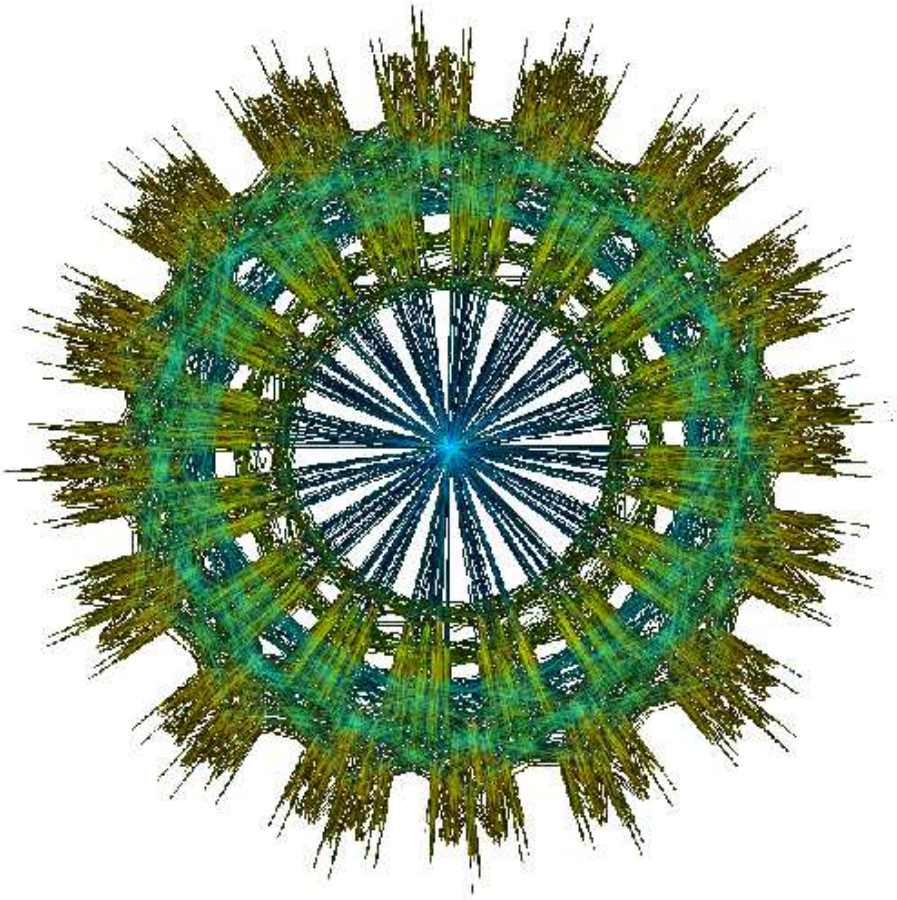
Contributions au calcul exact intensif

Jean-Guillaume Dumas

Contributions au calcul exact intensif

Jean-Guillaume Dumas

20 juillet 2010



Matrice 1280×2800 de co-homologie de variétés toriques par Matthias Franz (Universität Konstanz), image du graphe d'incidence créée par Yifan Hu (AT&T).

Table des matières

1 Introduction

Notre travail s'effectue dans le cadre du calcul formel et plus précisément autour de l'algorithmique exacte efficace. Il s'agit de produire du logiciel de calcul sur des domaines numériques mais sans approximation. Les domaines considérés sont principalement les corps et anneaux finis, les entiers et rationnels à précision fixée ou arbitraire, les polynômes à coefficients exacts. Les algorithmes sont le plus souvent des algorithmes d'algèbre linéaire ou de théorie algorithmique des nombres.

Largement inspirée du développement de la bibliothèque LINBOX, notre approche consiste à produire des logiciels génériques les plus efficaces possibles. En effet, deux approches extrêmes sont les plus fréquentes : développer un module spécifique à une application ou développer un système de calcul formel. Nous nous situons entre ces deux approches, au niveau parfois appelé intergiciel. Notre but est de fournir des routines quasiment aussi efficaces que des routines dédiées, mais avec un plus large spectre d'applications. En ce sens, les bibliothèques GIVARO et LINBOX, tout en étant des outils de recherche permettant de développer rapidement des solutions pour de nombreuses applications, sont également les noyaux efficaces et stables d'un système de calcul comme SAGE.

Au niveau logiciel, la difficulté est de combiner efficacité pérenne, généricité et interfaçage. Nous présentons des techniques et des modèles pour le couplage d'algorithmes adaptatifs, des patrons de conception génériques pour le calcul algébrique, des structures de données spécifiques. Par ailleurs, dans une bibliothèque générique, la forme des problèmes n'est pas connue. Il s'agit alors de fournir des algorithmes pouvant s'adapter aux données et aux ressources. Nous présentons un cadre générique pour l'adaptativité aussi bien séquentielle que parallèle d'algorithmes de calcul formel.

Au niveau algorithmique, la difficulté est d'identifier les domaines d'applicabilité des différentes solutions existantes ou d'en créer de nouvelles. L'analyse de complexité joue alors un rôle prépondérant pour prévoir le comportement des différentes instances. Si parmi les nouveaux algorithmes proposés, certains améliorent l'exposant de complexité, une particularité de notre approche est de s'intéresser spécialement aux constantes et facteurs logarithmiques pour proposer plus généralement des algorithmes de meilleur *terme dominant* de complexité.

2 Arithmétique

Nous nous intéressons dans cette partie aux arithmétiques qui ne sont pas fournies directement par le matériel : corps et éventuellement anneaux finis, en nombres entiers ou rationnels avec ou sans précision fixée, polynômes, courbes etc. Notre travail dans ce domaine est principalement centré sur les corps finis. La bibliothèque GIVARO[†] propose des implémentations variées de corps et anneaux finis de la taille du mot machine. En effet, outre les applications directes sur ces corps, une grande partie des algorithmes efficaces de calcul en plus grande taille s'effectue par remontées des restes chinois ou de Hensel. Au niveau algorithmique, au moins polynomiale et matricielle, dès que la complexité du problème dépasse la complexité de l'arithmétique dans la taille considérée, les remontées seront quasi systématiquement utilisées. Au niveau logiciel, cela donne un grain de calcul adapté aux hiérarchies mémoire des machines actuelles qui permet de définir des noyaux de calcul approchant la puissance de crête.

Notre domaine d'application principal est le calcul formel avec les calculs sur les polynômes et les matrices. La cryptologie et les codes correcteurs sont d'autres domaines naturels d'application de l'arithmétique exacte. Nous avons utilisé des techniques similaires pour l'arithmétique des courbes utilisées en cryptographie, pour attaquer des implémentations embarquées du système RSA et pour développer des permutations et des générateurs pseudo-aléatoires utilisés par exemple dans les systèmes de chiffrement par flots.

En particulier, nous avons proposé plusieurs implémentations de corps et anneaux finis dans GIVARO. La difficulté majeure ici est de pallier les déficiences des processeurs au niveau de la vitesse de la division. Nous avons donc proposé des algorithmes pour effectuer la division euclidienne avec des routines numériques et des méthodes générales pour retarder ou factoriser ces divisions que nous avons appliquées principalement à la multiplication de polynômes ou de matrices. Une autre technique repose sur l'utilisation de générateurs du groupe des inversibles, les racines primitives. Nous avons également proposé un nouvel algorithme probabiliste pour le calcul effectif de racines primitives de grande taille avec application à la construction de générateurs pseudo-aléatoires et à un nouveau test de primalité probabiliste. Les techniques développées sur les racines primitives ont également permis de caractériser les permutations spirales, générateurs pseudo-aléatoires de période maximale. Par ailleurs, nous avons construit une bibliothèque d'entiers en précision fixée, PALOALTO [P42], pour l'arithmétique des courbes ou les attaques de systèmes algébriques de type RSA. Sur ce point, nous sommes partenaires du projet SHIVA [P41], labellisé Minalogic à Grenoble, entre le CEA, plusieurs industriels (CS, Netheos, iWall/Mataru, EasyiiC) et plusieurs laboratoires de recherche grenoblois (LJK, IF, LIG, Verimag). Ce projet devra fournir un module matériel programmable et reconfigurable, avec un haut niveau de sécurité évaluable au sens des critères communs, et s'intégrant sur des plates-formes d'infrastructure réseau à haut débit. Il offrira aux entreprises, aux institutions et aux opérateurs la possibilité de sécuriser leur réseau, par application de leur propre chiffre symétrique, soit choisi ou spécialisé parmi des standards génériques, soit personnalisé. Nous sommes également partenaires de la société CS (Communications et Systèmes), au sein du contrat industriel EAU [P43], pour des formations à la cryptologie et à la sécurité et la mise en place d'infrastructures sécurisées.

De manière générale, notre travail a deux composantes : l'amélioration des algorithmes d'arithmétique et l'utilisation de techniques de calcul formel pour des applications, notamment en cryptologie.

[†] http://ljk.imag.fr/CASYS/LOGICIELS/givaro

Plusieurs développements en cours ou perspectives sont donnés en fins de sections suivantes.

2.1 Petits corps et anneaux finis

Les résultats obtenus ont été publiés dans [B26, A4] pour détailler les choix de représentations de la bibliothèque GIVARO. Les résultats sont des modules de GIVARO pour différentes représentations, selon les besoins. Tout d'abord des représentations pour les corps premiers : représentation classique utilisant la division entière machine avec des entiers entre 0 et $p-1$ ou entre $(1-p)/2$ et $(p-1)/2$ pour une représentation centrée ; représentation de Montgomery ; représentation à inverse numérique : représentation à logarithme discret (et logarithme de Zech) ; représentations compressées, voir section 2.1.3 et [A12, A2].

Ensuite, des représentations pour les extensions de corps finis, $\mathtt{GF}(p^k)$, sont en général implémentées comme des polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ modulo un polynôme irréductible de degré k. Une alternative pour des petites extensions est de tabuler les entrées et d'utiliser les logarithmes de Zech. Comme pour les corps premiers, plusieurs représentations peuvent être utilisées pour réduire le coût de la phase de réduction modulo le polynôme irréductible : réduction de Montgomery ; logarithme discret (comme le groupe des inversibles reste cyclique monogène, il est toujours possible d'utiliser la représentation par l'exposant d'une racine primitive, cette fois polynomiale) ; substitution de Kronecker et représentation compressée. Par exemple, les modules de GIVARO sont utilisés par SAGE[‡] pour son arithmétique des corps finis.

Enfin, avec P. Vignard [202 - Vignard (2003)], nous avons également proposé des implémentations spécifiques pour les anneaux $\mathbb{Z}/2^{32}\mathbb{Z}$, $\mathbb{Z}/2^{64}\mathbb{Z}$, $\mathbb{Z}/2^{128}\mathbb{Z}$, $\mathbb{Z}/2^{32}-1\mathbb{Z}$ et $\mathbb{Z}/2^{64}-1\mathbb{Z}$. L'astuce principale permettant d'obtenir de bonnes performances dans les anneaux de type 2^{2^k} est une méthode d'inversion récursive par carrés : $x \in \mathbb{Z}/2^{2^n}\mathbb{Z}$ est inversible si et seulement si il est premier avec 2^{2^n}, et donc, si et seulement si il est impair. Par conséquent, on peut écrire $x - v * 2 = 1$ (v peut être obtenu facilement en décalant x d'un bit vers la droite). L'intérêt de cette écriture est le suivant: si on a pour un certain k

$$u * x + v * 2^{2^k} = 1$$

alors, en élevant au carré les deux membres de cette égalité, on obtient

$$(u^2 * x + 2^{2^k+1}uv) * x + v^2 * 2^{2^{k+1}} = 1$$

c'est à dire

$$u_1 * x + v_1 * 2^{2^{k+1}} = 1.$$

On obtient donc l'inverse de x par exemple en 5 itérations si $n = 32$ (resp. 6 si $n = 64$) au lieu d'environ 32 (resp. 64) avec l'algorithme de pgcd classique. On retrouve ainsi, plus simplement, le résultat de convergence que l'on obtient avec une itération de Newton-Raphson appliquée à $f(u) = x \cdot u - 1$.

Dans le cadre des corps et anneaux finis, nous avons par ailleurs traité la possibilité d'utiliser des routines numériques (section 2.1.1), de retarder au maximum le calcul du reste modulo (section 2.1.2), de compresser plusieurs éléments d'un corps dans un seul mot machine. Par ailleurs, les applications à la multiplication de polynômes et l'arithmétique des extensions de corps finis sont abordées sections 2.1.3, 2.1.4 et 2.1.5.

Un certain nombre de points reste à développer, par exemple

[‡] http://www.sagemath.org

- Développer une arithmétique compressée indépendante de l'algorithme de calcul. Nous évoquons quelques pistes permettant notamment d'adapter nos techniques à la DFT.
- Plus généralement, factoriser le code de façon que des structures de données compressées ou non puissent être utilisées de manière transparente par des algorithmes génériques par rapport au corps de base.
- Appliquer des techniques compressées à l'arithmétique des courbes.

2.1.1 Division euclidienne par des routines numériques

Pour implémenter les corps finis, soit avec une représentation flottante, soit avec une représentation compressée (voir section 2.1.3), il est possible de précalculer un inverse numérique du modulo pour remplacer la division machine par une multiplication machine par cet inverse.

Maintenant, si r et p sont entiers et que nous voulons calculer r/p par une partie entière de l'arrondi de la multiplication par un arrondi de l'inverse, le résultat n'est pas toujours celui attendu (voir [159 - Lefèvre (2005)] pour plus de détails).

Une façon de pallier ce problème est d'ajouter des tests et des corrections pour les cas erronés, comme dans NTL [190 - Shoup (2005), Théorème 1.5].

Nous proposons ici des bornes sur r pour lesquelles le résultat est garanti correct, tenant compte du mode d'arrondi choisi. En dehors de ces bornes, nous montrons qu'il est possible de garantir une différence de au plus 1 après le calcul de partie entière, pour certains r, p et modes d'arrondi. Dans [191 - Shoup (2009)] par exemple, ces cas sont détectés par 2 tests sur le résidu obtenu (inférieur à 0 ou supérieur à p). Nous avons montré qu'un seul de ces tests est obligatoire si les modes d'arrondi peuvent être mélangés. L'inverse du modulo peut être précalculé pour chaque mode d'arrondi, ce qui évite les changements de mode coûteux en cours de calcul.

En pratique, le gain peut être important si un pipeline est préservé par exemple grâce à cette technique.

Par ailleurs, calculer les meilleures bornes possibles permet de tirer le meilleur parti de la réduction retardée dans les calculs modulaires.

Les résultats de cette section sont tirés de [A2].

Nous notons $r = kp + u$ la division euclidienne avec $0 \leq r \leq 2^\beta - 1$ et nous nous intéressons aux conditions sur r et p pour que l'algorithme 1 retourne le quotient k, suivant les modes d'arrondi $\circ_1$ et $\circ_2$.

Algorithme 1 FDIV

Entrées : – Un entier r vérifiant $0 \leq r \leq 2^\beta - 1$;
– Un entier p vérifiant $1 \leq p \leq 2^\beta - 1$;
– Deux choix de modes d'arrondi $\circ_1$ et $\circ_2$.
Sorties : – $\lfloor \frac{r}{p} \rfloor$.
1 : $invp \leftarrow \circ_1(1/p)$
2 : $x \leftarrow \circ_2(r \cdot invp)$
3 : Retourner $\lfloor x \rfloor$.

Nos résultats sont résumés dans la table 1.

La colonne "intervalle"donne les bornes garanties sur le résultat. Cela montre quels tests et corrections sont nécessaires pour calculer la valeur correcte du quotient. Ces

intervalles sont optimaux dans le sens où nous avons des exemples pour lesquels $\lfloor x \rfloor \neq k$ dans chaque direction possible.

La colonne "Borne sur r"donne des bornes supérieures strictes sur r pour lesquelles $\lfloor x \rfloor \leq k$, dans les cas où le résultat pourrait effectivement dépasser. Nous ne savons pas si ces bornes sont optimales ; dans certains cas nous avons pu trouver des familles d'exemples systématiques, indexées par β, qui atteignent ces bornes asymptotiquement ; d'autres bornes ont pu être approchées par des recherches exhaustives sur des petits β. Nous conjecturons que toutes ces bornes, sauf le cas 2, sont en fait optimales. Pour le cas 2, une borne plus proche de $\frac{3}{8}$ (au lieu de $\frac{1}{3}$) devrait pouvoir être établie.

La colonne "Bits perdus"donne une version simplifiée de cette dernière borne : si r tient dans β moins ce nombre de bits alors $\lfloor x \rfloor \leq k$.

Cas	$\circ_1$	$\circ_2$	Intervalle	Borne sur r	Bits perdus
1	$\bigtriangleup(\cdot)$	$\bigtriangleup(\cdot)$	$k \leq \lfloor x \rfloor \leq k+1$	$2^\beta/(4+2^{2-\beta})$	3
2	$\bigtriangleup(\cdot)$	$\diamond(\cdot)$	$k \leq \lfloor x \rfloor \leq k+1$	$2^\beta/(3+2^{1-\beta})$	2
3	$\bigtriangleup(\cdot)$	$\bigtriangledown(\cdot)$	$k \leq \lfloor x \rfloor \leq k+1$	$2^\beta/2$	1
4	$\diamond(\cdot)$	$\bigtriangleup(\cdot)$	$k \leq \lfloor x \rfloor \leq k+1$	$2^\beta/(3+2^{1-\beta})$	2
5	$\diamond(\cdot)$	$\diamond(\cdot)$	$k-1 \leq \lfloor x \rfloor \leq k+1$	$2^\beta/(2+2^{-\beta})$	2
6	$\diamond(\cdot)$	$\bigtriangledown(\cdot)$	$k-1 \leq \lfloor x \rfloor \leq k$	–	0
7	$\bigtriangledown(\cdot)$	$\bigtriangleup(\cdot)$	$k-1 \leq \lfloor x \rfloor \leq k+1$	$2^\beta/2$	1
8	$\bigtriangledown(\cdot)$	$\diamond(\cdot)$	$k-1 \leq \lfloor x \rfloor \leq k$	–	0
9	$\bigtriangledown(\cdot)$	$\bigtriangledown(\cdot)$	$k-1 \leq \lfloor x \rfloor \leq k$	–	0

Table 1: [A2] Valeurs possibles de $\lfloor x \rfloor$ et bornes sur r pour lesquelles $\lfloor x \rfloor \leq k$ (les modes d'arrondi sont : vers $+\infty$, $\bigtriangleup(\cdot)$; vers $-\infty$, $\bigtriangledown(\cdot)$; au plus proche, $\diamond(\cdot)$).

