
Berlekamp-Massey matriciel rapide, résolution de gros systèmes linéaires par « block Wiedemann »

Emmanuel Thomé

`Emmanuel.Thome@polytechnique.fr`

Laboratoire d'Informatique (LIX)

École polytechnique

France

Avec le soutien de l'ACI CRYPTOLOGIE



1. Introduction, motivations.
2. Algorithme de Wiedemann
3. L'algorithme de Wiedemann par blocs
4. Berlekamp-Massey matriciel
5. Paramètres de bloc optimaux
6. Détails d'un gros calcul

Notre problème :

Soit $M \in \mathcal{M}_{N \times N}(\mathbb{F}_q)$ singulière, trouver w tel que $Mw = 0$.

On retrouve ce problème (entre autres !) :

- Dans les algorithmes de factorisation d'entiers (QS, NFS)
- Dans les algorithmes de calcul de logarithme discret.

Dans les deux cas :

- N est **très grand** ($> 100,000$)
- M est très **creuse** : peu de coefficients par ligne
- **MAIS** : le corps de définition peut changer :
 \mathbb{F}_2 pour la factorisation (plus facile),
ou \mathbb{F}_q pour le logarithme discret (plus dur).

Rapidement, un exemple pour $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/f(X)$ (Adleman) :

- **Base de facteurs** : polynômes π_i de petit degré (\sqrt{n}).
- Tester, pour de nombreuses valeurs de m , si $X^m \bmod f(X)$ est **friable** (Seuls les π_i sont facteurs).
- Alors pour chacune de ces valeurs de m , on a des coefficients e_i t.q. :

$$X^m \equiv \prod_i \pi_i^{e_i} \bmod f(X),$$

$$m \equiv \sum_i e_i \log \pi_i \bmod (2^n - 1).$$

On obtient ainsi un **système linéaire**.

Cas du logarithme discret

Propriétés du système linéaire obtenu :

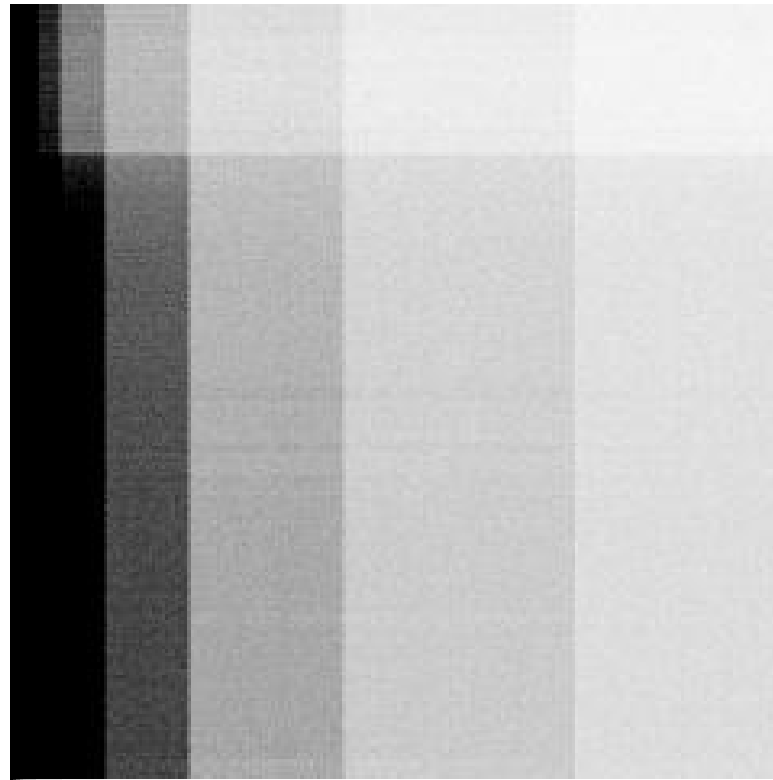
- Sa taille (N) est celle de la base de facteurs.
- Le système est défini modulo $2^n - 1$.
- Pour chaque m , presque tous les e_i sont nuls. Ce sont toujours des **petits** coefficients.

Dans notre cas, on a utilisé l'algorithme de Coppersmith (meilleure complexité) pour calculer des log. discrets sur $\mathbb{F}_{2^{607}} \Rightarrow$ **système linéaire défini sur $\mathbb{Z}/2^{607} - 1$** (corps).

