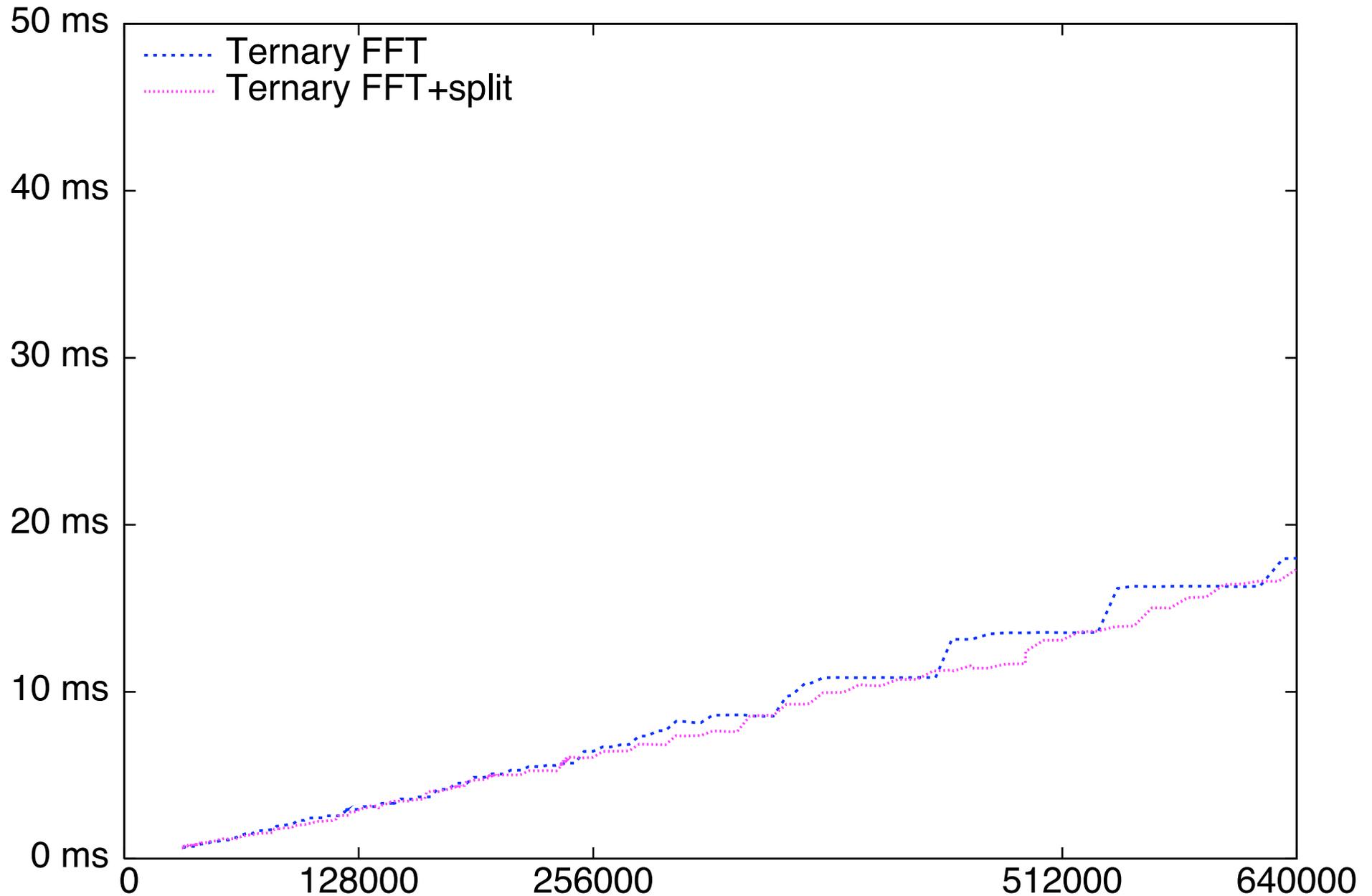
On peut calculer un produit de degré $< N$ en deux fois :

- Calcul d'un produit modulo $X^{N'} + 1$, $N' > N/2$.
- Calcul d'un second produit modulo $X^{N''} + 1$, $N'' > N'$.
- Reconstruction : très simple avec des XORs.