Pour utiliser les résultats de la table 1 dans un programme, on précalcule $1/p$ dans plusieurs modes d'arrondi et on utilise la meilleure version dans la multiplication, en fonction du mode d'arrondi courant $\circ(\cdot)$. La stratégie consiste à être sûr que $\lfloor x \rfloor \geq k$ après la multiplication de sorte qu'*un seul* test au plus soit nécessaire pour la correction. En outre, nous choisissons la version qui maximise la borne B.

Il est à noter qu'il n'y a pas de stratégie dans le choix de $\circ_1(\cdot)$ qui garantisse que $\lfloor x \rfloor \leq k$ pour tout choix de $\circ_2(\cdot)$ (voir par exemple avec $\circ_2(\cdot) = \bigtriangleup(\cdot)$). Cela induit que notre stratégie est la seule qui minimise le nombre de tests nécessaires à la correction.

Dans une utilisation typique de l'algorithme 1 quand r est une accumulation de plusieurs produits modulo p (voir section suivante), la borne B est interprétée comme le nombre d'opérations qui peuvent être effectuées avant qu'une réduction ou une correction devienne obligatoire. En particulier, si B n'est jamais dépassée, la correction n'est pas nécessaire.

2.1.2 Produit scalaire retardé

L'opération de base en algèbre est souvent la succession de deux opérations, une multiplication et une addition. On appelle *AXPY* (ou "fused-mac", FMA) cette opération. Dans de nombreux algorithmes, il est même nécessaire d'enchaîner plusieurs de ces opérations : produit scalaire, produit de polynômes, étape d'élimination dans l'algorithme de Gauß, etc.

Il est donc crucial d'optimiser la représentation pour l'enchaînement d'`AXPY`. Nous allons par exemple effectuer les multiplications et les additions successivement, *sans faire la réduction modulo*, et faire cette dernière une seule fois pour tout un bloc

d'opérations arithmétiques. Ainsi le coût de la réduction est amorti par la quantité de calculs effectués.

Par exemple, pour une représentation sur un intervalle $[0, p-1]$, il est possible de calculer λ accumulations sans division si

$$\lambda(p-1)^2 < 2^\gamma \leq (1+\lambda)(p-1)^2. \tag{1}$$

Notons qu'avec une arithmétique signée, une représentation centrée peut être utilisée (i.e. $-\frac{p-1}{2} \leq x \leq \frac{p-1}{2}$ pour le stockage d'un élément x dans un corps premier impair) et l'équation 1 devient

$$\lambda\left(\frac{p-1}{2}\right)^2 < 2^{\gamma-1} \leq (1+\lambda)\left(\frac{p-1}{2}\right)^2 \tag{2}$$

ce qui améliore les performances d'un facteur 2.

Dans un produit scalaire de dimension n, une approche retardée permet de n'effectuer que $\lceil \frac{n}{\lambda} \rceil$ divisions qui sont donc amorties par les n multiplications et additions.

Sur des architectures récentes, et bien qu'ayant une mantisse accessible aux calculs exacts plus petite (53 bits au lieu de 64 par exemple), les représentations utilisant les flottants sont souvent les plus performantes : pour des petits nombres premiers le retard permet d'amortir parfaitement les divisions ; au contraire, pour des nombres plus grands, le nombre de divisions machine s'approche du nombre de calculs et les mauvaises performances de la division machine prédominent [B26].

2.1.3 REDQ : réduction modulaire simultanée

Une autre routine importante sur les corps finis est la multiplication de polynômes. Par exemple, dans des extensions de corps finis, une alternative à la représentation par générateurs est d'utiliser la substitution de Kronecker. Si le corps est petit, il suffit de peu de bits pour stocker ses éléments. Il est alors possible de compresser plusieurs d'entre eux dans un seul mot machine. Pour des éléments polynomiaux, il faut donc transformer un polynôme en une représentation binaire. En caractéristique 2, c'est naturel : un bit par coefficient ; pour d'autres caractéristiques, on peut remplacer l'indéterminée par un entier plus grand que la caractéristique du corps. Le principe de cette substitution de Kronecker est simple : un polynôme $\overline{\alpha_0} + \cdots + \overline{\alpha_k} X^k$ dans $\mathbb{Z}/p\mathbb{Z}[X]$ est représenté par la valeur $\alpha_0 + \cdots + \alpha_k q^k$ pour un entier donné q, tel que les α_i vérifient $0 \leq \alpha_i < p$ et $\alpha_i \equiv \overline{\alpha_i} \bmod p$, pour $i = 0, \ldots, k$. Pour obtenir cette représentation, il suffit donc d'évaluer le polynôme pour un entier donné. L'entier q est en général une puissance de 2 pour la simplicité et la rapidité d'utilisation des décalages binaires. La motivation principale de cette substitution est son utilisation pour la multiplication. Il faut simplement choisir une substitution suffisamment grande pour que les coefficients intermédiaires de la multiplication ne dépassent pas q. Si en outre le degré du produit n'est pas trop grand alors le produit peut même tenir dans un mot machine. Dans le cas contraire, nous verrons qu'il est parfois avantageux d'utiliser une arithmétique à plus grande précision.

Comme dans la section précédente, le coût de calcul est dominé par les réductions modulaires. Nous proposons plusieurs façons de réduire ce coût : en retardant la réduction, en réduisant simultanément plusieurs coefficients et en tabulant certaines parties.

La technique est donc celle de la section 2.1.2 : il faut effectuer plusieurs multiplications de polynômes par exemple dans la représentation entière, convertir en base q et réduire modulo p après plusieurs opérations seulement. On appellera alors DQT

pour (Discrete Q-adic Transform) cette substitution de Kronecker, combinée à une réduction retardée. Plus précisément, la DQT est l'évaluation d'un polynôme modulo p en un entier q suffisamment grand. On appelle alors DQT inverse le retour à la base q suivi de la réduction modulo p.

Un coût important dans la transformée inverse est la réduction modulaire de chacun des coefficients du polynôme, et en particulier des divisions machines que cela implique.

En caractéristique 2, ces divisions sont effectuées simultanément par un ET logique. Modulo 3, 4, 5, 7 et 8 [89 - Boothby et Bradshaw (2009)] ont proposé des méthodes ad hoc pour réaliser cette vectorisation. Différemment, nous avons proposé une méthode générique pour diviser simultanément plusieurs coefficients par un modulo p en une seule division machine, ou même une seule multiplication numérique par l'inverse précalculé. Celle-ci est rappelée dans l'algorithme 2.

Algorithme 2 REDQ

Entrées : – Deux entiers p et q et $\tilde{r} = \sum_{i=0}^{d} \widetilde{\mu_i} q^i \in \mathbb{Z}$, vérifiant les conditions (3).
Sorties : – $\rho \in \mathbb{Z}$, avec $\rho = \sum_{i=0}^{d} \mu_i q^i$ où $\mu_i = \widetilde{\mu_i} \bmod p$.

REDQ COMPRESSION
$s = \left\lfloor \frac{\tilde{r}}{p} \right\rfloor$;
Pour $i = 0$ **jusqu'à** d **Faire**
  $u_i = \left\lfloor \frac{\tilde{r}}{q^i} \right\rfloor - p \left\lfloor \frac{s}{q^i} \right\rfloor$;
Fin Pour

REDQ CORRECTION {quand $p \nmid q$}
$\mu_d = u_d$
Pour $i = 0$ **jusqu'à** $d - 1$ **Faire**
  $\mu_i = u_i - q u_{i+1} \bmod p$;
Fin Pour
Retourner $\rho = \sum_{i=0}^{d} \mu_i q^i$;

Chacune des deux étapes de cet algorithme peut être ensuite améliorée : pour q une puissance de 2, il est possible de combiner les calculs de la boucle de compression pour diminuer de moitié le nombre d'opérations nécessaires ; par ailleurs, le calcul de la correction peut être tabulé puisque les coefficients obtenus après compression sont petits.

Quand q est une puissance de 2 et que les éléments sont stockés dans un type entier, les divisions par q^i et les parties entières sont naturellement des opérations atomiques : décalages/extractions de bits. Nous avons également démontré les bornes suivantes, permettant de choisir q au plus près.

Lemme 3

Soit $\tilde{r}(X)$ l'accumulation de n produits de polynômes de degré $k-1$ à coefficients compris entre 0 et $p-1$. Si q vérifie la condition suivante

$$q > nk(p-1)^2 \quad \textit{et} \quad (2k-1)\log_2(q) \leq \beta, \tag{3}$$

avec β le nombre de bits de mantisse accessibles, alors l'algorithme 2 est correct.

De plus, il se trouve que les k restes obtenus après la phase de compression de REDQ$_k$ peuvent être calculés avec seulement $k/2$ AXPY. En effet, chaque calcul nécessite un même AXPY situé sur la même partie des bits des opérandes, indépendemment des autres $k-1$ calculs. En recopiant donc des sous-ensembles de bits sur un même mot machine, il est possible d'effectuer plusieurs compressions simultanément.

De manière générale, l'algorithme est efficace car il est possible de précalculer $1/p$, $1/q$, $1/q^2$ etc. Le calcul de chaque u_i et μ_i peut aussi être pipeliné ou vectorisé puisqu'ils sont indépendants.

En outre, il est possible de tabuler la multiplication par q avec une table de taille p^j ajustable jusqu'à p^{k+1} qui effectue une REDQ$_k$-CORRECTION avec $\lfloor (k-1)/(j-1) \rfloor$ accès mémoire.

- En pratique, on essaiera de prendre une table la plus grande possible ne pénalisant pas les accès cache, et donc si possible de taille maximale afin d'avoir un seul accès mémoire.
- En résumé, dans ce cas, la transformation REDQ permet de remplacer k divisions machine par 1 seule division et 1 accès mémoire, ou mieux, par 1 seule multiplication par l'inverse et 1 accès mémoire.
- Si p divise q les accès mémoire deviennent inutiles. Un choix de q de la forme $2^k p^j$ pourrait alors encore améliorer les performances.

2.1.4 Substitution de Kronecker et multiplication de polynômes

Dans cette section, nous utilisons la substitution de Kronecker et la réduction simultanée REDQ pour obtenir une multiplication de polynômes rapide sur des petits corps finis.

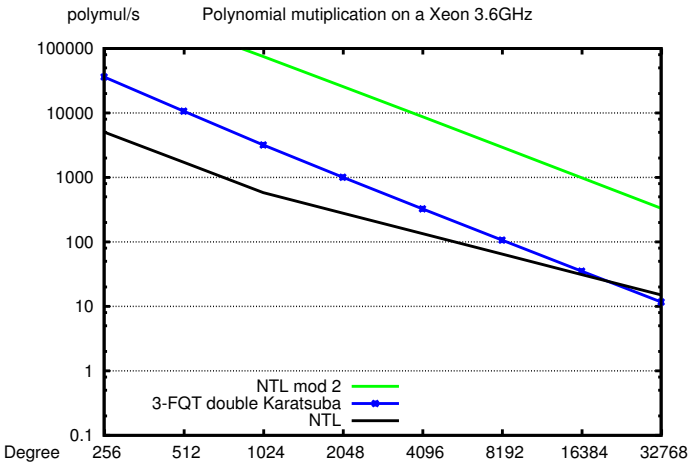


Figure 1: Accélération de la FFT pour la multiplication de polynômes

Dans [A2] nous avons vu que la DQT est plus rapide que NTL tant que le même algorithme est utilisé (classique ou Karatsuba). Cela montre que notre stratégie est

utile pour des petits nombres premiers et des petits degrés : dans le cas d'algorithmes classiques ou sous-quadratiques, l'utilisation de la DQT améliore la vitesse de calcul d'un ordre de grandeur. Toutefois, un changement de pente sur la courbe NTL reflète le passage à la FFT, comme on peut le voir sur la figure 1. Nous n'avons pas implémenté de DQT dans le domaine FFT et donc l'implémentation de NTL devient plus rapide. À noter également que pour le cas particulier $p = 2$, NTL présente une implémentation spécifique (voir par exemple [206 - Weimerskirch et al. (2003)]), toujours un ordre de grandeur plus rapide que notre stratégie. Une idée pour étendre cette stratégie, au moins théoriquement, est de retarder le modulo pour la multiplication par transformée de Fourier.

DFT déroulée Soit ω une racine primitive N^{e} de l'unité.

Pour un vecteur (x_i) de taille N, on note E_j la DFT des éléments d'indice pair et O_j la DFT des éléments d'indice impair et $M = \frac{N}{2}$. Ainsi l'algorithme de DFT rapide ("Decimation in Time") est résumé dans la formule ci-après :

$$\begin{aligned} H_k &= \sum_{m=0}^{M-1} x_{2m}\omega^{(2m)k} + \sum_{m=0}^{M-1} x_{2m+1}\omega^{(2m+1)k} && (4)\\ &= \sum_{m=0}^{M-1} x_{2m}(\omega^2)^{mk} + (\omega)^k \sum_{m=0}^{M-1} x_{2m+1}(\omega^2)^{mk} && (5)\\ &= \begin{cases} E_k + \omega^k O_k & \text{si } k < M \\ E_{k-M} - \omega^{k-M} O_{k-M} & \text{si } k \geq M. \end{cases} && (6) \end{aligned}$$

En considérant qu'il n'y a qu'une réduction modulaire par AXPY, le nombre de réductions de cet algorithme est $R(N) = 2R(N/2) + N$, soit $R(N) = N \log_2 N$.

L'idée générale pour retarder le modulo dans cet algorithme est de dérouler les appels récursifs. Avec les notations EE, OE, EO, OO pour les éléments d'indices respectifs 0, 2, 1 et 3 modulo 4, si $L = \frac{M}{2} = \frac{N}{4}$, on obtient :

$$E_j = \begin{cases} EE_j + \omega^j OE_j & \text{si } j < L \\ EE_{j-L} - \omega^{j-L} OE_{j-L} & \text{si } L \leq j < M. \end{cases} \quad (7)$$

$$O_j = \begin{cases} EO_j + \omega^j OO_j & \text{si } j < L \\ EO_{j-L} - \omega^{j-L} OO_{j-L} & \text{si } L \leq j < M. \end{cases} \quad (8)$$

et donc H_k vaut :

$$\begin{cases} EE_k + \omega^k OE_k + \omega^k EO_k + \omega^{2k} OO_k & \text{si } k < \frac{M}{2} \\ EE_{k-L} + \omega^{k-L} OE_{k-L} + \omega^k EO_{k-L} + \omega^{2k-L} OO_{k-L} & \text{si } \frac{M}{2} \leq k < M \\ EE_{k-M} + \omega^{k-M} OE_{k-M} + \omega^{k-M} EO_{k-M} + & \\ \qquad \omega^{2k-2M} OO_{k-M} & \text{si } M \leq k < \frac{3M}{2} \\ EE_{k-M-L} + \omega^{k-M-L} OE_{k-M-L} + & \\ \qquad \omega^{k-M} EO_{k-M-L} + \omega^{2k-2M-L} OO_{k-M-L} & \text{si } \frac{3M}{2} \leq k < N. \end{cases} \quad (9)$$

En supposant que les ω^j sont précalculés, que toutes les valeurs de DFT de niveau $\frac{N}{4}$ sont prises modulo p en représentation centrée, on voit dans l'expression (9) que

chaque calcul est borné par $\frac{p-1}{2} + 3\left(\frac{p-1}{2}\right)^2$. Plus généralement, pour une mantisse de γ bits, il est possible de dérouler la DFT jusqu'au niveau $\frac{N}{2^i}$ et de ne faire qu'un modulo tous les i appels récursifs dès que

$$\frac{p-1}{2} + \frac{2^i - 1}{4}(p-1)^2 < 2^{\gamma-1} \tag{10}$$

Le nombre total de réductions modulaires devient alors $R_i(N) = 2^i R_i(N/2^i) + N$, soit $R_i(N) = \frac{1}{i} N \log_2 N$, réduisant ainsi le nombre de réductions modulaires (les opérations les plus coûteuses) d'un facteur i.

DFT retardée En terme de nombre de réductions, il est possible d'utiliser une meilleure version de la DFT ("Decimation in Frequency") :

Algorithme 4 Transformée de Fourier Discrète Rapide, DFT_ω.

Entrées : – Un vecteur h dont la taille N est une puissance de 2
Sorties : – Sa transformée le vecteur H
1 : **Si** h est de taille 1 **Alors** retourner h
2 : **Sinon**
3 : $M = N/2$;
4 : Calculer $\tilde{h}^{(p)} = (h_0 + h_M, \ldots, h_{M-1} + h_{N-1})$;
5 : Calculer $\tilde{h}^{(i)} = (h_0 - h_M, (h_1 - h_{M+1})\omega, \ldots, (h_{M-1} - h_{N-1})\omega^{M-1})$;
6 : Calculer récursivement $\tilde{H}^{(p)} = DFT_{\omega^2}(\tilde{h}^{(p)})$ de taille $N/2$;
7 : Calculer récursivement $\tilde{H}^{(i)} = DFT_{\omega^2}(\tilde{h}^{(i)})$ de taille $N/2$;
8 : Les coefficients pairs de H sont les coefficients de $\tilde{H}^{(p)}$ et les coefficients impairs de H sont les coefficients de $\tilde{H}^{(i)}$.
9 : **Fin Si**

Dans cette version, le nombre de réductions vaut seulement la moitié de celles de l'algorithme précédent car les multiplications par ω^j sont déjà factorisées, soit $R(N) = \frac{1}{2} N log_2(N)$.

Une idée est donc d'étendre cette factorisation dans une structure de données. Soit k quelconque vérifiant la modification suivante de l'équation (10) :

$$\frac{k}{4}(p-1)^2 < 2^{\gamma-1} \tag{11}$$

On transforme alors chaque coefficient en entrée en un polynôme en ω comportant toujours *au plus* k *monômes* : $h_i = \sum_{j=1}^{k} a_j \omega^{\alpha_j}$, chaque monôme étant stocké comme un coefficient et un exposant. Les opérations se font alors comme suit :

- Multiplication par ω^j : addition de j à chaque exposant des monômes, modulo N (donc addition simple suivie éventuellement d'une soustraction de N)
- Addition d'un élément : on applique l'algorithme 5 suivant :

Algorithme 5 Somme en place de deux éléments

Entrées :
– ω une racine primitive N^{e} de l'unité modulo p.
– Une table contenant toutes les puissances de ω précalculées modulo p.
– k vérifiant (11).
– $A = \sum_{j=1}^{k_A} a_j \omega^{\alpha_j}$ avec $k_A \leq k$.
– $B = \sum_{j=1}^{k_B} b_j \omega^{\beta_j}$ avec $k_B \leq k$.