$\mathbb{F}_{2^{607}}$ est un gros corps (record) \Rightarrow base de facteurs importante : degré 23, donc $N = 766\,150$.

Remarque : \sim un an sur 100 PCs pour obtenir la matrice.

Forme d'une matrice de log. discret



Noir \Leftrightarrow densité $\geq 0.01\%$

2. Algèbre linéaire creuse

Pour une matrice M de taille $N \times N$ avec γ coeffs/ligne, on peut utiliser :

- Gauss $\Rightarrow N^3$,
- Les algorithmes “creux” (Lanczos, Wiedemann) $\Rightarrow \gamma N^2$.

N énorme, γ ridicule \Rightarrow Le choix est vite fait !

On s'intéresse à l'algorithme de [Wiedemann](#).

Opération de base : [matrice](#) \times [vecteur](#) ; M agit comme une boîte noire.

On veut trouver w en $O(N)$ multiplications $M \times$ vecteur.

Algorithme de Wiedemann

- La suite des M^i est linéairement engendrée : polynôme minimal.
- x et y vecteurs aléatoires :
Les $a_i = x^T M^{i+1} y$ forment une suite linéairement engendrée.
- Le générateur linéaire est un diviseur du polynôme minimal.

Algorithme de Wiedemann

- La suite des M^i est linéairement engendrée : polynôme minimal.
- x et y vecteurs aléatoires :
Les $a_i = x^T M^{i+1} y$ forment une suite linéairement engendrée.
- Le générateur linéaire est un diviseur du polynôme minimal.

Algorithme :

- Calculer a_0, \dots, a_{2N} :
 $2N$ multiplications $M \times$ vecteur.
- Trouver le générateur P par **Berlekamp-Massey** ($< O(N^2)$).
Avec forte probabilité, $P = XQ =$ polynôme minimal de M .
- Calculer $Q(M)y$. Avec forte proba, $Q(M)y \in \text{Ker} M$.
Encore N multiplications $M \times$ vecteur.

Au total, $3N$ multiplications $M \times$ vecteur, $+O(N^2)$.

3. Une version “par blocs” (Coppersmith)

L’algorithme de Wiedemann est **séquentiel** :

$M^{i+1}y$ est obtenu à partir de $M^i y$.

Comment peut-on distribuer ce calcul ?

On remplace les vecteurs aléatoires x et y par des **blocs** de vecteurs aléatoires $\Rightarrow x \in \mathcal{M}_{N \times m}(\mathbb{F}_q)$, $y \in \mathcal{M}_{N \times n}(\mathbb{F}_q)$.

- Les $a_i = x^T M^{i+1} y$ sont des matrices $m \times n$,
On les regroupe en considérant $A(X) = \sum_i a_i X^i$.
 $A(X)$ est donc une **matrice de polynômes**.
- On ne calcule les a_i que jusqu’à $\frac{N}{m} + \frac{N}{n} (< 2N)$.
- Sur \mathbb{F}_q , deux colonnes de $A(X)$ peuvent être calculées **complètement indépendamment**.
- Sur \mathbb{F}_2 , on peut travailler sur 32 colonnes **simultanément**.

Ce qu'apporte la version par blocs

- Grâce à la distribution possible, l'étape du calcul des a_i est **facilitée**.
- Il faut un Berlekamp-Massey matriciel. C'est **plus dur**.

Pour pouvoir distribuer au maximum le calcul des a_i , il faut un Berlekamp-Massey matriciel très efficace.

- Coppersmith (1993) : $O((m + n)N^2)$.
- ET (2001) : Adaptation en $O((m + n)N \log^2 N)$,
en utilisant :
 - De la FFT pour multiplier des matrices de polynômes (matrices de plus de 1Go).
 - Une structure récursive.

4. Berlekamp-Massey matriciel

On veut généraliser le problème du calcul de générateur linéaire au cas d'une suite scalaire.

Problème : Étant donné $A(X) \in K[[X]]^{m \times n}$, trouver $F(X) \in K[X]^n$ tq :

$$A(X)F(X) \in K[X]^m.$$

(= approximants de Padé matriciels)

On regarde d'abord comment fonctionne le **cas scalaire** ($m = n = 1$) :

On maintient $(f_1, g_1, e_1, \delta_1)$ et $(f_2, g_2, e_2, \delta_2)$ tels que, pour t croissant :

$$A(X)f_i(X) = g_i(X) + X^t e_i(X),$$

$$\deg f_i \leq \delta_i,$$

$$\deg g_i < \delta_i.$$

Initialement $f_1 = 1, f_2 = X, \delta_i = t = 1$. À la fin, l'un des f_i est F .