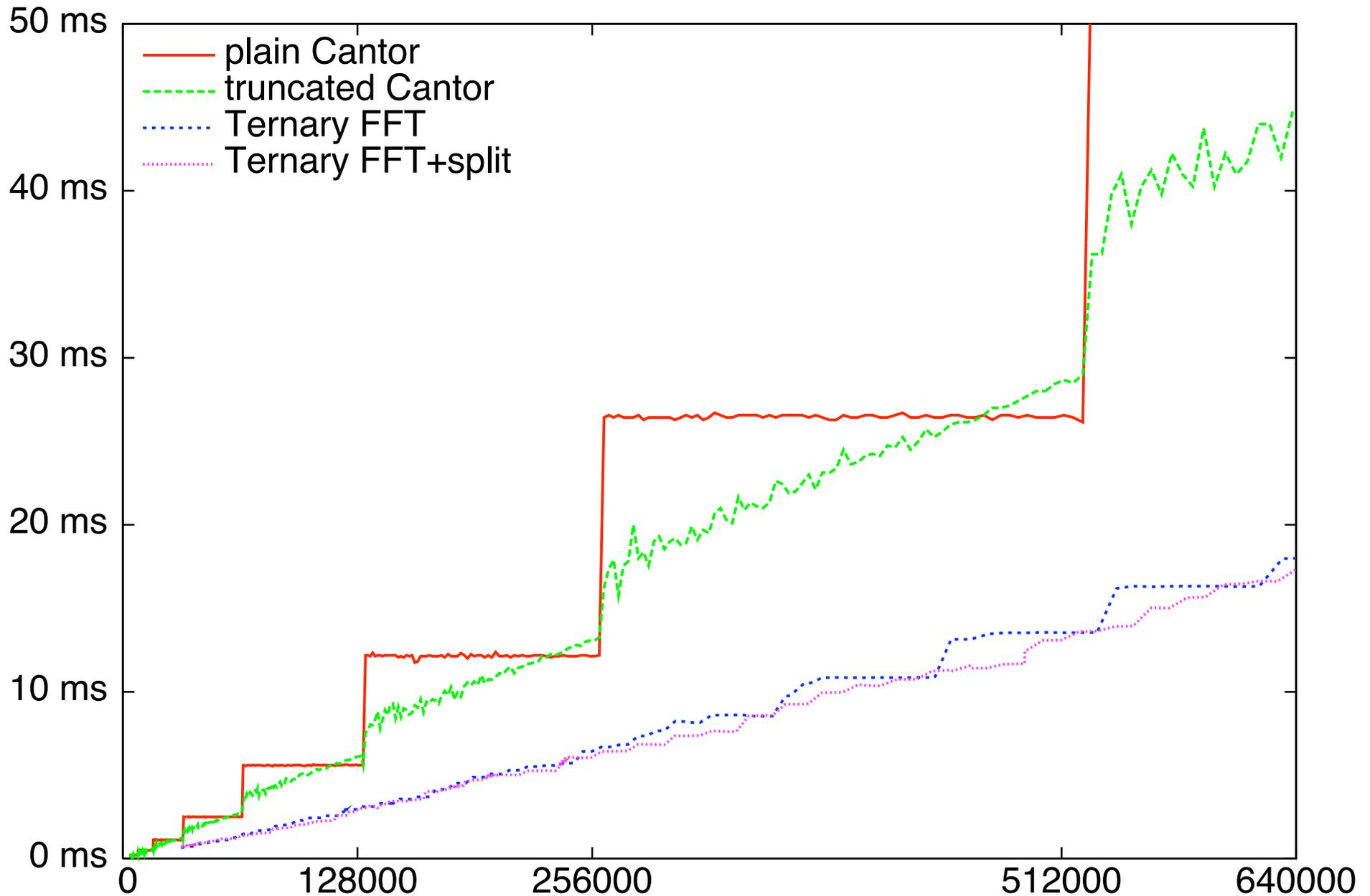
FFT ternaire + splitting



FFT ternaire + splitting



Comparaison Cantor – Schönhage



Comparaison Cantor – Schönhage

Attention :

- FFT Additive : produits point par point rapide.
- FFT Ternaire : évaluation / interpolation rapide.

Si on peut réutiliser les transformées (matrices sur $\mathbb{F}_2[x]$), la FFT additive est un meilleur choix.

Exemple pour $\deg ab < 2^{20}$:

- FFT additive : 57 ms, 2.3 ms dans les produits point par point.
- FFT ternaire : 28 ms, 18 ms dans les produits point par point.
- $n \times n$ matrix mult : $c_{\text{eval/interp}} * n^2 + c_{\text{pointwise}} * n^3$
- FFT additive + rapide pour 3×3 et au-delà.

Conclusion

- Significativement plus rapide que l'existant.
- Implementation disponible de deux algorithmes de FFT.
- URL : `rpbrent.com/gf2x.html`