Sorties : – Leur somme en place $A \leftarrow A + B$.
1 : **Pour** $j = 1$ **jusqu'à** k_A **Faire**
2 : **Pour** $l = 1$ **jusqu'à** k_B **Faire**
3 : **Si** α_j == β_l **Alors**
4 : $a_j += b_l$;
5 : **Si** $a_j \geq p$ **Alors** $a_j -= p$ **Fin Si** ;
6 : Retirer le monôme $b_l\omega^{\beta^l}$ de B;
7 : **Fin Si**
8 : **Fin Pour**
9 : **Fin Pour**
10 : Renuméroter les monômes restants dans $B = \sum_{j=1}^{k'_B} b_j\omega^{\beta_j}$ avec $k'_B \leq k_B$;
11 : **Si** $k_A + k'_B \leq k$ **Alors**
12 : $A \leftarrow A \cup B$;
13 : **Sinon**
14 : $a_1 += \sum_{l=1}^{k'_B} b_l\omega^{\beta_l-\alpha_1} + \sum_{j=k_A+k'_B-k+1}^{k_A} a_j\omega^{\alpha_j-\alpha_1} \mod p$;
15 : Retirer les monômes $j = k_A + k'_B - k + 1$ **jusqu'à** k_A de A;
16 : **Fin Si**

L'étape qui suit permet de diminuer le nombre total d'additions

17 : Renuméroter les monômes de $A = \sum_{j=1}^{k'_A} a_j\omega^{\alpha_j}$ avec $k'_A \leq k$;
18 : **Si** $k'_A == k$ **Alors**
19 : $a_1 += \sum_{j=1}^{k'_A} a_j\omega^{\alpha_j} \mod p$;
20 : $\alpha_1 = 0$;
21 : Retirer les monômes $j = 2$ **jusqu'à** k'_A de A;
22 : **Fin Si**
23 : Retourner A;

- Pour les deux solutions proposées, le problème sera de concilier les opérations modulaires et l'existence de racines primitives acceptables, ou alors d'utiliser la transformée de Fourier tronquée [200 - van der Hoeven (2004)]. Dans cette première approche, N est borné et donc ces transformations seront sans doute limitées à des petits degrés.

2.1.5 Petites extensions de corps finis

Pour une petite extension de corps fini, on utilise dans les opérations atomiques la représentation des éléments en logarithme discret. Or, il est intéressant, dans le cadre de la multiplication de matrices par exemple, d'utiliser une transformation entre cette représentation et les flottants. On utilise alors plutôt une représentation polynomiale avec substitution de Kronecker et en particulier la réduction modulaire simultanée pour récupérer tous les coefficients du polynôme en sortie des calculs flottants [A12].

Ainsi, les performances de l'algèbre linéaire dans des petites extensions de corps fini approchent celle de l'algèbre linéaire dans des corps premiers de la taille du mot machine.

Par exemple, il est possible de calculer à une vitesse de 7850 millions d'opérations de $\mathtt{GF}(9)$ par seconde, sur un AMD Opteron 2352 Barcelona à 2.1GHz. Cela ne représente plus qu'un surcoût de l'ordre de 4% par rapport à une implémentation directe sur le corps premier $\mathtt{GF}(11)$. La table 2 donne une comparaison avec Magma V2.16-6 sur un cœur d'un Opteron 2.1GHz.

Dimension	255	1000	2000	4000	6000	7000	10000
FFLAS+REDQ	1.81	4.88	**5.96**	**6.90**	**7.13**	**7.70**	**7.85**
magma	**3.62**	**5.61**	5.04	6.10	6.55	6.87	7.08

Table 2: Vitesse, en Mffops, de la multiplication de matrices dans GF(9) vs Magma v2.16-6, sur un opteron 2.1GHz

2.2 Calcul de racines primitives avec probabilité adaptative

Une racine primitive est un générateur du groupe des inversibles d'un corps fini. Le calcul d'un tel générateur est lié à la factorisation de l'ordre du groupe (taille du corps moins 1). Nous présentons dans cette partie un résultat théorique sur la localisation des racines primitives dans un tel groupe. Ce résultat nous permet d'utiliser une factorisation partielle de l'ordre du groupe pour obtenir un calcul effectif, polynomial, d'un probable générateur du groupe : l'effort de factorisation est ainsi lié à la probabilité de bien choisir un générateur. Cet algorithme a de nombreuses applications, notamment pour accélérer les mises en place de systèmes cryptographiques de type El Gamal ou d'échange de clefs Diffie-Hellman, et permet de définir un nouveau test probabiliste de primalité compétitif avec le test de Miller-Rabin. Ces résultats ont été publiés dans [A6, T68].

2.2.1 Localisation des racines primitives

Dans un corps fini de taille q, il y a $\varphi(q-1)$ générateurs, avec φ l'indicatrice d'Euler. Ainsi sauf pour $q = 2$ la proportion de générateurs est toujours plus petite que $1/2$ et très souvent plus grande que $1/5$. Il est donc facile de trouver des générateurs en tirant au hasard des éléments puis en testant si leur ordre est bien maximal, c'est-à-dire égal à $p-1$. Le problème réside dans le calcul de l'ordre. La première idée est de tester toutes les puissances successives de g, jusqu'à trouver la première occurrence de 1. Malheureusement, ce calcul est exponentiel en la taille de q. L'accélération classique consiste à factoriser $q-1$ et à tester seulement si les puissances $g^{\frac{q-1}{p}}$ sont différentes de 1 pour tous les facteurs premiers p de $q-1$. La complexité est alors celle de la factorisation d'entiers, à ce jour non polynomiale.

Par ailleurs, il existe des méthodes polynomiales pour isoler des sous-ensembles de taille polynomiale en la taille de q contenant *au moins* une racine primitive [189 - Shoup (1992), 80 - Bach (1997)] ou encore des méthodes fabriquant des éléments d'ordre exponentiellement grand [120 - Gathen et Shparlinski (1998)]. Notre idée est intermédiaire : trouver un ensemble contenant une proportion aussi grande que l'on veut de générateurs ; ainsi en tirant au hasard dans cet ensemble, on aura toutes les chances de tomber sur un générateur.

Théorème 6

Soient $q \neq 2$ le cardinal d'un corps fini, $C > 1$ un facteur de $q-1$ et $S = \{b \in \mathtt{GF}(q)^*, b^{\frac{q-1}{C}} \not\equiv 1\}$. Dans S, la proportion d'éléments b tels que $b^{\frac{q-1}{C}}$ soit d'ordre maximal C est

$$\frac{\varphi(C)}{C-1}.$$

En théorie des nombres, un nombre entier positif est dit B-friable si tous ses facteurs premiers sont plus petits que B.

En pratique on utilise un algorithme qui effectue seulement une factorisation partielle de $q-1 = kC$ avec $pgcd(k, C) = 1$, en temps proportionnel en la racine carrée de la proportion désirée. k est totalement factorisé et contient les petits facteurs de $q-1$, il est donc facile de trouver des éléments d'ordre k. Au contraire C est potentiellement composé mais ne contient pas de petits facteurs, l'ensemble S associé permet donc avec une bonne probabilité de trouver des éléments d'ordre C. Comme $pgcd(k, C) = 1$, le produit de deux éléments d'ordres respectifs k et C est une racine primitive.

Corollaire 7

L'algorithme [A6, Algorithme 1] est correct et nécessite, avec une arithmétique classique et l'algorithme rho de Pollard pour la factorisation partielle,

$$O\left(\sqrt{\frac{1}{\varepsilon}}\log^{2.5}(q) + \log^3(q)\log(\log(q))\right)$$

opérations arithmétiques en moyenne pour garantir qu'une proportion $1-\epsilon$ *de ses sorties sont des racines primitives.*

Ainsi, si l'on est prêt à se contenter d'une probabilité fixée de se tromper, notre algorithme devient polynômial. Par exemple il est possible de calculer des éléments probablement primitifs pour des tailles jusqu'alors impossible à atteindre, pour une probabilité de se tromper inférieure à une sur 1000 milliards.

Une application de cet algorithme est de fabriquer des générateurs pseudo-aléatoires à la Blum-Micali. Nous avons montré par exemple que notre approche permet d'aller plus loin que les méthodes consistant à fabriquer des nombres premiers p à factorisation de $p-1$ connue [87 - Blum et Micali (1984), 79 - Bach (1988)].

2.2.2 Un nouveau test de primalité probabiliste

Le test de primalité déterministe de Lucas consiste à tenter de fabriquer des racines primitives pour un entier m. En cas de réussite, le nombre est prouvé premier. Au contraire, si m est composé, il sera difficile voire impossible de fabriquer des éléments de certains ordres divisant $m-1$.

Nous proposons ici d'essayer de construire une racine primitive probable à l'aide de l'algorithme de la section précédente. Si m est premier, l'algorithme terminera correctement, sinon la proportion d'éléments de certains ordres divisant $m-1$ sera plus faible. Dans la plupart des cas, sauf exceptions traitées par ailleurs, cette faible proportion est détectée et le nombre est reconnu comme composé.

Cet algorithme peut être vu comme une généralisation de l'algorithme de Miller-Rabin : ce dernier teste uniquement les ordres de la forme $\frac{m-1}{2^e}$, nous testons tous les ordres de la forme $\frac{m-1}{p^e}$ pour tout $p \leq B$, petit facteur premier de $m-1$.

L'idée est que pour obtenir un test de primalité à probabilité de réussite satisfaisante, il faut effectuer plusieurs tests de Miller-Rabin avec des témoins différents, chaque témoin diminuant la probabilité d'erreur d'au moins $1/4$. Dans notre algo-

rithme, un témoin peut être plus long à tester, mais peut diminuer la probabilité d'erreur d'un facteur proche de $1/B$.

La figure 2 montre que cet algorithme est compétitif avec plusieurs appels à la routine Miller-Rabin de GMP.

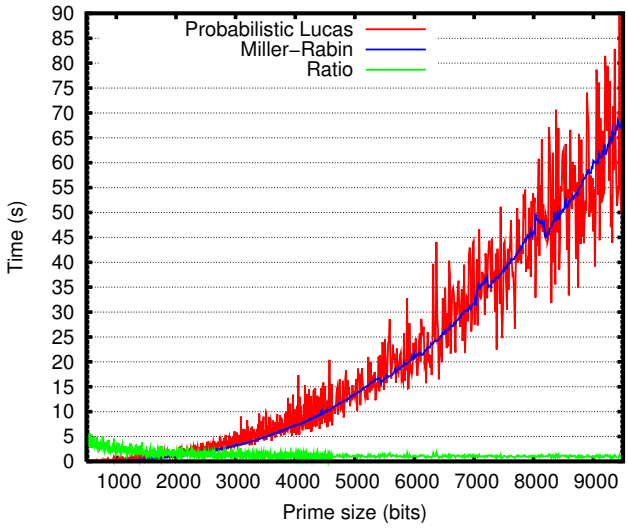


Figure 2: Lucas Probabiliste (GIVARO) vs Miller-Rabin (GMP) avec probabilité de réussite $< 10^{-6}$ sur un PIV 2.4GHz

- Une extension naturelle de cet algorithme doit être envisageable avec des factorisations de type $p+1$ au lieu de $p-1$, et a fortiori avec des factorisations par courbes elliptiques.

2.3 Arithmétique à précision fixée

Pour implémenter l'arithmétique de corps plus grands que la taille du mot machine, plusieurs approches sont possibles. Pour des opérations compatibles avec le modulo, des techniques de restes chinois seront souvent préférables car elles permettent d'obtenir directement une complexité croissant linéairement avec la taille du corps. Au contraire, les techniques de précision arbitraire nécessitent l'introduction d'algorithmique adaptative pour obtenir une complexité quasi-linéaire, asymptotiquement. Par ailleurs, pour des applications en cryptologie ou en codes, la taille du corps ou des entiers considérés est en général fixée. Toutefois, la bibliothèque générique d'entiers à précision arbitraire GMP est souvent plus rapide que les bibliothèques spécialisées pour la cryptologie comme openSSL ou MIRACL. En effet, ces dernières utilisent également une arithmétique à précision arbitraire. Dans le cadre du projet UJF-MSTIC Palo-Alto [P42], nous réalisons donc une bibliothèque d'entiers à précision fixée [S54], i.e. réalisant les opérations modulo 2^s pour s une puissance de 2 dans les tailles, petites pour GMP, mais adaptées à la cryptologie. Pour cela nous avons défini des structures de données récursives où des entiers de taille 2^k, pour $k \geq 7$ peuvent être stockés comme deux entiers de taille 2^{k-1} [T63]. Si $k \leq 6$ nous utilisons l'arithmétique machine. Notre design utilise les template C++ afin de définir des algorithmes génériques doublant la précision (voir par exemple [210 - Yoshino et al. (2009)]) et de les spécialiser pour l'arithmétique en taille de mot machine. Grâce à cette technique et des routines

assembleur de GMP (fichier `longlong.h`) nous avons pu obtenir les performances de la table 3.

Taille	64	128	256	512	1024	2048
RecInt	**310.67**	**123.36**	**41.94**	**9.36**	1.80	0.11
GMP-4.3.1	72.31	56.68	20.97	8.59	**2.54**	**0.79**
openSSL-0.9.8	34.24	32.26	15.20	8.59	1.86	0.52

Table 3: Multiplication d'entiers à précision fixée et structure de données récursive vs GMP/openSSL, icc v11, Xeon X5482, 3.2GHz, en millions d'opérations arithmétiques par seconde

Le gain s'explique d'une part par le fait que nous implémentons seulement la précision fixée et donc, modulo 2^{2^k}, la moitié seulement des bits d'un produit de deux éléments de taille 2^k est calculée. D'autre part, nous évitons d'utiliser des structures de données lourdes dont le coût n'est pas amorti pour les petites tailles.

- La bibliothèque MPFQ[♮] propose des implémentations spécifiques très efficaces pour certains corps particuliers. Nous devons pouvoir coupler et unifier ces différentes approches.
- Enfin, notre module sert de noyau de calcul à une autre bibliothèque en cours de construction pour l'arithmétique des courbes elliptiques et hyper-elliptiques, puis, à terme pour des primitives et protocoles cryptologiques.

2.4 Permutations spirales

Les nombres de Raymond Queneau sont les entiers n pour lesquels la quenine (permutation spirale envoyant tout nombre pair sur sa moitié et tout nombre impair sur son opposé ajouté à n) est d'ordre maximal n. Grâce à des techniques de racines primitives, nous étudions dans cette partie la caractérisation des nombres de Queneau et plus généralement des permutations spirales, ainsi que leur représentation graphique sous forme de spirale [A9, T60].

L'application première des quenines est la poésie des troubadours [184 - Roubaud (2000)]. Toutefois, nous avons montré que ces permutations spirales ont des liens avec les bases normales optimales de $\mathtt{GF}(2^n)$ qui permettent de calculer rapidement dans ces corps finis (voir [75 - Arndt (2010), §40.8.2] ou [170 - Mullin et al. (1989), Théorème 3.2]) ainsi qu'avec des battements de cartes [77 - Asveld (2009), §3], pour par exemple décrire des synchronisations de processus concurrents (voir e.g. [143 - Jantzen (1981)] pour plus de détails), et enfin avec des générateurs congruentiels pseudo-aléatoires de période maximale.

Nous considérons la permutation δ_n définie comme suit :

$$\delta_n(x) = \begin{cases} 2x & \text{si } 2x \leq n \\ 2n+1-2x & \text{sinon} \end{cases}$$

Nous donnons ici une caractérisation complète des quenines (permutations δ_n formant un cycle de longueur n, n est dans ce cas dit **admissible**) définies par Raymond Queneau, puis Jacques Roubaud [184 - Roubaud (2000)].

[♮] `http://mpfq.gforge.inria.fr` [121 - Gaudry et Thomé (2007)].

Théorème 8

$2n+1$ étant premier, soit $\mathbb{Z}/2n+1\mathbb{Z}$ le corps à $2n+1$ éléments, alors n est admissible si et seulement si :

i. $n \equiv 1 \mod 4$ *; +2 est primitif et -2 d'ordre exactement n dans $\mathbb{Z}/2n+1\mathbb{Z}$*

ii. $n \equiv 2 \mod 4$ *; +2 et -2 sont racines primitives dans $\mathbb{Z}/2n+1\mathbb{Z}$*

iii. $n \equiv 3 \mod 4$ *; -2 est primitif et +2 d'ordre exactement n dans $\mathbb{Z}/2n+1\mathbb{Z}$*

On voit que le multiplicateur 2 dans la définition des quenines joue un rôle prépondérant. En particulier, il induit que les quenines sont des spirales comportant deux "rayons" : à partir du centre de la spirale, on trouve deux séquences d'entiers consécutifs. J. Roubaud a ensuite généralisé les quenines en considérant des multiplicateurs différents de deux et nous avons montré que cela correspond à l'obtention d'autant de séquences d'entiers consécutifs dans la représentation spirale. Dans la suite, nous nous intéressons de plus près à ces rayons et en déduisons de nouvelles constructions de spirales.

Nous avons également donné une construction "polynomiale"des g-quenines : en prenant une racine primitive probabiliste de la section 2.2 en $\log(n)$, nous montrons qu'il suffit de $\log^3(n)$ opérations arithmétiques pour trouver tous les rayons de la spirale et leurs orientations.

Algorithme 9 Construction polynomiale d'une quenine

Entrées : – Un entier $n > 0$ tel que $2n+1$ est premier.
Sorties : – g et une g-quenine d'ordre n.
1 : Fabriquer une racine g primitive dans $\mathbb{Z}/2n+1\mathbb{Z}$, de manière probabiliste ;
2 : **Si** $g > n/2$ **Alors** $g := -g$ **Fin Si**
3 : $h \equiv g^{-1} \mod 2n+1$;
4 : **Pour** $k = 1$ **jusqu'à** $g-1$ **Faire**
5 : $r_k = k \cdot h \mod 2n+1$;
6 : **Si** $r_k \leq n$ **Alors**
7 : Le k-ième rayon est entrant et commence à r_k ;
8 : **Sinon**
9 : Le k-ième rayon est sortant et commence à $2n+1-r_k$;
10 : **Fin Si**
11 : **Fin Pour**
12 : Le g-ième rayon est entrant et commence à 1 ;

Nous avons alors introduit plusieurs autres classes de stricte permutation spirale après les quenines, pérecquines et mongines : les roubines. Nous avons caractérisé ces roubines, par des techniques de générateurs congruentiels linéaires, et en avons déduit la spiraline, permutation spirale pour tout entier. Enfin, nous avons classifié les notions de jolie permutation spirale, de permutation spirale ordonnée, entrante ou sortante, grâce à leurs rayons. Ceci a donné notamment la sortine (permutation spirale ordonnée sortante) et une alternative graphique aux spiralines et aux spinines pour la

construction générale de permutations spirales, donnée dans l'algorithme 10.