Itération dans Berlekamp-Massey scalaire

Par exemple $\delta_2 \leq \delta_1$. L'équation $A(X)f_i(X) = g_i(X) + X^t e_i(X)$ donne :

$$\Rightarrow \begin{cases} f_1 \leftarrow f_1 - \frac{\lambda}{\mu} f_2 \\ f_2 \leftarrow X f_2 \end{cases}$$

C'est un **pivot de Gauss** (un peu amélioré) sur la matrice $[\lambda \ \mu]$.

Après :

Au bout du compte :

$$\begin{aligned} \exists i, t \geq N + \delta_i \\ > N + \deg g_i \end{aligned}$$

À chaque étape t , la matrice $[e_1(0) \ e_2(0)] (= [\lambda \ \mu])$ et les δ_i déterminent une matrice P telle que :

$$[f_1 \ f_2]^{\text{étape } t+1} = [f_1 \ f_2]^{\text{étape } t} \times P,$$

À chaque étape t , la matrice $[e_1(0) \ e_2(0)] (= [\lambda \ \mu])$ et les δ_i déterminent une matrice P telle que :

$$[f_1 \ f_2]^{\text{étape } t+1} = [f_1 \ f_2]^{\text{étape } t} \times P,$$

$$[e_1 \ e_2]^{\text{étape } t+1} = [e_1 \ e_2]^{\text{étape } t} \times P,$$

À chaque étape t , la matrice $[e_1(0) \ e_2(0)] (= [\lambda \ \mu])$ et les δ_i déterminent une matrice P telle que :

$$[f_1 \ f_2]^{\text{étape } t+1} = [f_1 \ f_2]^{\text{étape } t} \times P,$$

$$[e_1 \ e_2]^{\text{étape } t+1} = [e_1 \ e_2]^{\text{étape } t} \times P,$$

$$[\delta_1 \ \delta_2]^{\text{étape } t+1} = [\delta_1 \ \delta_2]^{\text{étape } t} \times P.$$

La matrice P est $\left\{ \begin{array}{l} \text{de taille } 2 \times 2 \text{ (ici),} \\ \text{de degré } \leq 1. \end{array} \right.$

L'algorithme est donc :

- Construire $[f_1 \ f_2]^{\text{étape } 1}$ et $[\delta_1 \ \delta_2]^{\text{étape } 1}$. $t \leftarrow 1$.
- Calculer $[e_1(0) \ e_2(0)]^{\text{étape } t}$. Déduire P .
- Déduire $[f_1 \ f_2]^{\text{étape } t+1}$ et $[\delta_1 \ \delta_2]^{\text{étape } t+1}$.
- $t \leftarrow t + 1$. Boucler autant qu'il faut.

Le cas matriciel fonctionne de manière similaire.

$A(X)$ était dans $K[[X]] \rightarrow$ maintenant : $K[[X]]^{m \times n}$.

On a $m + n$ candidats f_1, \dots, f_{m+n} , vérifiant (à l'étape t) :

$$A(X)f_i(X) = g_i(X) + X^t e_i(X),$$

$$\deg f_i \leq \delta_i,$$

$$\deg g_i < \delta_i.$$

On regroupe les f_i ensemble \rightarrow **une matrice** $f(X) \in K[X]^{n \times (m+n)}$.

De même, matrices $g(X)$, $e(X) \in K[X]^{m \times (m+n)}$, $\delta \in \mathbb{N}^{m+n}$.

Comme dans le cas scalaire, on fait du pivot de Gauss amélioré (...)

Identité vérifiée à chaque étape : $A(X)f(X) = g(X) + X^t e(X)$.

La matrice $e(X)$ est de rang maximal (m). Pour passer à l'étape $t + 1$:

- On élimine n colonnes par pivot.
- On « pousse » les autres (multiplication par X).
- Les δ_i n'augmentent en moyenne que de $\frac{m}{m+n}$.

Comme précédemment, à chaque étape t , $(e(X), \delta) \rightsquigarrow$ une matrice P telle que :

$$f(X)^{\text{étape } t+1} = f(X)^{\text{étape } t} \times P,$$

$$e(X)^{\text{étape } t+1} = e(X)^{\text{étape } t} \times P,$$

$$\delta^{\text{étape } t+1} = \delta^{\text{étape } t} \times P.$$

Point-clé : Connaître k coefficients de $e(X)$ permet d'avancer de k étapes.
c'est-à-dire : calculer $P^{\text{étape } t}, \dots, P^{\text{étape } t+k-1}$.

Stratégie :

- Regrouper les matrices P pour faire des **gros produits**
- Travailler seulement avec $e(X)$ et δ .
- Calculer seulement $f(X) \times P^{\text{étape } 1} \times \dots \times P^{\text{fin}}$ **à la fin**.

Pour calculer $P^{(a)} \times \dots \times P^{(a+b-1)}$ à partir de $e(X)^{(a)} \bmod X^b$ et $\delta^{(a)}$:

Une structure réursive

Point-clé : Connaître k coefficients de $e(X)$ permet d'avancer de k étapes.
c'est-à-dire : calculer $P^{\text{étape } t}, \dots, P^{\text{étape } t+k-1}$.

Stratégie :

- Regrouper les matrices P pour faire des **gros produits**
- Travailler seulement avec $e(X)$ et δ .
- Calculer seulement $f(X) \times P^{\text{étape } 1} \times \dots \times P^{\text{fin}}$ **à la fin**.

Pour calculer $P^{(a)} \times \dots \times P^{(a+b-1)}$ à partir de $e(X)^{(a)} \bmod X^b$ et $\delta^{(a)}$:

- Calculer récursivement $P^{(a)} \times \dots \times P^{(a+\frac{b}{2}-1)}$
à partir de $e(X)^{(a)} \bmod X^{\frac{b}{2}}$ et $\delta^{(a)}$.

Une structure récursive

Point-clé : Connaître k coefficients de $e(X)$ permet d'avancer de k étapes.
c'est-à-dire : calculer $P^{\text{étape } t}, \dots, P^{\text{étape } t+k-1}$.

Stratégie :

- Regrouper les matrices P pour faire des **gros produits**
- Travailler seulement avec $e(X)$ et δ .
- Calculer seulement $f(X) \times P^{\text{étape } 1} \times \dots \times P^{\text{fin}}$ **à la fin**.

Pour calculer $P^{(a)} \times \dots \times P^{(a+b-1)}$ à partir de $e(X)^{(a)} \bmod X^b$ et $\delta^{(a)}$:

- Calculer récursivement $P^{(a)} \times \dots \times P^{(a+\frac{b}{2}-1)}$
à partir de $e(X)^{(a)} \bmod X^{\frac{b}{2}}$ et $\delta^{(a)}$.
- Dédire $e(X)^{(a+\frac{b}{2})} \bmod X^{\frac{b}{2}}$ et $\delta^{(a+\frac{b}{2})}$.

Une structure récursive

Point-clé : Connaître k coefficients de $e(X)$ permet d'avancer de k étapes.
c'est-à-dire : calculer $P^{\text{étape } t}, \dots, P^{\text{étape } t+k-1}$.

Stratégie :

- Regrouper les matrices P pour faire des **gros produits**
- Travailler seulement avec $e(X)$ et δ .
- Calculer seulement $f(X) \times P^{\text{étape } 1} \times \dots \times P^{\text{fin}}$ **à la fin**.

Pour calculer $P^{(a)} \times \dots \times P^{(a+b-1)}$ à partir de $e(X)^{(a)} \bmod X^b$ et $\delta^{(a)}$:

- Calculer récursivement $P^{(a)} \times \dots \times P^{(a+\frac{b}{2}-1)}$
à partir de $e(X)^{(a)} \bmod X^{\frac{b}{2}}$ et $\delta^{(a)}$.
- Dédire $e(X)^{(a+\frac{b}{2})} \bmod X^{\frac{b}{2}}$ et $\delta^{(a+\frac{b}{2})}$.
- Faire l'autre moitié.

Point-clé : Connaître k coefficients de $e(X)$ permet d'avancer de k étapes.
c'est-à-dire : calculer $P^{\text{étape } t}, \dots, P^{\text{étape } t+k-1}$.

Stratégie :

- Regrouper les matrices P pour faire des **gros produits**
- Travailler seulement avec $e(X)$ et δ .
- Calculer seulement $f(X) \times P^{\text{étape } 1} \times \dots \times P^{\text{fin}}$ **à la fin**.