Algorithme 10 Construction d'une sortine

Entrées : – Deux entiers $n > g > 0$.
Sorties : – La n, g-sortine.
1 : $m := n \mod g$; {Nombre de grands rayons}
2 : $L := \lfloor \frac{n}{g} \rfloor + 1$; {Longueur des grands rayons}
3 : $l := \lfloor \frac{n}{g} \rfloor$; {Longueur des petits rayons}
4 : **Pour** $k = 0$ **jusqu'à** $m - 1$ **Faire**
5 : $x := n - kL$; {Point de départ du $k+1$-ième grand rayon}
6 : $R_{k+1} =$ `[x<-(x-L+1)]`;
7 : **Fin Pour**
8 : **Pour** $k = m$ **jusqu'à** $g - 1$ **Faire**
9 : $x := n - kl - m$; {Point de départ du $k - m + 1$-ième petit rayon}
10 : $R_{k+1} =$ `[x<-(x-l+1)]`;
11 : **Fin Pour**

La figure 3 donne la première permutation spirale de taille 7 et une permutation de taille 18.

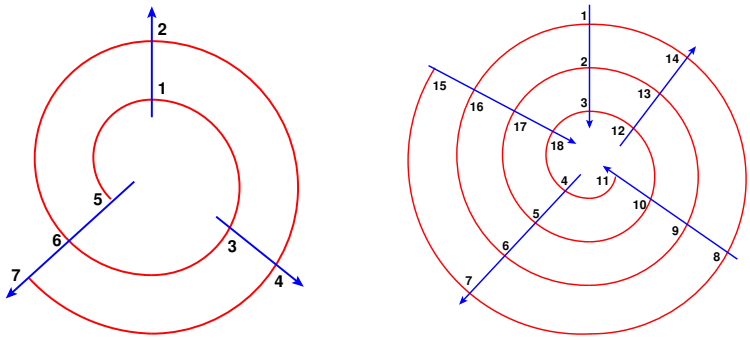


Figure 3: La septine et une 5-dixhuitine

Enfin, nous avons remarqué que sur une représentation carrée, les rayons des permutations spirales deviennent des segments parallèles ou à angle symétrique. Cette observation rejoint celles faites sur les générateurs congruentiels linéaires dont les valeurs restent dans des hyperplans parallèles [161 - Marsaglia (1968)].

- Outre plusieurs conjectures données dans [T60], notamment sur le rayon minimal d'une permutation spirale, le lien avec les bases normales optimales mériterait plus d'attention. En particulier, il serait intéressant de comprendre comment une permutation d'ordre maximal peut induire une matrice à nombre d'éléments non nuls minimal. Les périodes de Gauß [170 - Mullin et al. (1989)] sont peut-être une clef.

2.5 Attaques par perturbation sur RSA

Un domaine important d'utilisation de l'arithmétique est la cryptanalyse de chiffrements utilisant la théorie des nombres. Dans l'article [B20], nous avons proposé une attaque par perturbation sur le déchiffrement RSA de gauche à droite.

Les hypothèses de travail sont les suivantes :

- Un schéma RSA avec $N = pq$ public, produit de deux nombres premiers, des clefs publiques et privées e et d vérifiant $e \cdot d \equiv 1 \mod \varphi(N)$.
- La clef secrète RSA, d, est stockée sur un appareil (carte à puce) dont on a acquis la maîtrise.
- Un déchiffrement RSA de type $m \equiv C^d \mod N$ est effectué par exponentiation récursive par carrés sur l'appareil.
- Il est possible de perturber un ou plusieurs octets du modulo N durant le calcul (par exemple par laser) à des moments relativement précis de ce calcul.
- L'algorithme d'exponentiation peut être protégé contre les attaques par canaux auxiliaires (redondance de multiplications) mais pas contre les modifications du modulo.

En considérant le développement binaire de d, l'exponentiation récursive est effectuée principalement par deux méthodes : de droite à gauche

$$C^{d_0+2d_1+\cdots+2^i d_i} = C^{d_0} \left(C^2\right)^{d_1} \cdots \left(C^{2^i}\right)^{d_i}$$

ou de gauche à droite

$$C^{d_0+2d_1+\cdots+2^i d_i} = \left(\cdots\left(\left(C^{d_i}\right)^2 C^{d_{i-1}}\right)^2 \cdots C^{d_1}\right)^2 C^{d_0}$$

L'attaque de [85 - Berzati et al. (2008)] consiste à modifier le modulo en cours de calcul sur l'exponentiation de droite à gauche : N devient $\tilde{N}$ (les résultats seront comparés grâce à un dictionnaire de $\tilde{N}$ précalculé). Il est alors possible d'obtenir les bits de d en comparant les résultats avec et sans perturbation.

Dans le cas de l'exponentiation de gauche à droite, la recherche est plus complexe. Nous l'illustrons sur une recherche des deux derniers bits :

1. Dans un premier calcul correct on obtient :

$$m \equiv C^d \equiv \left(R^2 \cdot C^{d_1}\right)^2 C^{d_0} \mod N$$

2. Après modification de N en cours d'algorithme sur un second calcul, on obtient :

$$\tilde{m} = \left((R^2 \mod N) \cdot C^{d_1}\right) C^{d_0} \mod \tilde{N}$$

Suivant les valeurs de d_0 et d_1 nous avons alors 4 cas :

$$\begin{array}{lll}
\text{Si } d_1 d_0 = 00 & \text{Alors} & \left(\sqrt{\tilde{m} \mod \tilde{N}}\right)^2 \equiv m \mod N \\
\text{Si } d_1 d_0 = 01 & \text{Alors} & \left(\sqrt{\tilde{m} C^{-1} \mod \tilde{N}}\right)^2 \equiv mC^{-1} \mod N \\
\text{Si } d_1 d_0 = 10 & \text{Alors} & \left(\left(\sqrt{\tilde{m}} \cdot C^{-1} \mod \tilde{N}\right) \cdot C\right)^2 \equiv m \mod N \\
\text{Si } d_1 d_0 = 11 & \text{Alors} & \left(\left(\sqrt{\tilde{m} C^{-1}} C^{-1} \mod \tilde{N}\right) \cdot C\right)^2 \equiv mC^{-1} \mod N
\end{array}$$

Pour récupérer les bits de d, il faut alors être capable de calculer des racines carrées soit modulo N soit modulo $\tilde{N}$. Or il est aussi difficile de calculer des racines modulo un nombre composé que de factoriser ce nombre.

L'idée de [B20] est donc de précalculer un dictionnaire de $\tilde{N}$ premiers ou faciles à factoriser, puis de réaliser l'attaque de modification du modulo jusqu'à tomber sur un des modulo pour lesquels il est possible de calculer des racines carrées.

Nous avons alors montré que ces hypothèses étaient réalistes et permettaient de casser effectivement des codes RSA sur les tailles de modulo actuelles.

Tout d'abord une analyse théorique a montré que l'on peut trouver des modulo $\tilde{N}$ faciles à factoriser pour un N donné, dans plus de 99.996% des cas. En pratique, il n'a d'ailleurs pas été possible de trouver des N pour lesquels aucun $\tilde{N}$ n'était premier.

Ensuite, il s'agit de calculer les racines carrées le plus rapidement possible. Les ingrédients principaux sont connus (voir par exemple [190 - Shoup (2005), §12.5]) : il s'agit de calculer modulo chaque facteur premier, puis de remonter modulo les puissances des facteurs premiers par Hensel et enfin de reconstruire le résultat par restes chinois. Un point restait flou : la remontée de Hensel en caractéristique 2.

En effet, la remontée de Hensel modulo p^k impair s'écrit de la manière suivante :

Lemme 11

Soient p premier, $a \in \mathbb{Z}/p\mathbb{Z}^*$, x_0 tel que $x_0^2 \equiv a \mod p$ et soient x_i et h_i définis par la remontée de Hensel : $h_{i+1} \equiv \frac{(a-x_i^2) \mod p^{2^{i+1}}}{p^{2^i}} \cdot (2x_i)^{-1} \mod p^{2^i} \mod p^{2^i}$ et $x_{i+1} \equiv x_i + h_{i+1}p^{2^i} \mod p^{2^{i+1}}$ Alors x_i est inversible modulo p^{2^k}, pour tout k.

On voit que le calcul nécessite une inversion modulaire de 2. Or en caractéristique 2, évidemment, 2 n'est pas inversible. Il faut donc modifier l'algorithme.

Tout d'abord, on traite les cas $k = 1, 2, 3$ directement.

Ensuite, si a est pair, on calcule $a = b2^t$ avec b impair.
Si t est impair, a n'a pas de racine carrée modulo 2^k.
Si t est pair, soit u une racine carrée de b, alors $x = u2^{t/2}$ est une racine carrée de a.

Enfin, le cas restant est a est impair et on a trouvé une racine x_0 modulo 8, i.e. $a \equiv 1 \mod 8$ et x_0 est impair. On a alors la variante suivante de l'algorithme précédent :

Lemme 12

Soient a impair, x_0 tel que $x_0^2 \equiv a \mod 8$ et $c_0 = 3$. Soient c_i, x_i et h_i définis par la remontée de Hensel en caractéristique 2 : $c_{i+1} = 2c_i - 2$, $h_{i+1} \equiv \frac{(a-x_i^2) \mod 2^{2c_i-2}}{2^{c_i}} \cdot (x_i)^{-1} \mod 2^{c_i-1} \mod 2^{c_i-1}$ et $x_{i+1} \equiv x_i + h_{i+1}2^{c_i-1} \mod 2^{2c_i-2}$. Alors x_i est inversible modulo 2^k, pour tout k.

PREUVE. Même preuve que celle du lemme 11, par récurrence, si a est impair, 2 ne divise pas x_0 et ensuite 2 ne peut pas diviser x_{i+1}. □

Pour le nombre d'itérations, au lieu de travailler modulo 2 à la puissance 2^i à la i-ième itération, on travaille modulo 2 à la puissance $2^{i-1}+2$. L'ordre reste donc bien logarithmique.

Au final on obtient donc la borne de complexité suivante pour l'attaque, implémentée avec GIVARO :

Théorème 13

L'attaque sur l'exponentiation de droite à gauche d'un modulo RSA de N bits, par blocs de perturbations sur l bits nécessite

$$\frac{2^{8+l} \cdot log_2(N)^3 \cdot (log_2(N)+l)}{16 \cdot l} \text{ exponentiations.}$$

Par ailleurs, l'attaque est probabiliste : en effet, il est possible qu'un modulo perturbé reproduise la même valeur de déchiffrement qu'un modulo non perturbé pour certaines entrées. Cette probabilité de faux-positif est faible et vérifie :

$$0 \leq \Pr[F.Pos.] < min \left\{ \frac{(N-1)\cdot 2^l \cdot D_{length}}{N \cdot (2^l \cdot D_{length} - 1)} \; ; \; \frac{2^l \cdot D_{length}}{N} \right\} \tag{12}$$

où D_{length} est la taille du dictionnaire des modulo perturbés $\tilde{N}$ factorisables.

- Enfin, il existe une contre-mesure à cette attaque : à chaque calcul de chiffrement, la puce choisit un λ sur au moins 256 bits et effectue l'exponentiation non pas avec la clef secrète d mais avec $d + \lambda\varphi(N)$. Dans ce cas, à chaque étape de l'attaque on trouve k bits, mais d'un $\tilde{d}$ différent a priori des précédents. Une attaque par perturbation sur un exposant aléatoire reste donc à découvrir.
- Des extensions de ces attaques par perturbation sur les algorithmes embarqués sont également en cours, par exemple dans le projet SHIVA ou sur les chiffrements par flots.
- Plus généralement, la modification par perturbation des propriétés algébriques d'un chiffrement (comme rendre le modulo premier ou décomposable en plus de deux facteurs premiers) est un domaine en pleine expansion.

3 Algèbre linéaire

Le cœur de ce travail réside dans la réalisation efficace d'algorithmes d'algèbre linéaire exacte. Il se concrétise par la réalisation de la bibliothèque générique LINBOX[¶] développée conjointement entre la France, les États-Unis et le Canada. Le projet a démarré en 1997 sous l'égide de Erich Kaltofen (North Carolina State University), B. David Saunders (University of Delaware) et Gilles Villard (ÉNS Lyon). J'ai réalisé le premier prototype de la bibliothèque durant ma thèse et la première version stable, fruit du travail d'une trentaine de chercheurs, est sortie en 2005. Aujourd'hui la bibliothèque est devenue la référence internationale pour l'algèbre linéaire exacte. Le design et les algorithmes ont été repris par des logiciels de calcul scientifique comme Maple[§] ou Magma[ʻ] et la bibliothèque est utilisée comme noyau par le système SAGE.

L'analyse de complexité en temps et en mémoire est l'élément principal pour comparer les algorithmes et implémentations proposées à l'existant. Ainsi, dans cette partie, nous proposons des améliorations d'exposants (parfois seulement au niveau des facteurs logarithmiques) pour le déterminant sur les entiers, le polynôme caractéristique dense et creux, algébrique et entier, la forme de Kalman, la résolution de systèmes en parallèle ; des améliorations de constantes pour l'algèbre linéaire dense sur des corps finis.

Outre la création de nouveaux algorithmes, la technique principale utilisée est l'adaptativité : déterminer différents algorithmes et implémentations candidats puis construire une cascade d'algorithmes par des choix statiques et dynamiques.

Avec les avancées récentes en terme de complexité binaire des algorithmes à coefficients entiers, rationnels ou polynomiaux et avec les nouveaux paradigmes de calcul multi-cœurs, le travail à réaliser pour obtenir des noyaux de calcul *efficaces*, *portables* et *pérennes* est immense.

Durant mon doctorat, nous avions proposé une méthode de triangularisation de matrices denses améliorant la localité des opérations [T69, A7]. Une avancée simple a ensuite été l'introduction de routines numériques denses pour travailler sur des corps finis. Cela a permis d'aligner la vitesse de calcul exact sur la vitesse de calcul numérique. À partir de là, nous construisons toute l'algèbre linéaire dense sur des corps finis en la ramenant au produit de matrices pour les tailles asymptotiques et en travaillant sur les dégénérations pour les plus petites tailles. Ainsi, la réduction du calcul du polynôme caractéristique à la multiplication de matrices s'est avérée particulièrement efficace.

Ensuite, le cas creux reste toujours difficile, les problèmes étant très variés, les méthodes directes restant en général assez imprévisibles et les méthodes itératives relativement lentes. Durant mon doctorat, nous avions travaillé sur le calcul du rang [T69, B29] et de la forme normale de Smith [T69, A8]. Dorénavant, le travail se focalise sur les algorithmes par blocs, qui doivent pouvoir bénéficier en partie des avancées denses et également des ressources de plus en plus multi-cœurs. Les problèmes algorithmiques (fiabilité sur les petits corps finis, besoin de préconditionnements, etc.) ne sont pas encore bien résolus et les implémentations nécessitent souvent des machineries assez complexes pour traiter des cas particuliers. Nous allons donc vers une modularité plus grande du produit matrice-vecteur ou matrice-bloc, vers un parallélisme sous-jacent et également modulable, afin de fournir ici également des noyaux performants et pérennes et afin de ramener efficacement les problèmes à la résolution de systèmes (Lanczos) ou au calcul du polynôme minimal (Wiedemann).

[¶] http://linalg.org
[§] http://www.maplesoft.com
[ʻ] http://magma.maths.usyd.edu.au/magma

Enfin, le troisième chantier est celui des algorithmes pour matrices à coefficients entiers ou polynomiaux. Depuis une dizaine d'années et les travaux de G. Villard puis A. Storjohann, notamment, les complexités binaires (tenant compte du grossissement exponentiel des coefficients) sont progressivement ramenées au niveau des complexités algébriques ou au niveau, optimal, de la taille des sorties. Nous travaillons sur les complexités du calcul du déterminant entier, du calcul du polynôme caractéristique entier, du calcul du noyau.

3.1 FFLAS : multiplication de matrices denses dans les corps finis

Dans [A16, A4], nous avons proposé un noyau de calcul de la multiplication de matrices dans des corps finis de la taille du mot machine. Les caractéristiques principales de ce noyau sont :

1. Réduction modulaire retardée ou simultanée.
2. Adaptation aux niveaux de cache et utilisation des routines numériques BLAS (*Basic Linear Algebra Subprograms*).
3. Utilisation de l'algorithme rapide de Strassen-Winograd.

L'idée est simple, il s'agit de convertir une matrice dans un corps fini vers une représentation flottante ; utiliser une routine numérique optimisée pour faire les calculs (et donc retarder le modulo implicitement) ; convertir les flottants obtenus de nouveau vers le corps fini.

Cela permet d'avoir à la fois l'efficacité de l'arithmétique flottante, l'adaptabilité aux caches mémoires des BLAS ainsi qu'une portabilité et une pérennisation du code. Les distributions numériques de BLAS les plus utilisées par LinBox sont ATLAS[††] et GotoBLAS[‡‡], mais toute distribution de BLAS numériques peut être utilisée automatiquement par notre noyau exact.

La figure 4 montre les avantages de cette méthode (`FFLAS::classic`) par rapport à des méthodes sans arithmétique flottante (la triple boucle naïve sur des entiers, `long-noblock`, et une implémentation optimisant l'utilisation des caches sur des entiers machine, `long-block-40`, ici sur des blocs de taille 40). La figure montre également les performances des routines numériques (`dgemm`) ainsi que la vitesse équivalente obtenue en utilisant l'algorithme rapide de Strassen-Winograd au-dessus de ces mêmes routines numériques (`FFLAS::fgemm`).

3.2 Ordonnancement et contrôle du grossissement des coefficients dans l'algorithme de Strassen-Winograd

En calcul numérique, les algorithmes de borne de complexité asymptotique meilleure que n^3 sont rarement utilisés du fait de problèmes de stabilité. En calcul exact, au contraire, ces algorithmes peuvent être remis au goût du jour puisqu'ils permettent de réduire le nombre total d'opérations arithmétiques.

Dans un premier temps, nous avons choisi la variante de Winograd [119 - Gathen et Gerhard (1999), Algorithme 12.1] de l'algorithme de Strassen[195 - Strassen (1969)], pour son applicabilité aux tailles de matrices aujourd'hui traitables.

Nous notons MM(n) le terme dominant de la complexité arithmétique des algorithmes de multiplication de matrices. La valeur de MM(n) reflète alors le choix

[††] `http://math-atlas.sourceforge.net` [207 - Whaley et al. (2001)]

[‡‡] `http://www.tacc.utexas.edu/tacc-projects/gotoblas2` [131 - Goto et van de Geijn (2008)]

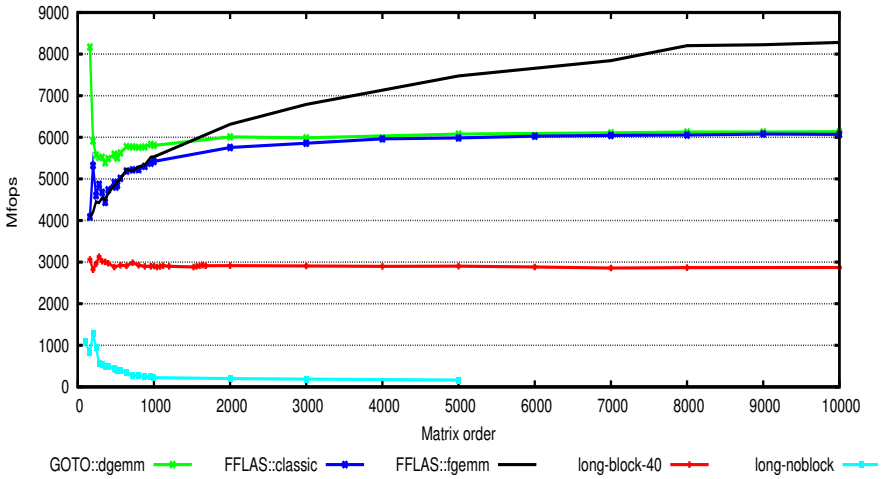


Figure 4: Multiplication de matrices par blocs modulo 65521 sur un Xeon 3.6GHz.

de l'algorithme, e.g. $\mathrm{MM}(n) = 2n^3$ pour l'algorithme classique, et signifie que la meilleure borne de complexité actuelle est $2n^3 + n^2$. On note également ω l'exposant asymptotique de $\mathrm{MM}(n)$. Pour l'algorithme classique, $\omega = 3$, pour l'algorithme de Strassen-Winograd $\omega = \log_2(7) \approx 2.807354922$, la meilleure complexité connue étant environ 2.375477 par [93 - Coppersmith et Winograd (1990)]. En outre, nous pouvons borner le coût d'une multiplication rectangulaire $m \times k$ par $k \times n$ (notée $R(m,k,n)$) comme suit : $R(m,k,n) \leq C_\omega \left\lceil \frac{m}{z} \right\rceil \left\lceil \frac{n}{z} \right\rceil \left\lceil \frac{k}{z} \right\rceil z^\omega$ où $z = \min(m,k,n)$ [137 - Huang et Pan (1997)]. Les implémentations que nous donnons doivent être performantes pour n quelconque, même si les algorithmes sont souvent décrits pour n une puissance de 2. Plusieurs techniques entrent en jeu : d'abord la dégénérescence d'un algorithme récursif, où la récursion est arrêtée à un niveau seuil pour appeler de manière terminale un autre algorithme ; ensuite des techniques de "padding", où des zéros virtuels sont ajoutés, ou encore de "peeling"où des bandes de taille 1 sont découpées et plus généralement de partition de n. Plus de détails à ce sujet sont donnés dans [177 - Pernet (2006)].

3.2.1 Contrôle du grossissement des coefficients

L'implémentation de la multiplication de matrices rapide dans FFLAS commence par découper les matrices en blocs, appelle l'algorithme de Winograd sur quelques niveaux récursifs, puis applique l'algorithme classique quand les blocs sont suffisamment petits.

Le nombre optimal de niveaux récursifs dépend de la taille initiale de la matrice et de l'architecture :

Lemme 14

Si la taille de matrice, pour laquelle un niveau récursif de Winograd est exactement aussi rapide que l'algorithme classique, est notée w, alors le nombre optimal de niveaux récursifs est donné par

$$l = \left\lfloor log_2 \frac{n}{w} \right\rfloor + 1.$$

Ensuite cet algorithme va être utilisé avec des réductions retardées. Il faut alors s'assurer que les résultats intermédiaires ne vont pas dépasser la taille de la mantisse utilisée. Nous avons montré en toute généralité que :

Théorème 15

Soit $A \in \mathbb{Z}^{m\times k}$, $B \in \mathbb{Z}^{k\times n}$ $C \in \mathbb{Z}^{m\times n}$ trois matrices et $\beta \in \mathbb{Z}$ avec $m_A \leq a_{i,j} \leq M_A$, $m_B \leq b_{i,j} \leq M_B$ et $m_C \leq c_{i,j} \leq M_C$. On suppose de plus que $0 \leq -m_A \leq M_A$, $0 \leq -m_B \leq M_B$, $0 \leq -m_C \leq M_C$, $M_C \leq M_B$ et $|\beta| \leq M_A, M_B$. Alors toute valeur intermédiaire z intervenant dans le calcul de $A \times B + \beta C$ avec l ($l \geq 1$) niveaux récursifs de l'algorithme de Winograd vérifie :

$$|z| \leq \left(\frac{1+3^l}{2}M_A + \frac{1-3^l}{2}m_A\right)\left(\frac{1+3^l}{2}M_B + \frac{1-3^l}{2}m_B\right)\left\lfloor \frac{k}{2^l} \right\rfloor$$

En outre cette borne est atteinte.

Le facteur 3 provient des calculs intermédiaires récursifs de type $(A_{21} + A_{22} - A_{11}) \times (B_{22} + B_{11} - B_{12})$, soit un facteur 3 par niveau récursif.

On en déduit que la représentation centrée est optimale parmi les représentations sur intervalle et donne les résultats pratiques du corollaire suivant ainsi que les performances de la courbe `FFLAS::fgemm` sur la figure 4.

Corollaire 16

Avec les mêmes notations et $a_{i,j}, b_{i,j}, c_{i,j}, \beta \in [-\frac{p-1}{2} \dots \frac{p-1}{2}]$, on a

$$|z| \leq \left(\frac{3^l}{2}\right)^2 \left\lfloor \frac{k}{2^l} \right\rfloor (p-1)^2$$

Par ailleurs, il est possible de calculer l niveaux récursifs de l'algorithme de Winograd sans réduction modulaire, avec une mantisse de γ dès que $k < k_{Winograd}$ où

$$k_{Winograd} = \left(\frac{2^{\gamma+2}}{(3^l(p-1))^2} + 1\right) 2^l$$

3.2.2 Ordonnancement

Un autre aspect de l'implémentation de l'algorithme de Strassen-Winograd est la consommation éventuelle d'espace mémoire intermédiaire supplémentaire. En effet, nous pouvons aujourd'hui, par exemple, multiplier deux matrices de taille 10000×10000 sur $\mathbb{Z}/65521\mathbb{Z}$ en moins de 200 secondes sur un PC de bureau, pourvu que celui-ci dispose d'au moins 3 Giga octets de mémoire RAM. Outre les aspects quantitatifs, les accès mémoires peuvent être lents comparés à la vitesse de calcul. Il est donc essentiel de minimiser l'espace mémoire requis dans la multiplication de matrices. Nous avons proposé dans [A10] plusieurs nouveaux ordonnancements dont les caractéristiques sont résumées dans le tableau 4.

$C \leftarrow A \times B$	$C \leftarrow C + A \times B$
$2/3n^2$ [103 - Douglas et al. (1994)]	n^2 [139 - Huss-Lederman et al. (1996)]
0 (en place sur A et/ou B)	$2/3n^2$ (matrices carrées)
$\rightarrow 0$ (compromis Temps-Mémoire)	$\rightarrow 0$ (compromis Temps-Mémoire)

Table 4: Mémoire supplémentaire pour la multiplication de matrices rapide (outre celle de A, B et C)

Les nouvelles idées introduites afin d'améliorer les ordonnancements existants ont été d'autoriser des préadditions pour former de nouvelles variantes, d'écraser des matrices intermédiaires (ou initiales quand autorisé), d'introduire des ordonnancements hybrides classiques/Winograd et de développer un générateur automatique d'ordonnancements par jeu de galets [S53]. Quand ce générateur pouvait être exhaustif, il a donc également fourni des certificats d'optimalité.

La table 5 donne par exemple un ordonnancement de l'opération $C \leftarrow \alpha A \times B + \beta C$ avec l'algorithme de Strassen-Winograd, utilisant seulement *deux* blocs temporaires. Cet ordonnancement donne le record actuel pour la minimisation de l'espace supplémentaire nécessaire dans l'algorithme de Strassen-Winograd à $\frac{2}{3}n^2$.

#	opération	loc.	#	opération	loc.
1	$Z_1 = C_{22} - C_{12}$	C_{22}	14	$P_2 = \texttt{Acc}(\alpha A_{12}B_{21} + \beta C_{11})$	C_{11}
2	$Z_3 = C_{12} - C_{21}$	C_{12}	15	$\mathbf{U_1} = P_1 + P_2$	C_{11}
3	$S_1 = A_{21} + A_{22}$	X	16	$\mathbf{U_5} = U_2 + P_3$	C_{12}
4	$T_1 = B_{12} - B_{11}$	Y	17	$S_3 = A_{11} - A_{21}$	X
5	$P_5 = \texttt{Acc}(\alpha S_1 T_1 + \beta Z_3)$	C_{12}	18	$T_3 = B_{22} - B_{12}$	Y
6	$S_2 = S_1 - A_{11}$	X	19	$U_3 = P_7 + U_2$	C_{21}
7	$T_2 = B_{22} - T_1$	Y		$= \alpha\texttt{AcLR}(S_3 T_3 + U_2)$	
8	$P_6 = \texttt{Acc}(\alpha S_2 T_2 + \beta C_{21})$	C_{21}	20	$\mathbf{U_7} = U_3 + W_1$	C_{22}
9	$S_4 = A_{12} - S_2$	X	21	$T'_1 = B_{12} - B_{11}$	Y
10	$W_1 = P_5 + \beta Z_1$	C_{22}	22	$T'_2 = B_{22} - T'_1$	Y
11	$P_3 = \texttt{Acc}(\alpha S_4 B_{22} + P_5)$	C_{12}	23	$T_4 = T'_2 - B_{21}$	Y
12	$P_1 = \alpha A_{11} B_{11}$	X	24	$\mathbf{U_6} = U_3 - P_4$	C_{21}
13	$U_2 = P_6 + P_1$	C_{21}		$= -\alpha\texttt{AccR}(A_{22} T_4 - U_3)$	

Table 5: Ordonnancement `Acc` pour l'opération $C \leftarrow \alpha A \times B + \beta C$ avec seulement 2 blocs intermédiaires temporaires ; `AcLR` écrase S_3 et T_3, `AccR` écrase T_4.

La table 6, page 31, donne un résumé des ordonnancements disponibles avec leurs complexités en temps et en mémoire respectives.

3.3 Multiplication de matrices compressées

Dans [B21, A2] nous avons développé des produits de matrices spécialisés pour les corps finis de très petite taille. La bibliothèque M4RI par exemple [73 - Albrecht et Bard (2008)], est spécialisée pour les calculs modulo 2 avec des algorithmes de type "4 Russians"[74 - Arlazarov et al. (1970), 72 - Aho et al. (1974)], et [89 - Boothby et Bradshaw (2009)] ont commencé à étendre ces implémentations modulo 3 et 5. Dans cette section, nous proposons deux méthodes *génériques* qui s'appliquent à tout nombre premier pourvu qu'il soit suffisamment petit par rapport à la mantisse disponible. Au contraire des méthodes de type "4 Russians"nous n'avons pas de facteur $\log(n)$ d'accélération, mais un facteur constant dépendant du nombre de résidus qu'il est possible de "compresser"dans un mot.

Les deux idées sont d'utiliser la multiplication de polynômes compressés rapide de la section 2.1.4 pour faire soit du produit scalaire, soit directement du produit de matrices.

3.3.1 Produit médian

La première idée est donc de récupérer un produit scalaire de vecteurs de taille $d+1$ grâce à une multiplication de polynômes effectuée dans un seul mot machine. Cela s'étend ensuite à la multiplication de matrices en compressant les deux matrices en entrée comme illustré ci-après pour des matrices 2×2 et $d = 1$. Le produit

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

est reconstruit à partir de

$$\begin{bmatrix} Qa+b \\ Qc+d \end{bmatrix} \times \begin{bmatrix} e+Qg & f+Qh \end{bmatrix} = \begin{bmatrix} *+(ae+bg)Q+*Q^2 & *+(af+bh)Q+*Q^2 \\ *+(ce+dg)Q+*Q^2 & *+(cf+dh)Q+*Q^2 \end{bmatrix},$$

où le caractère $*$ indique des coefficients inutiles.

En général, A est une matrice $m \times k$ à multiplier par une matrice B, $k \times n$. La matrice A est d'abord compressée en une matrice CA, $m \times \lceil k/(d+1) \rceil$ et B est transformée en une matrice CB, $\lceil k(d+1) \rceil \times n$. Les matrices compressées sont alors multipliées par la méthode désirée (y compris Strassen-Winograd avec des BLAS par exemple).

En terme de nombre d'opérations machine, la multiplication $CA \times CB$ permet une réduction d'un *facteur* $d+1$ par rapport à la multiplication de $A \times B$.

Nous avons vu en section 2.1.4 comment effectuer la compression efficacement, il reste à voir comment effectuer l'extraction du produit médian rapidement et exactement.

Si le produit $CA \times CB$ est réalisé en flottants (comme avec les FFLAS) il est donc au moins nécessaire pour que le calcul soit exact que le coefficient de degré d soit plus petit que la mantisse disponible.

Dans un premier temps, il parait nécessaire que la somme des coefficients jusqu'au degré d tienne dans la mantisse pour que le calcul soit exact. En fait, en précalculant et en arrondissant correctement l'inverse de Q^d, avec l'algorithme 1, on peut utiliser la virgule flottante et calculer plutôt :

$$\begin{bmatrix} *Q^{-1}+(ae+bg)+*Q & *Q^{-1}+(af+bh)+*Q \\ *Q^{-1}+(ce+dg)+*Q & *Q^{-1}+(cf+dh)+*Q \end{bmatrix},$$

	Algorithme	Entrées	Temporaires	Mémoire supplémentaire	Nombre d'allocations supplémentaires	Complexité arithmétique
$A \times B$	[103 - Douglas et al. (1994)]	constantes	2	$\frac{2}{3}n^2$	$\frac{2}{3}(n^{2.807} - n^2)$	$6n^{2.807} - 5n^2$
	[A10, Table 3]	écrasées	0	0	0	$6n^{2.807} - 5n^2$
	[A10, Tables 4 et 5]	A ou B écrasée	1	$\frac{1}{3}n^2$	$\frac{1}{4}n^2 \log_2(n)$	$6n^{2.807} - 5n^2$
	[A10, Algorithme 7.1]	constantes	0	0	0	$7.2n^{2.807} - 13n^2$
$\alpha A \times B + \beta C$	[139 - Huss-Lederman et al. (1996)]	constantes	3	n^2	$\frac{2}{3}n^{\log_2(7)} + n^{\log_2(5)} - \frac{5}{3}n^2$	$6n^{2.807} - 4n^2$
	[A10, Table 6]	écrasées	2	$\frac{2}{3}n^2$	$\frac{1}{2}n^2 \log_2(n)$	$6n^{2.807} - 4n^2 + \frac{1}{2}n^2 \log_2(n)$
	[A10, Table 7]	B écrasée	2	$\frac{2}{3}n^2$	$2n^{2.322} - 2n^2$	$6n^{2.807} - 4n^2 + \frac{1}{2}n^2 \log_2(n)$
	Table 5 [A10, Table 9]	constantes	2	$\frac{2}{3}n^2$	$\frac{2}{9}n^{2.807} + 2n^{2.322} - \frac{22}{9}n^2$	$6n^{2.807} - 4n^2 + \frac{4}{3}n^2 \log_2(n)$
	[A10, Algorithme 7.2]	constantes	N/A	$\frac{1}{4}n^2$	$\frac{1}{4}n^2$	$6.857n^{2.807} - 8n^2$
	[A10, Algorithme 7.2]	constantes	N/A	$\frac{1}{9}n^2$	$\frac{1}{9}n^2$	$7.414n^{2.807} - 12n^2$

Table 6: Complexités des ordonnancements pour la multiplication de matrices carrées

En ajoutant une réduction retardée, cela implique que le nombre premier et le degré de compression doivent vérifier :

$$\sum_{i=0}^{d} \frac{k}{d+1}(i+1)(p-1)^2 Q^{d-i} < 2^{\beta}.$$

Par ailleurs, d'après l'équation (3), la réduction retardée d'un produit de k termes requiert que :

$$k(p-1)^2 < Q. \tag{13}$$

et donc l'extraction est exacte si :

$$Q^{d+1} < 2^{\beta}. \tag{14}$$

Quelques astuces supplémentaires existent quand Q est une puissance de 2 : la figure 5 montre qu'une compression $d+1$ donne un facteur de gain de l'ordre de d dans ce cas.

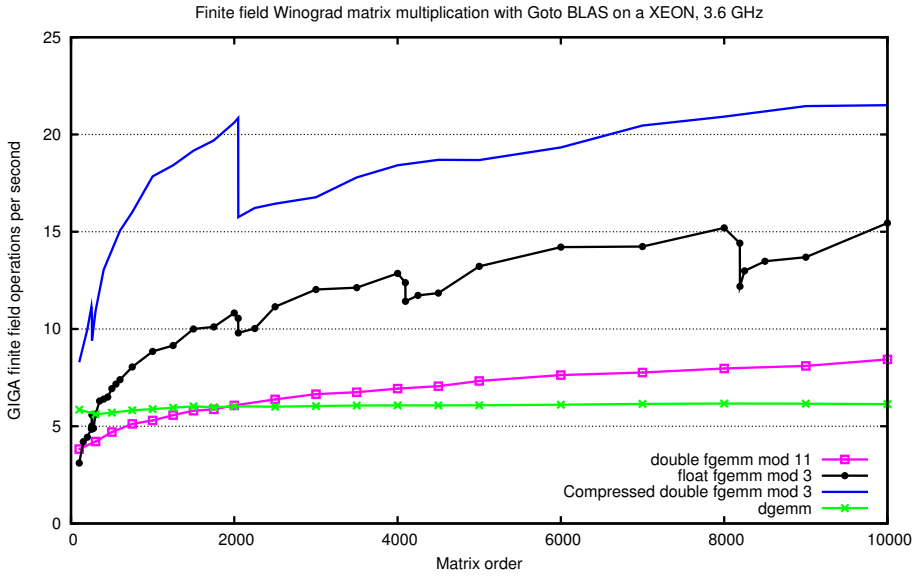


Figure 5: CMM vs `dgemm` (GotoBlas MM numérique) et `fgemm` (la routine exacte FFLAS) avec double ou simple précision.

- La figure 5 montre l'impact important du passage d'une compression 4 à une compression 3 entre $n = 2048$ et $n = 2049$. Il serait intéressant de comparer la multiplication d'une matrice de taille 2049, compressée 3 fois, avec une décomposition de cette même matrice en matrices de tailles 1024 et 1025 compressées 4 fois, mais avec plus de réductions modulaires en fin de calcul.