Pour calculer $P^{(a)} \times \dots \times P^{(a+b-1)}$ à partir de $e(X)^{(a)} \bmod X^b$ et $\delta^{(a)}$:

- Calculer récursivement $P^{(a)} \times \dots \times P^{(a+\frac{b}{2}-1)}$
à partir de $e(X)^{(a)} \bmod X^{\frac{b}{2}}$ et $\delta^{(a)}$.
- Dédire $e(X)^{(a+\frac{b}{2})} \bmod X^{\frac{b}{2}}$ et $\delta^{(a+\frac{b}{2})}$.
- Faire l'autre moitié.
- Multiplier $P^{(a)} \times \dots \times P^{(a+\frac{b}{2}-1)}$ et $P^{(a+\frac{b}{2})} \times \dots \times P^{(a+b-1)}$.

Gain de l'approche récursive

- Comme on fait désormais des **gros** produits de matrices, on peut utiliser des algorithmes asymptotiquement rapides (FFT).

$$\text{Complexité } O(N^2) \rightarrow O(N \log^2 N)$$

- Les produits de matrices bénéficient de la FFT :

$$(m+n)^3 N^2 \longrightarrow \begin{cases} (m+n)^2 N \log^2 N \\ + (m+n)^3 N \log N \end{cases}$$

- Le problème est découpé en de nombreux petits sous-problèmes : avantageux pour l'implantation (cache, etc).
- Les tout petits sous-problèmes sont traités par la version quadratique. (phénomène de *Cross-Over*)
- On va plus vite dans cette phase séquentielle \Rightarrow on peut distribuer plus amplement les autres phases.

5. Paramètres de bloc optimaux

On se place sur \mathbb{F}_q (sur \mathbb{F}_2 , c'est différent).

On veut minimiser le temps de calcul.

C'est raisonnable car le temps de calcul correspond bien à la théorie.

Il y a trois étapes :

- BW 1 : Calcul des a_i : $\gamma \frac{m+n}{mn} N^2$, avec n ordinateurs.
- BW 2 : Calcul de F : $c \frac{(m+n)^3}{mn} N \log^2 N$.
- BW 3 : Calcul final : $\gamma \frac{1}{n} N^2$, avec n ordinateurs.

Il faut jouer sur m et n pour minimiser le total.

Paramètres de bloc optimaux

BW 1 : $\gamma \frac{m+n}{mn} N^2$, avec n ordinateurs.

BW 2 : $c \frac{(m+n)^3}{mn} N \log^2 N$.

BW 3 : $\gamma \frac{1}{n} N^2$, avec n ordinateurs.

En pratique, le nombre maximal n_0 de CPUs disponibles est contraint.

Si on minimise :

- Avoir $n \geq n_0$ n'est pas rentable.
- Notre approche est plus efficace que BW classique
- Dans notre cas, BW classique était à peine meilleur que Wiedemann !
- Optimum théorique ($n = c' \sqrt{\frac{N}{\log^2 N}}$) pas très réaliste : mieux vaut utiliser les temps de calcul réels.

Paramètres de bloc optimaux sur \mathbb{F}_2

Qu'est-ce que ça donne sur \mathbb{F}_2 (avec un seul ordinateur) ?

$$\text{BW 1 : } \gamma \frac{m+n}{m} N^2 \frac{\lceil \frac{n}{32} \rceil}{n}$$

$$\text{BW 2 : } c \frac{(m+n)^3}{mn} N \log^2 N.$$

$$\text{BW 3 : } \gamma N^2 \frac{\lceil \frac{n}{32} \rceil}{n}.$$

Où on a supposé que 32 était la taille de mot.

Le calcul d'optimum est plus compliqué.

- Intuitivement, on choisit n multiple de 32.
- Les effets du matériel et les hacks divers font que rien ne vaut l'expérimentation !

- M : matrice $766\,150 \times 766\,150$, définie sur $\mathbb{Z}/2^{607} - 1$.
- 50 000 000 coefficients non nuls (0.008%).
- Première phase d'**élimination structurée** $\Rightarrow 484\,604 \times 484\,604$, pas plus de coefficients (~ 100 /ligne).
- Block Wiedemann avec $m = 8$, $n = 8$, matrice de 400 Mo.

Calcul en plusieurs étapes. On commence par distribuer le travail à faire à 8 machines : le calcul des colonnes de $A(X) = \sum_i x^T M^i y X^i$.

Pour 8 machines : 122 000 produits matrice (400Mo) \times vecteur (40Mo) à faire.