3.3.2 Multiplication de matrices compressées avec REDQ

Une autre façon de compresser est d'obtenir les différents coefficients de la matrice répartis dans les coefficients de la multiplication de polynômes. Par exemple, en

multipliant une matrice normale $m \times k$ à droite, par une matrice compressée lignes $k \times n/(d+1)$. Cela est illustré dans le cas 2×2 ci-après :

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e+Qf \\ g+Qh \end{bmatrix} = \begin{bmatrix} (ae+bg)+Q(af+bh) \\ (ce+dg)+Q(cf+dh) \end{bmatrix}$$

Il y a trois variantes possibles : compressée à droite, compressée à gauche ou compressée complète.

Pour la compression gauche et droite, Q et d doivent également vérifier les équations (13) et (14).

La différence principale avec l'algorithme CMM de la section 3.3.1 est qu'il faut récupérer simultanément tous les coefficients de la multiplication de polynômes. Cela est réalisé par l'algorithme 2.

Nous voyons sur la figure 6 que cette utilisation de REDQ est bénéfique : en effet, pour les petites matrices, le temps de compression peut représenter jusqu'à 30% du temps de calcul et donc la moindre amélioration a un impact important.

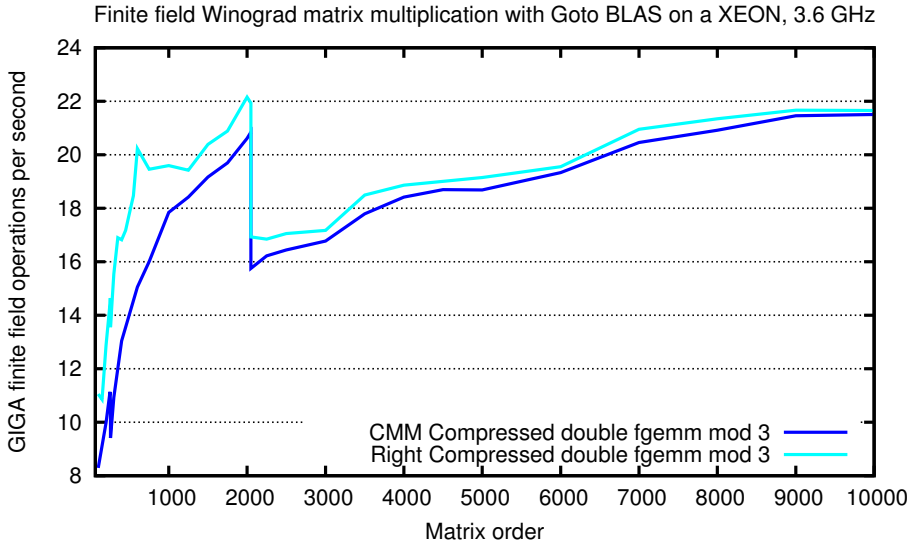


Figure 6: Compression à droite vs CMM.

Par exemple, avec l'algorithme de compression droite, la multiplication de deux matrices $10,000 \times 10,000$ modulo 3 a pris 90.73 secondes et il a fallu 1.63 seconde pour extraire le résultat modulaire des polynômes flottants. Le surcoût de structure représente donc moins de 2%. Pour des matrices 250×250 cela prend en moyenne moins de 0.00216 secondes pour multiplier et environ 0.0016 secondes pour convertir. Dans ce cas le surcoût représente 43% du temps. La table 7 donne une comparaison avec Magma V2.16-6 sur un cœur d'un AMD Opteron 2352 Barcelona 2.1GHz.

Enfin, le cas de la compression complète est différent car il nécessite deux substitutions de Kronecker distinctes ce qui impose de prendre Q de l'ordre du carré nécessaire pour les bornes (13). Les gains restent cependant du même ordre de grandeur car la compression est doublée. Cette compression est utile dans le cas de matrices rectangulaires, où le choix d'algorithme dépend de la plus grande dimension du produit.

Dimension	255	1000	2000	4000	7000	10000
Comp. droite GF(3)	6.02	14.81	22.85	19.97	21.36	23.83
magma GF(3)	**82.91**	**100.00**	**110.34**	**119.63**	**132.18**	**140.45**
Comp. droite GF(11)	4.68	**11.53**	10.52	**13.09**	**13.14**	9.58
magma GF(11)	**5.53**	8.33	**10.53**	11.89	12.09	**12.78**

Table 7: Vitesse, en Gffops, de la multiplication de matrices vs Magma v2.16-6, sur un opteron 2.1GHz

3.4 Résolution de systèmes triangulaires en cascade

Nous considérons ici un système linéaire $AX = B$ d'inconnue X, où A est une matrice triangulaire supérieure $m \times m$ inversible et X et B sont des matrices $m \times n$, avec $m \leq n$. Dans [A4] nous avons proposé un algorithme en cascade permettant de choisir entre différentes variantes pour résoudre ce problème sur un petit corps premier. Dans tous les cas, l'algorithme récursif par blocs permet d'utiliser la multiplication de matrices de la section 3.1 pour les grands blocs puis de dégénérer vers différentes variantes pour le traitement des blocs de petites tailles : utilisation de la routine numérique `dtrsm`, utilisation de l'arithmétique retardée de la section 2.1.2, et une combinaison des deux.

L'algorithme récursif est donc de type diviser pour régner et n'utilise que la multiplication de matrices comme calculs.

1. $X_2 :=$`trsm`(A_3, B_2)
2. $B_1 := B_1 - A_2X_2$
3. $X_1 :=$`trsm`(A_1, B_1)

Si $m = n$, le coût arithmétique de cet algorithme avec la multiplication de matrices classique est $TRSM(m) = m^3$.

Ce coût est relié à celui de la multiplication et de l'inversion de matrices triangulaires suivants, notés respectivement TRMM et INVT.

1. $X_1 :=$`trmm` (A_1, B_1);
2. $B_1 := B_1 + A_2X_2$;
3. $X_2 :=$`trmm` (A_3, B_2);

Le coût de ce dernier est $TRMM(m, n) = TRMM(m/2, n) + R(m/2, m/2, n)$, soit $TRMM(m, n) = m^2n$ avec la multiplication de matrices classique. Maintenant l'inverse d'une matrice triangulaire inversible est donnée ci-dessous :

$$\begin{bmatrix} A_1 & A_2 \\ & A_3 \end{bmatrix}^{-1} = \begin{bmatrix} A_1^{-1} & -A_1^{-1}A_2A_3^{-1} \\ & A_3^{-1} \end{bmatrix}.$$

Par conséquent, le coût de son calcul est donné par $INVT(m) = 2INVT(m/2) + 2TRMM(m/2, m/2)$, soit $INVT(m) = \frac{1}{3}m^3$ dans le cas de la multiplication classique.

3.4.1 Dégénération numérique

Il est possible d'utiliser la routine numérique `dtrsm` de la même façon que nous utilisons `dgemm` pour la multiplication de matrices si les conditions suivantes sont réunies :

- la diagonale de la matrice triangulaire est unitaire (pour éviter les divisions numériques qui n'auraient pas de sens dans le corps)
- tous les résultats intermédiaires doivent être représentables dans la mantisse numérique (le grossissement des coefficients doit être contrôlé).

Pour le premier point il faut donc préalablement factoriser la matrice triangulaire en $A = UD$ où U est unitaire et D est la diagonale de A. Après résolution du système triangulaire unitaire $UY = B$ par les BLAS, la solution du système initial est obtenue par $X = D^{-1}Y$ en n divisions sur le corps. Cette normalisation induit un coût supplémentaire de mn opérations arithmétiques.

En ce qui concerne le grossissement des coefficients, la k-ième ligne de la matrice solution X est une combinaison linéaire des $(n-k)$ dernières lignes de X, déjà calculées. Cela implique un grossissement linéaire en la taille des coefficients borné dans le théorème 17.

Théorème 17

Soit $T \in \mathbb{Z}^{n\times n}$ une matrice triangulaire diagonale unitaire et $b \in \mathbb{Z}^n$, avec $m \leq T_{i,j} \leq M$, $m \leq b_i \leq M$ et $m \leq 0 \leq M$. Soit $x = (x_i)_{i\in[1\dots n]} \in \mathbb{Z}^n$ la solution du système $Tx = b$. Alors $\forall\, k \in [0 \dots n-1]$:

$$\begin{cases} -u_k \leq x_{n-k} \leq v_k & \textit{pour } k \textit{ pair,} \\ -v_k \leq x_{n-k} \leq u_k & \textit{pour } k \textit{ impair} \end{cases}$$

avec

$$\begin{cases} u_k = \frac{M-m}{2}(M+1)^k - \frac{M+m}{2}(M-1)^k, \\ v_k = \frac{M-m}{2}(M+1)^k + \frac{M+m}{2}(M-1)^k. \end{cases}$$

Ainsi, il est possible de résoudre un système triangulaire unitaire de dimension n avec des entiers stockés sur γ bits si

$$\frac{p-1}{2}(p^{n-1} + (p-1)^{n-1}) < 2^\gamma \tag{15}$$

pour une représentation positive et si

$$\frac{p-1}{2}\left(\frac{p+1}{2}\right)^{n-1} < 2^\gamma \tag{16}$$

pour une représentation centrée.

Par exemple avec une mantisse de 53 bits on obtient des blocs de taille au plus 93×93 pour $p = 2$, au plus 4×4 pour $p \leq 19483$, et le plus grand $p = 189812531$ pour des blocs 2×2. Même si ces tailles de blocs semblent petites, en pratique, comme la division est dominante, une accélération est souvent de mise.

Dans la suite nous notons $S_{BLAS}(p)$ une telle taille maximale de matrice pour laquelle la résolution BLAS est exacte. Notre première cascade, `BLASTrsm`, devient donc : utilisation de l'algorithme récursif tant que la taille de bloc est plus petite que $S_{BLAS}(p)$, puis résolution BLAS.

3.4.2 Entrelacement avec le modulo retardé

Une autre idée est d'utiliser la réduction modulaire retardée sans passer par les BLAS : la multiplication de matrices (étape 2 de l'algorithme TRSM) est calculée dans $\mathbb{Z}$ et la réduction modulo p appliquée seulement quand nécessaire. Cette fois-ci le calcul est exact tant que la dimension commune k des deux blocs reste inférieure à un second seuil S_{DEL}. En choisissant $S_{DEL} = k_{\text{Winograd}}$, du corollaire 16, la multiplication de matrices peut être effectuée sans réduction modulaire. Nous le combinons avec le seuil précédent S_{BLAS}, en définissant S_{SPLIT} comme le plus grand multiple de S_{BLAS} plus petit que S_{DEL}. À partir de ces deux seuils, nous construisons le découpage du système de la figure 7.

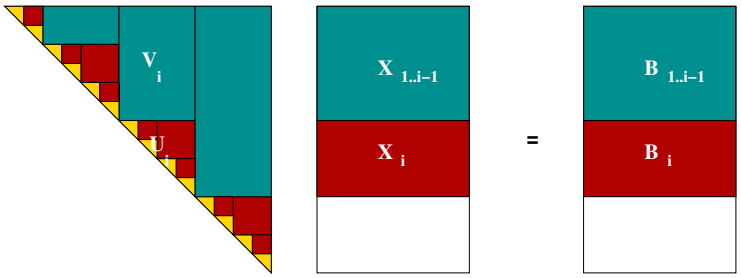


Figure 7: Découpage pour le TRSM en double cascade

L'algorithme général devient alors une boucle sur tous les blocs de colonnes de dimension S_{DEL}. Pour chacun d'entre eux, un système triangulaire est résolu par une sous-routine puis la solution est mise à jour par multiplication de matrices FFLAS. La sous-routine consiste à utiliser un algorithme en cascade formé de l'algorithme récursif par blocs et d'une résolution BLAS par `dtrsm` dès que la taille du bloc devient plus petite que S_{BLAS}. Dans cette dernière, la multiplication de matrices utilise le modulo retardé et donc les seules réductions modulaires ont lieu après les appels à `dtrsm`.

Pour les résultats, par exemple, la réduction retardée améliore de 500 Mffops la vitesse de calcul, et cela pour $p = 5$ et $p = 1\,048\,583$ puisque dans les deux cas $n < S_{\text{DEL}}$, ou encore, la table 8 indique que la vitesse de calcul de la résolution de système triangulaire dans un petit corps premier, `ftrsm`, est comparable au numérique.

n	*1000*	*2000*	*3000*	*5000*	*7000*	*8000*	*9000*	*10000*
ftrsm	0.25s	1.66s	5.08s	21.47s	**55.95s**	**80.77s**	**111.57s**	**150.81s**
dtrsm	**0.17s**	**1.35s**	**4.50s**	**20.64s**	56.19s	83.85s	119.18s	163.33s
$\frac{ftrsm}{dtrsm}$	1.47	1.23	1.13	1.04	1.00	0.96	0.94	0.92

Table 8: Résolution triangulaire multiple avec GotoBLAS sur un Xeon, 3.6GHz

3.4.3 Résolution de systèmes à dégénération parallèle

Ce cadre nous a permis également de proposer un nouvel algorithme parallèle pour TRSM [B33]. L'idée est que, pendant que X_2 et B_1 sont calculés, des processeurs supplémentaires pourraient calculer des morceaux de A_1^{-1}. En effet, X_1 peut être calculé de deux manières différentes :

i. $X_1 = TRSM(A_1, B_1)$: le coût arithmétique est $T_1 = k^3$ mais le temps parallèle est $T_\infty = k$;

ii. $X_1 = TRMM(A_1^{-1}, B_1)$: le coût arithmétique est identique $T_1 = k^3$ mais le temps parallèle est bien meilleur $T_\infty = \log k$.

La version avec TRMM est plus efficace d'un point de vue parallèle. Mais le calcul de A_1^{-1} induit un accroissement du nombre d'opérations arithmétiques et donc ne devrait être effectué que si des processeurs supplémentaires sont inactifs. La difficulté est donc de décider de la taille k de la matrice A_1 qui doit être inversée en parallèle. Pour une valeur optimale, ce calcul se terminerait en même temps ou légèrement après les calculs de X_2 et B_1. Cette valeur optimale dépend de nombreux facteurs : nombre de processeurs, architectures des processeurs, sous-routines, etc. Nous avons donc proposé un algorithme introspectif, estimant automatiquement cette valeur optimale durant l'exécution et couplant les deux versions ci-dessus. En effet, les calculs récursifs de TRSM vont naturellement débuter par les blocs en bas à droite. Ces calculs sont donc effectués avec une grande priorité. Au contraire, on lance simultanément le calcul de A_1^{-1} en basse priorité et les appels récursifs à TRSM sur les blocs hauts gauches interrogent simplement le calcul d'inverse pour connaître son avancement, comme sur la figure 8.

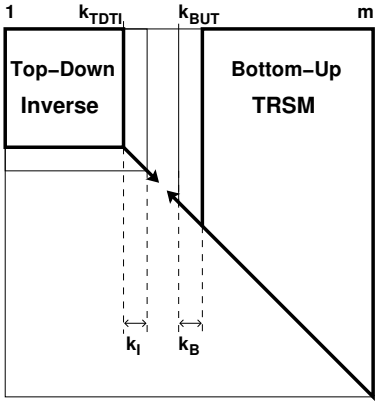


Figure 8: TRSM parallèle et introspectif

Une étude des paramètres nous permet alors de donner un algorithme de borne de complexité $T_1 = n^2$ et $T_\infty = \sqrt{n}\log^2(n)$ pour la résolution avec un seul vecteur second membre. Cette complexité est similaire à celle obtenue par [175 - Pan et Preparata (1995)] avec un algorithme à grain fin. Au contraire, notre algorithme est à gros grain et s'adapte automatiquement à l'hétérogénéité des ressources, même limitées.

3.5 Triangularisation dense et routines usuelles

Avec la multiplication de matrices et la résolution de systèmes triangulaires, il est possible de réduire l'élimination de Gauß à l'exposant n^ω. Les factorisations classiques LDU ou LUP (voir e.g. [72 - Aho et al. (1974)]) ne traitent pas le cas singulier, il faut donc utiliser la factorisation LQUP de [141 - Ibarra et al. (1982)]. Durant mon doctorat, nous avons proposé avec Jean-Louis Roch un algorithme améliorant la localité de cette factorisation, et donc intéressant en *out-of-core* ou en parallèle [T69, B30, A7].

Avec Clément Pernet et Pascal Giorgi, nous avons ensuite, dans [A15, A4], donné une version efficace *en place* de la factorisation LQUP par blocs, et des applications à la majorité des routines de base d'algèbre linéaire dense.

3.5.1 Triangularisation en place

Soit A une matrice $m \times n$, nous voulons calculer $< L, Q, U, P >$ telles que $A = LQUP$. La matrice `L` triangulaire inférieure unitaire, `P` et `Q` sont des permutations et `U` est triangulaire supérieure de rang r avec ses premières r lignes non nulles.

L'algorithme de [141 - Ibarra et al. (1982)] est récursif : tout d'abord il découpe A et procède par un appel récursif sur la première moitié. Après permutations, il donne les blocs T, Y et L_1 de la figure 9, et des permutations de lignes stockées dans Q. Ensuite, après des permutations de colonnes ($[XZ] = [A_{21}A_{22}]P$), l'algorithme calcule G tel que $GT = X$ via `trsm`, remplace X par des zéros et met à jour $Z = Z - GY$. La dernière étape est alors un appel récursif sur Z, suivi d'une mise à jour de Q (voir aussi [86 - Bini et Pan (1994), (2.7c)] pour plus de détails).

L'idée de notre implémentation LQUP est de stocker L sous une forme compressée $\tilde{L}$ de sorte que la triangularisation soit totalement en place.

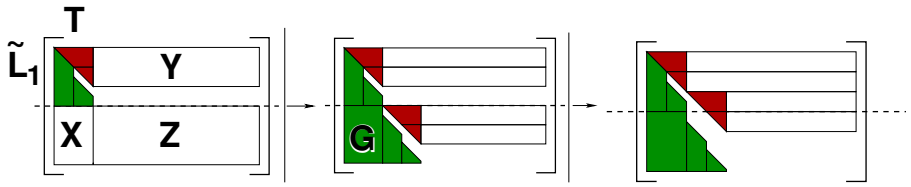


Figure 9: Principe de la factorisation LQUP en place

Lemme 18

Le terme dominant de la complexité temporelle de l'algorithme LQUP avec $m \leq n$ est

$$LQUP(m,n) = \left(\left\lceil \frac{n}{m} \right\rceil \frac{1}{2^{\omega-1}-2} - \frac{1}{2^{\omega}-2} \right) MM(m).$$

avec la multiplication de matrices classique, cela devient $nm^2 - \frac{1}{3}m^3$.

Cet algorithme donne également de très bonnes performances en pratique, comme illustré dans la table 9.

Une alternative à cet algorithme, si par exemple la capacité de la mémoire vive est insuffisante, est d'utiliser l'algorithme TURBO [T69, B30, A7], développé durant mon doctorat, qui maintient les blocs les plus carrés possibles à la place des blocs rectangulaires de LQUP. La localité est alors meilleure et les performances s'en ressentent. Cela est en faveur d'un développement plus important de structures de données récursives par blocs.

n	*1000*	*2000*	*3000*	*5000*	*7000*	*8000*	*9000*	*10000*
lqup	0.25s	1.52s	4.47s	17.93s	44.54s	67.88s	89.63s	119.65s
dgetrf	**0.15s**	**1.03s**	**3.33s**	**14.84s**	**39.58s**	**58.61s**	**82.89s**	**113.47s**
$\frac{lqup}{dgetrf}$	1.67	1.48	1.34	1.21	1.13	1.16	1.08	1.05

Table 9: Performances de la triangularisation (pour $\mathbb{Z}65521$ et en flottant) avec GotoBLAS sur un Xeon, 3.6GHz

3.5.2 Réductions

Dans [A4], nous nous sommes également attachés à donner des algorithmes, et le terme dominant de la complexité associée, pour plusieurs problèmes se réduisant à la multiplication de matrices. Nous avons cherché les meilleures constantes afin de déterminer la variante à implémenter dans LINBOX et FFPACK [S56]. Les résultats sont résumés dans la table 10.

Algorithme	Réduction	Si $\omega = 3$
TRSM	$\frac{1}{2^{\omega-1}-2}\lceil\frac{n}{m}\rceil$ MM(m)	m^2n
LQUP	$\left(\lceil\frac{n}{m}\rceil\frac{1}{2^{\omega-1}-2}-\frac{1}{2^\omega-2}\right)$ MM(m)	$nm^2-\frac{1}{3}m^3$
LDL^T	$\frac{4}{(2^\omega-4)(2^\omega-2)}MM(n)$	$\frac{1}{3}n^3$
TRMM	$\frac{1}{2^{\omega-1}-2}$MM(n)	n^3
U^{T}U^T	$\frac{4}{(2^\omega-4)(2^\omega-2)}MM(n)$	$\frac{1}{3}n^3$
U^{T}U	$\frac{4}{(2^\omega-4)(2^\omega-2)}MM(n)$	$\frac{1}{3}n^3$
U^{T}L^T	$\frac{2^\omega}{(2^\omega-4)(2^\omega-2)}MM(n)$	$\frac{2}{3}n^3$
AA^T	$\lceil\frac{n}{m}\rceil\frac{2}{2^\omega-4}MM(m)$	nm^2
SymAA^T	$\frac{2(2^\omega-3)}{(2^\omega-4)(2^\omega-2)}MM(n)$	$\frac{5}{6}n^3$
NullSpace	$(\lceil\frac{n}{m}\rceil\frac{2}{2^{\omega-1}-2}-\frac{1}{2^\omega-2})MM(m)$	$2m^2n-\frac{4}{3}m^3$
INV	$\frac{3\times2^\omega}{(2^\omega-4)(2^\omega-2)}MM(n)$	$2n^3$
INVT	$\frac{4}{(2^\omega-4)(2^\omega-2)}MM(n)$	$\frac{1}{3}n^3$
SymINV	$\frac{12}{(2^\omega-2)(2^\omega-4)}MM(n)$	n^3
MPINV	$\left(\lceil\frac{n}{m}\rceil\frac{6}{2^\omega-4}+\frac{4}{(2^\omega-2)(2^\omega-4)}\right)$ MM(m)	$3m^2n+\frac{1}{3}m^3$

Table 10: Réductions à la multiplication de matrices

Dans cette table, MPINV désigne le calcul de la pseudo-inverse de Moore-Penrose pour une matrice rectangulaire de plein rang. Nous avons également proposé un nouvel algorithme pour la multiplication de Moore-Penrose dans le cas non inversible de complexité : $2rmn + 2r^2m + 2r^2n + m^2n - \frac{1}{3}m^3 - \frac{4}{3}r^3$, quand $\omega = 3$, avec r le rang de la matrice. À titre de comparaison, le meilleur algorithme jusqu'alors était dû à [94 - Courrieu (2005)], en $3m^2n + 2r^2m + 3r^3$; notre algorithme améliore donc la meilleure constante connue d'environ un tiers.

Dans tous les cas de la table 10, les performances sur un corps fini sont comparables au numérique, voire meilleures (par exemple pour les calculs d'inverses [177 - Pernet (2006), 126 - Giorgi (2004)]).

3.6 Algorithmes par blocs sur architectures multi-cœurs

En théorie, les méthodes de Krylov par bloc donnent les meilleures bornes de complexités pour les matrices creuses. Néanmoins, en pratique ces méthodes sont ardues à mettre en œuvre, comme en témoignent les récents travaux de Pascal Giorgi notamment [109 - Eberly et al. (2006), 110 - Eberly et al. (2007)]. Tant que la mémoire n'est pas un obstacle, il est difficile par exemple de battre des méthodes directes et dans le cas contraire bien que ces méthodes soient les seules ressources disponibles, il faut parfois plusieurs semaines pour effectuer des calculs [B31, B22].

Un point central de la thèse en cours de Brice Boyer est l'étude pratique des méthodes de Krylov par bloc et un outil logiciel fondamental est la parallélisation de séquences de produits A^iU où A est creuse, et U est un bloc, dense, de vecteurs.

Il s'agit ici de coupler l'utilisation de multi-cœurs, CPU, et de cartes graphiques génériques, GPGPU. Les difficultés sont d'ordre logiciel : comment coupler efficacement différents formats de données et différentes ressources traitant différentes parties de la matrice creuse ? Là encore, des techniques adaptatives sont essentielles pour associer les formats de données aux différentes régions de la matrice, adapter les formats existants aux GPU et aux corps finis. Un des buts du projet //INBOX est d'initier une évolution majeure de la bibliothèque LINBOX vers le parallélisme [P40].

3.6.1 Produit matrice-vecteur creux

La bibliothèque FFSpMVgpu [S52] implémente l'opération $\mathbf{y} \leftarrow \alpha \mathrm{A}\mathbf{x} + \beta\mathbf{y}$ sur un corps premier.
La matrice A est creuse et $\mathbf{x}$ est un (ou un bloc de) vecteur(s). Nous tirons parti ou non de la présence d'une carte graphique supportant la technologie Cuda de Nvidia ainsi que des architectures multi-cœurs. Différentes représentations de matrice creuse, classiques ou non, sont fournies et permettent de s'adapter automatiquement aux données et aux ressources. La table 11 donne un exemple d'accélération potentielle.

matrice (Dim ; # non nuls) $\times 10^6$	m133.b3 (0.2M ; 0.8M)	GL7d15 (0.4M ; 6M)
LINBOX	0.2	0.3
FFSpMV 1 cœur	0.3	0.4
FFSpMV 8 cœurs	1.5	1.6
FFSpMV GTX	**6.6**	**8.3**

Table 11: GFlops sur l'opération $\mathbf{y} \leftarrow \mathrm{A}^{20}\mathbf{x} \mod 31$ sur Intel 8core 3.2GHz avec Nvidia GTX 280

- Nous avons également préparé une future représentation hybride (multi)GPU + (multi)CPU, utilisable dès que les environnements de calcul général sur carte graphique le permettront.
- Enfin, les résultats obtenus empiriquement doivent permettre de guider le développement de nouveaux algorithmes (de type bloc Krylov) pour petits corps finis, en collaboration avec Brice Boyer et Wayne Eberly.

3.6.2 Parallélisation de l'algorithme de Wiedemann

Pour la résolution de systèmes, les algorithmes par bloc de type Lanczos sont les plus performants. Au contraire, pour le calcul du rang, nous avons montré que les algorithmes de type Wiedemann sont à préférer [B29] et leur parallélisation est essentielle

pour accélérer les calculs de rang. En effet, pour de nombreuses applications en K-theory ou en topologie algébrique, ces méthodes sont la seule ressource et peuvent nécessiter plusieurs mois de calcul. C'est le cas par exemple pour les matrices GL7d16 à GL7d22, de la collection SIMC[♯] (*Sparse Integer Matrix Collection*), utilisées en K-theory, ou plus généralement dès que le nombre d'éléments non nuls dans la matrice dépasse la dizaine de millions.

Dans [B22], nous avons réalisé un premier prototype d'implémentation parallèle avec openMP dans laquelle nous utilisons la parallélisation à gros grain de [148 - Kaltofen et Lobo (1999), 198 - Turner (2006)] pour distribuer les vecteurs de projection à droite. Ensuite, l'algorithme des sigma-bases de [127 - Giorgi et al. (2003)] repose sur la multiplication de matrices et donc tire parti de la multiplication parallèle, mais sur de petites matrices. Dans [B18], nous passons à une parallélisation à grain plus fin pour les produits matrice vecteur, en utilisant les algorithmes de la section précédente, couplée à une terminaison anticipée parallèle. En outre, nous proposons une parallélisation de l'algorithme des sigma-bases grâce à un produit de matrices polynomiales parallèle. Au final, le rang est obtenu par évaluation/interpolation où chaque évaluation est indépendante. Cette parallélisation s'est avérée efficace par exemple pour calculer des homologies d'espaces de modules[♯].

3.7 Déterminant adaptatif sur les entiers

Le problème traité dans [B25] et dans la thèse de Anna Urbańska-Marzalek [199 - Urbańska-Marzalek (2010)] est d'accélérer le calcul du déterminant de matrices denses à coefficients entiers. Sur un corps, le calcul du déterminant se réduit à la multiplication de matrices par des factorisations récursives par blocs [141 - Ibarra et al. (1982)]. Avec des coefficients entiers, une approche naïve utilisant cet algorithme et des entiers en précision arbitraire induirait un grossissement des coefficients qui rendrait l'algorithme exponentiel. Ce problème de grossissement a été largement réduit par [163 - Moenck et Carter (1979), 101 - Dixon (1982)] pour la résolution de systèmes linéaires en $\mathcal{O}^{\sim}\left(n^3\right)$. Or, A. Storjohann a proposé un algorithme de complexité $\mathcal{O}^{\sim}\left(n^w\right)$ pour ce calcul [194 - Storjohann (2005)], donc dépendant du grossissement des coefficients entiers seulement par des facteurs logarithmiques. En pratique, nous avons toutefois montré que pour l'instant les facteurs logarithmiques et constants inclus dans cette complexité, sont très importants. Nous nous sommes donc attachés à réaliser un algorithme adaptatif qui calcule rapidement pour les matrices actuellement traitables. Pour cela, nous avons combiné plusieurs algorithmes. Celui de [70 - Abbott et al. (1999)] permet d'obtenir d'assez grands facteurs du dernier invariant de Smith, et donc du déterminant, par une résolution de système. Les travaux de [111 - Eberly et al. (2000)] indiquent qu'en général les invariants de Smith sont non triviaux pour seulement un nombre logarithmique d'entre eux et permettent donc de calculer la totalité du déterminant. Enfin, l'algorithme de [187 - Saunders et Wan (2004), 205 - Wan (2005)] permet de calculer les deux derniers invariants par deux résolutions de systèmes.

Nous avons commencé par améliorer les bornes de [111 - Eberly et al. (2000)] sur le nombre de facteurs invariants non triviaux attendus. Ensuite, nous avons étendu l'algorithme de [187 - Saunders et Wan (2004), 205 - Wan (2005)] à un nombre quelconque d'invariants car si celui-ci est relativement petit, cette méthode est plus performante que celle de [111 - Eberly et al. (2000)]. Au final, nous obtenons un algorithme de type Monte-Carlo et de complexité $\mathcal{O}^{\sim}\left(n^3\right)$ en moyenne, mais avec des constantes

[♯] `http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/simc.html`
[♯] Matrices `Mgn` de la *Sparse Integer Matrices Collection*.

et des exposants de facteurs logarithmiques réduits. Cet algorithme utilise néanmoins des produits de matrices et a donc pu bénéficier pour partie des implémentations de la section 3.1. En pratique, cet algorithme s'est également avéré d'un ordre de grandeur plus rapide que l'existant, comme démontré sur la figure 10. En outre, il est en général

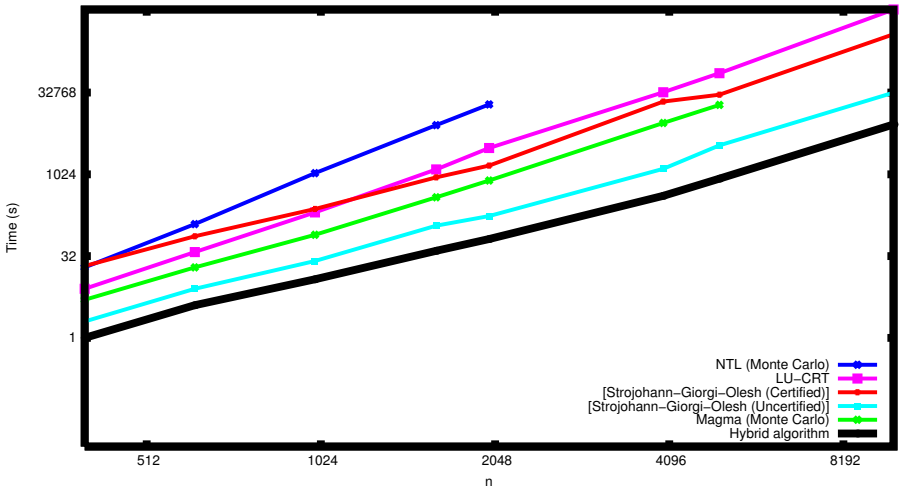


Figure 10: Temps de calcul du déterminant à coefficients entiers (sur des matrices aléatoires, entre 3000 et 8000 bits en sortie) sur un Itanium 1.3 GHz

possible de détecter quand cet algorithme va nécessiter trop de calculs et basculer alors par exemple sur l'algorithme [194 - Storjohann (2005)], afin de réduire la complexité asymptotique.

3.8 Calcul du polynôme caractéristique

Le calcul du polynôme caractéristique d'une matrice est un problème mathématique classique. Il est lié à celui de la forme normale de Frobenius (ou forme rationnelle canonique) qui est utilisée notamment pour les tests de similarité. Avec la multiplication de matrices classique, l'exposant de la complexité algébrique du calcul est optimale : plusieurs algorithmes nécessitent seulement n^3 opérations algébriques (à notre connaissance, le premier algorithme de ce type est dû à Danilevski [136 - Householder (1964), §24]). Avec la multiplication de matrices rapide, il en était tout autrement. Longtemps, le meilleur algorithme a été celui de [155 - Keller-Gehrig (1985)] en $n^\omega \log n$. Par ailleurs, le troisième algorithme de Keller-Gehrig a une complexité de n^ω, mais ne fonctionne que dans des cas génériques.

Enfin, les cas des matrices creuses et des matrices entières laissent encore beaucoup de place à l'optimisation et figurent parmi les problèmes importants du calcul exact [146 - Kaltofen (2000), Problème 3].

Nous avons donc travaillé, notamment avec Clément Pernet, sur ce problème. En particulier, nous avons tout d'abord introduit des produits de matrices dans des algorithmes de type n^3 pour les polynômes minimaux et caractéristiques, afin d'en améliorer les constantes de complexité et les vitesses pratiques [A13]. Nous avons également augmenté les cas traités par les algorithmes de type n^ω, jusqu'à la réduction finale du calcul du polynôme caractéristique au produit de matrices par [179 - Pernet et Storjohann (2007a)]. Nous avons ensuite donné des bornes fines sur les coefficients entiers des polynômes

minimaux et caractéristiques permettant de réduire les coûts de reconstruction dans le cas entier [A5].

3.8.1 Polynôme caractéristique dense par méthodes de Krylov

Un premier algorithme de coût n^3, mais assez efficace en pratique, repose sur le calcul probabiliste du polynôme minimal par méthodes de Krylov développé dans [176 - Pernet (2003), A13]. L'idée, donnée dans l'algorithme 19, est d'appliquer la décomposition $QLUP$ à la matrice de Krylov et de lire les coefficients du polynôme minimal (plus petite combinaison linéaire nulle des puissances de A appliquées à v) sur la dernière ligne de L (combinaison linéaire des colonnes de la matrice de Krylov $[A^i v]$.)

Algorithme 19 `MinPoly` : polynôme minimal de A et v

Entrées : – A une matrice $n \times n$ et v un vecteur
Sorties : – $P^{A,v}_{\text{min}}(X)$ le polynôme minimal de $(A^i v)_i$
1 : $K_{1\ldots n,1} = v$
2 : **Pour** $i = 1$ **jusqu'à** $\log_2(n)$ **Faire**
3 : $\quad K_{1\ldots n,2^i\ldots 2^{i+1}-1} = A^{2^{i-1}} K_{1\ldots n,1\ldots 2^i-1}$
4 : **Fin Pour**
5 : $(L,S,P) = \texttt{LSP}(K^T), k = \text{rank}(K)$
6 : $m = L_{k+1}.L^{-1}_{1\ldots k}$
7 : Retourner $P^{A,v}_{\text{min}}(X) = X^k - \sum_{i=0}^{k-1} m_i X^i$

Ensuite, les k premiers itérés de Krylov indépendants étant obtenus, on peut calculer une base de l'espace supplémentaire. Enfin, un appel récursif donne les facteurs restants, dans l'algorithme 20.

Algorithme 20 `LUK` : algorithme LU-Krylov

Entrées : – A une matrice $n \times n$
Sorties : – P^A_{char} le polynôme caractéristique de A
1 : Choisir aléatoirement un vecteur v.
2 : $P^{A,v}_{\text{min}} = \text{MinPoly}(A,v)$ de degré k ; $\{\left(\begin{bmatrix} L_1 \\ L_2 \end{bmatrix}, [S_1|S_2], P\right) = K_{A,v}$ sont calculés$\}$
3 : **Si** $(k = n)$ **Alors**
4 : $\quad$ Retourner $P^A_{\text{char}} = P^{A,v}_{\text{min}}$
5 : **Sinon**
6 : $\quad A' = PA^TP^T = \begin{bmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{bmatrix}$ où A'_{11} est $k \times k$.
7 : $\quad P^{A'_{22}-A'_{21}S_1^{-1}S_2}_{\text{char}}(X) = \texttt{LUK}(A'_{22} - A'_{21}S_1^{-1}S_2)$
8 : $\quad$ Retourner $P^A_{\text{char}} = P^{A,v}_{\text{min}} \times P^{A'_{22}-A'_{21}S_1^{-1}S_2}_{\text{char}}$
9 : **Fin Si**

Théorème 21

L'algorithme LU-Krylov calcule de manière déterministe le polynôme caractéristique d'une matrice $n \times n$ en n^3 opérations arithmétiques.