La machine i effectue :

- $v \leftarrow$ colonne i de y (matrice initiale).
- Pour $k = 0, \dots, 122\,000$: $v \leftarrow Mv$, $x^T v \rightarrow$ colonne i de a_k .
- Bien sûr, les colonnes sont rassemblées seulement à la fin.

Pour accélérer le produit **matrice \times vecteur** :

- On utilise des machines **multiprocesseurs** (SMP) :
- Chacun des p CPUs de la machine “gère” $\frac{1}{p}$ ème des **lignes** de M .
- Cela nécessite de bien **équilibrer** les lignes.
- Implantation avec des **threads POSIX**.

Pour 8 machines : 122 000 produits matrice (400Mo) \times vecteur (40Mo) à faire.

La machine i effectue :

- $v \leftarrow$ colonne i de y (matrice initiale).
- Pour $k = 0, \dots, 122\,000$: $v \leftarrow Mv, x^T v \rightarrow$ colonne i de a_k .
- Bien sûr, les colonnes sont rassemblées seulement à la fin.

Pour accélérer le produit matrice \times vecteur :

- On utilise des machines multiprocesseurs (SMP) :
- Chacun des p CPUs de la machine “gère” $\frac{1}{p}$ ème des lignes de M .
- Cela nécessite de bien équilibrer les lignes.
- Implantation avec des threads POSIX.

Et si une erreur vient se glisser dans $v \leftarrow Mv$?

Détection d'erreurs de calcul

Une erreur dans $v \leftarrow Mv$?

$M^{i+1}v$ est calculé à partir de $M^i v$

$M^{i+2}v$ est calculé à partir de $M^{i+1}v$

$M^{i+3}v$ est calculé à partir de $M^{i+2}v$

...

Si jamais **un bit** du vecteur v est erroné, **tout le calcul** est perdu !

Or : 122 000 produits \rightarrow 60 To de données lues.

C'est censé passer, mais **pas toujours**.

Détection d'erreurs : On construit un vecteur m tel que m et $M^T m$ ont des coefficients **petits** et **non nuls**.

On vérifie à chaque étape que $(m|Mv) = (M^T m|v)$. **Très utile !**

Calcul de ces 8 fois 122 000 produits matrice (400Mo) \times vecteur (40Mo) :

600 Mo de RAM utilisés, + grosse pression sur le matériel :

- Attention aux **crashes**.
- Attention aux **erreurs** ! (bon diagnostic pour barettes mémoire)
- Il faut pouvoir reprendre un job stoppé.

Calcul de $A(X)$: **12 jours...**

sur un cluster de 6 alphas 4×667 MHz (vedia.idris.fr)

Calcul de ces 8 fois 122 000 produits matrice (400Mo) \times vecteur (40Mo) :

600 Mo de RAM utilisés, + grosse pression sur le matériel :

- Attention aux **crashes**.
- Attention aux **erreurs** ! (bon diagnostic pour barettes mémoire)
- Il faut pouvoir reprendre un job stoppé.

Calcul de $A(X)$: 12 jours...

sur un cluster de 6 alphas 4×667 MHz (`vedia.idris.fr`)

Unique usage de ce cluster de janvier à mai 2002 !

Une fois $A(X)$ calculé, il faut le générateur.

- $A(X)$: 500 Mo
- Plus gros produit effectué : $2.5 \text{ Go} \times 5 \text{ Go} \rightarrow 2.5 \text{ Go}$ (taille des DFTs).
- 8 niveaux de récursion, 12.5% du temps chacun.
Cross-over avec l'algo quadratique : degré 600 (contre 122 000).
- **2 jours** de calcul sur une alpha à 667 MHz (1 CPU), contre 3 mois avec l'algorithme quadratique.

La dernière étape est semblable à la première (**6 jours**).

Temps total : 3 semaines, contre : 3 mois sans notre approche FFT.
4 mois en "non-bloc".

Nous avons présenté comment l'algorithme de Wiedemann par blocs peut être rendu plus compétitif pour de **très gros systèmes linéaires** définis sur un **gros corps fini**.

- Système linéaire $766\,150 \times 766\,150$ sur $\mathbb{Z}/2^{607} - 1 \Rightarrow$ C'est un record.
- On a largement utilisé la possibilité de **distribuer** et **paralléliser** simultanément.
- Notre **version FFT** du Berlekamp-Massey matriciel a grandement contribué.
- Dans l'ensemble, c'est une épreuve pour le matériel.