Un point à noter est que bien que le calcul du polynôme minimal soit probabiliste l'algorithme obtenu pour le polynôme caractéristique est lui déterministe : les polynômes minimaux obtenus sont toujours des facteurs des polynômes minimaux et l'algorithme de polynôme caractéristique ne s'arrête que quand le degré obtenu est n.

Ensuite, des algorithmes de complexité $n^\omega \log(n)$, puis finalement n^ω ont été obtenus par C. Pernet. Ces algorithmes sont également plus rapides en pratique à partir de tailles de matrices de quelques milliers. L'algorithme LU-Krylov reste donc néanmoins utile pour les plus petites matrices.

3.8.2 Multiplicités des facteurs du polynôme caractéristique creux

Dans le cas creux, l'histoire est différente. Le calcul du polynôme caractéristique est un des problèmes difficiles de l'algèbre linéaire [146 - Kaltofen (2000), Problème 3]. Nous avons proposé, dans [A11], une heuristique permettant de calculer ce polynôme en $n^{1.5}\Omega$ opérations arithmétiques où Ω est le nombre d'éléments non nuls de la matrice. L'idée est de calculer d'abord le polynôme minimal en $n\Omega$, puis de factoriser ce polynôme en polynômes irréductibles. Il ne reste plus alors qu'à calculer les multiplicités des facteurs dans le polynôme caractéristique.

Notre méthode repose sur les trois routines probabilistes suivantes, chacune de complexité $n\Omega$: le calcul du polynôme minimal par [208 - Wiedemann (1986)], le calcul du rang et du déterminant par préconditionnement [150 - Kaltofen et Saunders (1991)].

Pour une partie des facteurs de grand degré, on utilise le lemme suivant :

Lemme 22 [203 - Villard (2000)]

Soient U et V des matrices Toeplitz $n \times k$ et $k \times n$ aléatoires de plein rang. Soit g_k le polynôme minimal de $(A+UV)$. Alors avec une grande probabilité, $f_{n-k} = pgcd(g_k, P_{min;A})$, où f_{n-k} est le $(n-k)$ième facteur invariant de A.

Grâce à ce lemme, on peut calculer quelques-uns des facteurs invariants composant le polynôme caractéristique, pour un coût de $n\Omega$ pour chaque facteur.

Au final, en utilisant ce dernier lemme, il est possible de s'assurer qu'il ne reste que $\sqrt{n}$ multiplicités inconnues en au plus $\sqrt{n}$ calculs de polynômes minimaux (sinon plus de $\sqrt{n}$ facteurs invariants posséderaient plus de $\sqrt{n}$ facteurs irréductibles et le degré total dépasserait n).

Ensuite, nous utilisons le lemme suivant :

Lemme 23

Si le polynôme minimal et le polynôme caractéristique s'écrivent $P_{min;A} = \prod P_1^{\alpha_1} \dots P_k^{\alpha_k}$ *et* $P_{char;A} = \prod P_1^{\beta_1} \dots P_k^{\beta_k}$ *avec les* P_i *irréductibles, alors*

$$\beta_i = \frac{n - \mathit{rang}(P_i^{\alpha_i}(A))}{\mathit{degré}(P_i)}$$

Le calcul d'une multiplicité β_i par cette méthode nécessite $d_i\alpha_i\Omega$ opérations et est donc à réserver aux seuls facteurs de faibles degré et multiplicité dans le polynôme minimal.

Au final, il nous reste moins de $\sqrt{n}$ multiplicités, toutes pour des facteurs au moins non linéaires.

Nous utilisons alors une technique de calcul d'index. Informellement, le polynôme caractéristique évalué à une certaine valeur λ donne l'équation suivante d'inconnues les β_i et où le second membre est obtenu par des calculs de déterminant, chacun en $n\Omega$:

$$\prod_{j=1}^{k} P_j^{\beta_j}(\lambda) = \det(\lambda I - A). \tag{17}$$

Dans un corps fini $\mathtt{GF}(q)$, en prenant le logarithme discret de cette équation, on obtient une équation linéaire :

$$\sum_{j=1}^{k} \beta_j \log_g(P_j(\lambda)) \equiv \log_g(\det(\lambda I - A)) \mod q - 1, \tag{18}$$

On construit alors un système linéaire de $\sqrt{n}$ de ces équations, que l'on résout en $\sqrt{n}^{\omega} < n^{3/2}$ opérations.

- Cette méthode n'est qu'une heuristique puisque rien ne garantit que le système en question sera de plein rang. En pratique on a néanmoins une marge assez importante en nombre d'équations possibles (par rapport à la complexité globale $\sqrt{n}^{\omega}$) qui permet d'ajouter des lignes jusqu'à l'obtention d'un système de plein rang, dans tous les cas testés, et avec une bonne efficacité.

- Nous conjecturons que pour des P_i de degré au moins 2 et en choisissant $\sqrt{n}$ λ_i au hasard dans un corps fini suffisamment grand, il est possible d'obtenir un système $[\log_g(P_j(\lambda_i))]$ de plein rang avec une grande probabilité.

3.8.3 Bornes sur les coefficients des polynômes minimaux et caractéristiques entiers

Pour traiter le cas entier, un outil important est d'utiliser le théorème des restes chinois. Il est alors indispensable d'avoir des bornes fines sur les coefficients des polynômes minimaux et caractéristiques. Nous avons donné dans [A5] les bornes suivantes qui améliorent par exemple celles de [124 - Giesbrecht et Storjohann (2002), Lemme 2.1] et sont complémentaires des bornes obtenues sur la matrice polynomiale $xI - A$ [130 - Goldstein et Graham (1973), 113 - Emiris et Pan (2005)] :

Lemme 24

Soit $A \in \mathbb{C}^{n\times n}$, avec $n \geq 4$, de coefficients bornés en valeur absolue par $B > 1$. Les coefficients de $P_{char;A}$, le polynôme caractéristique de A sont dénotés c_j, $j = 0..n$ et $||P_{char;A}||_\infty = max\{|c_j|\}$. Alors

i. $\log_2(||P_{char;A}||_\infty) \leq \frac{n}{2}\left(\log_2(n) + 2\log_2(B) + 0.21163175\right)$

ii. $||P_{char;A}||_\infty \leq max_{i=0..\frac{-1+\sqrt{1+2\delta B^2 n}}{\delta B^2}} \binom{n}{i}\sqrt{(n-i)B^2}^{(n-i)}$ *où* $\delta \approx 5.418236$.
En outre, le coût du calcul de cette borne est $\frac{\sqrt{n}}{B}$.

La borne de Goldstein-Graham sur les coefficients du polynôme caractéristique, $det(X \cdot I - A)$, donne :

$$\sqrt{\sum_k |c_k|^2} \leq \prod_i \sqrt{(1+|A_{ii}|)^2 + \sum_{j\neq i}|A_{ij}|^2} \leq \sqrt{(nB^2+2B+1)^n} \tag{19}$$

Par exemple, sur la matrice 2.1 de [A5], reprise ci-dessous (20), et de polynôme caractéristique $X^5 - 5X^4 + 40X^2 - 80X + 48$, on obtient une valeur approchée de 181.02 pour la racine carrée de la somme des valeurs absolues des carrés des coefficients du polynôme caractéristique (qui vaut approximativement 101.64).

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 \end{bmatrix} \tag{20}$$

Différemment, la borne du lemme 24 donne 80.67 comme plus grande valeur absolue d'un coefficient du polynôme (qui vaut 80). Néanmoins, avec $B > 1$, la borne de Goldstein-Graham est meilleure que celle obtenue dans la première partie du lemme 24, dès que $log_2(1 + \frac{2B+1}{nB^2}) < \left(\frac{5}{6}\log_2(5) - \frac{2}{3}\log_2(6)\right) \approx 0.21163175$, i.e. dès que $n > \frac{18}{5ln(5)-4ln(6)} \approx 20.451$.

Enfin, nous avons également donné la borne suivante sur les coefficients du polynôme minimal. L'idée est qu'il est possible d'obtenir des bornes assez fines sur le spectre de la matrice par exemple avec les méthodes de type Gershgorin ou Cassini [201 - Varga (2004)]. Notre borne dépend alors du spectre matrice et du degré du polynôme minimal au lieu de la taille des coefficients initiaux et de la dimension de la matrice. Pour des polynômes minimaux de petit degré l'amélioration est conséquente.

Lemme 25

Soit $A \in \mathbb{C}^{n\times n}$ de spectre borné par $\beta \geq 1$. Si $P_{min;A}(X) = \sum_{k=0}^{d} m_i X^i$. Alors

$$\forall i,\ |m_i| \leq \begin{cases} \beta^d & \text{si } d \leq \beta \\ min\{\sqrt{\beta d}^d\ ;\ \sqrt{\frac{2}{d\pi}}2^d\beta^d\} & \text{sinon} \end{cases}$$

Outre pour le calcul des polynômes caractéristiques et minimaux, ces bornes sont utiles également par exemple pour le calcul de la forme normale de Smith par la méthode Valence développée durant mon doctorat [T69, A17]. Par exemple, le calcul de polynôme minimal le plus difficile de [A8], est celui de la matrice `n4c6.b12`, de dimension 25605×69235, dont le degré du polynôme minimal de la matrice de Wishart $A^T A$ est seulement 827 et dont le spectre est borné par 117. Le lemme 25 donne un gain de 5% par rapport aux bornes précédentes ce qui aurait permis l'économie de 23 projections modulaires et une heure de calcul distribué.

3.8.4 Calcul du polynôme caractéristique entier

Il existe plusieurs algorithmes pour calculer le polynôme caractéristique d'une matrice à coefficients entiers. Une première idée est de réaliser le calcul de manière algébrique sur l'anneau des entiers [71 - Abdeljaoued et Malaschonok (2001)], avec des divisions exactes [84 - Berkowitz (1984), 145 - Kaltofen (1992), 151 - Kaltofen et Villard (2004)]. La meilleure borne de complexité est due à [151 - Kaltofen et Villard (2004), §7.2], en $\mathcal{O}^{\sim}\left(n^{2.697263}\right)$. Nous avons proposé dans [A13] une alternative, efficace en pratique, calculant les multiplicités après avoir calculé le polynôme minimal :

Algorithme 26 `CIA` : polynôme caractéristique sur les entiers

Entrées : – $A \in \mathbb{Z}^{n \times n}$, dense ou creuse.
Sorties : – $P_{char;A}$, le polynôme caractéristique de A.
1 : $P_{min;A}$ = `IMP`(A, η) via [A8, §3] ou `CRT(MinPoly)`(A, η)
2 : Factoriser $P_{min;A}$ sur les entiers, e.g. par remontée de Hensel.
3 : Choisir un nombre premier p aléatoirement.
4 : Calculer $P_{char;A;p}$ le polynôme caractéristique de A mod p.
5 : **Pour tout** P_i facteur irréductible de $P_{min;A}$ **Faire**
6 : $\overline{P}_i \equiv P_i \mod p$.
7 : Retrouver β_i la multiplicité de $\overline{P}_i$ dans $P_{char;A;p}$.
8 : **Si** $\beta_i == 0$ **Alors** Retourner "Échec". **Fin Si**
9 : **Fin Pour**
10 : Calculer $P_{char;A} = \prod P_i^{\beta_i} = X^n - \sum_{i=0}^{n-1} a_i X^i$.
11 : **Si** $(\sum \beta_i \text{degree}(P_i) \neq n)$ or $(Trace(A) \neq a_{n-1})$ **Alors**
12 : Retourner "Échec".
13 : **Fin Si**
14 : Retourner $P_{char;A}$.

Cet algorithme est de type Monte-Carlo, il peut retourner une valeur erronée, avec probabilité connue, malgré les différents points de vérification inclus. Par ailleurs, [192 - Storjohann (2000)] a proposé de remplacer la factorisation de polynômes sur les entiers par un calcul de bases pgcd-libres, si cette factorisation est trop coûteuse. La borne de complexité obtenue n'est pas meilleure que celle de [151 - Kaltofen et Villard (2004)], mais en pratique, l'algorithme est très rapide sur les tailles de matrices calculables actuellement.

3.9 Espaces de matrices pour les codes correcteurs

Les codes correcteurs d'erreur permettent de corriger les erreurs de transmission numérique de l'information. Les propriétés recherchées sont par exemple des compromis

entre quantité de redondance ajoutée et capacité de correction à taux d'erreur donné. Dans les codes par blocs, cette redondance s'ajoute par blocs d'information de taille fixe. Afin de pouvoir décoder et corriger facilement, on utilise souvent des codes linéaires dans lesquels les blocs sont des vecteurs d'un espace vectoriel et les mots de code forment un sous-espace vectoriel. Si à la réception, le bloc reçu n'est pas un élément de ce sous-espace alors des erreurs de transmissions sont détectées et une correction automatique peut avoir lieu, par exemple en projetant le mot erroné reçu sur le sous-espace des mots corrects. Dans ce cadre, nous avons étudié des familles de codes dans des espaces vectoriels de matrices où les sous-espaces de mots de codes sont des matrices avec des propriétés de rang particulières. En effet les propriétés de rang sont très sensibles aux variations, mêmes minimes, et très rapides à calculer sur des tailles de matrices intéressantes comme tailles de blocs d'information. Nous avons trouvé des espaces particuliers prometteurs et également donné des bornes sur les dimensions possibles dans différents corps [A3, T64]. Enfin, nous avons utilisé des techniques similaires pour avancer la classification des semi-corps [T66].

3.9.1 Sous-espaces de matrices de rang contraint

Par exemple, nous avons obtenu des bornes sur la dimension maximale d'un sous-espace, sur des corps finis, de matrices dont le rang est contraint (sauf pour la matrice 0). Nous donnons ces bornes dans la table 12, les bornes inférieures étant constructives.

Type	Matrice	Contrainte	Max dim.
-	$m \times n$	rang $\geq r$	$\leq n(m-r+1)$
-	$m \times n$	rang $= r$	$\geq n$
-	$m \times n$	rang $= r$	$\leq m+n-r$
-	4×5	rang $= 3$	6 sur $\mathtt{GF}(2)$
-	5×5	rang $= 4$	6 sur $\mathtt{GF}(2)$
Anti-sym.	$n \times n$	rang $\geq 2r$	$\geq n(n-2r+1)/2$
Anti-sym.	$n \times n$	rang $= 2r$	$\leq 2n-2r-1$
Anti-sym.	$n \times n$	rang $= 2$	$\leq n-1$
Anti-sym.	$3n \times 3n$	rang $= 2n$	$\geq 3n$
Sym.	$n \times n$	rang $= 2$	$\leq n-1$
Sym.	$n \times n$	rang $= 3$	≤ 3
Sym.	$n \times n$	rang $= 2r+1$	$\leq n$

Table 12: Dimensions maximales d'un sous-espace de matrices sur un corps fini

Les preuves utilisent principalement la théorie des caractères d'un groupe fini de Delsarte [97 - Delsarte (1978), 98 - Delsarte et Goethals (1975)], de l'algèbre linéaire et des explorations intensives et exhaustives, ad-hoc ou avec M4RI, des petites dimensions.

- Ces bornes sont relativement fines puisque la latitude est souvent de l'ordre de $n-r$ et que des espaces de dimension supérieure à n ont été trouvés, contredisant plusieurs conjectures à ce sujet. Ainsi, l'exemple 1 de [A3] est en contradiction avec [82 - Beasley (1999), Conjecture 5].

- Dans le cas symétrique, les expérimentations sur de petits cas semblent cependant indiquer que les sous-espaces de matrices symétriques de rang impair ne peuvent être de dimension supérieure au rang.

3.9.2 Centralisateur et semi-corps

Un semi-corps fini est un groupe abélien muni d'une multiplication distributive et sans diviseur de zéro mais pas forcément commutative ni associative. Récemment des relations entre certains codes linéaires et des semi-corps ont été établies (voir par exemple [153 - Kantor et Williams (2004)]) et l'exploration de ces ensembles, en particulier en caractéristique 2, a été entamée. Plus précisément, la construction de semi-corps d'ordre donné peut se ramener à la construction de sous-espaces de matrices inversibles dans des corps finis [100 - Dempwolff (2008), 185 - Rúa et al. (2009)].

Nous nous sommes attelés à la classification des semi-corps d'ordre $3^5 = 243$ et $5^4 = 625$, et donc à des sous-espaces de $GL(5,3)$ et $GL(4,5)$ contenant l'identité. Il s'agit alors principalement de déterminer si deux sous-espaces sont équivalents. Nous avons donné un nouvel algorithme de complexité $q^d C^* dn^\omega$ testant si deux sous-espaces de dimension d de matrices inversibles $n \times n$ sur $\mathtt{GF}(q)$ sont équivalents, où C^* est la taille du plus petit centralisateur sur $GL(n,q)$ des matrices des sous-espaces considérés. Une fois la taille du plus petit centralisateur déterminée à partir des facteurs invariants de la matrice [132 - Green (1955), Lemme 2.4], il est possible de déterminer ce centralisateur en temps C^* [156 - Kunkle et Cooperman (2009)], ou par résolution d'un système linéaire de taille n^2.

L'idée principale est de ramener le test d'équivalence à un test de similarité en tirant parti de la structure particulière des espaces considérés. En effet, deux ensembles de matrices $\mathcal{S}$ et $\mathcal{T}$ sont équivalents si et seulement si il existe deux matrices inversibles L et R telles que la fonction $f : M \hookrightarrow f(M) = L \cdot M \cdot R$ est une bijection de $\mathcal{S}$ dans $\mathcal{T}$. Alors, comme les deux ensembles doivent contenir l'identité, il doit existe $T_* \in \mathcal{T}$ telle que $L \cdot T_* \cdot R = I$. Il s'en suit que $R = T_*^{-1} L^{-1}$ et que $f(M) = L \cdot M T_*^{-1} \cdot L^{-1}$.

Ainsi, les deux espaces sont équivalents si et seulement si $\mathcal{S}$ et $\mathcal{T} T_*^{-1}$ sont similaires. Ensuite, la similarité se ramène à des calculs de formes normales de Frobenius.

Plus précisément nous utilisons les idées suivantes :

- Les ensembles de formes de Frobenius de deux espaces similaires sont identiques.
- Nous utilisons par la forme de Frobenius pour obtenir des matrices de passage : nous distinguons F_*, la forme de Frobenius de S_*, une des matrices de $\mathcal{S}$, et V_* le changement de base associé, tel que $F_* = V_* S_* V_*^{-1}$. Comme $\mathcal{S}$ et $\mathcal{T} T_*^{-1}$ sont similaires, il existe une matrice $T_j \in \mathcal{T}$ telle que la forme de Frobenius de $T_j T_*^{-1}$ est également F_*. Ainsi, il existe une matrice de passage U_j telle que :

$$F_* = V_* S_* V_*^{-1} \tag{21}$$
$$F_* = U_j T_j T_*^{-1} U_j^{-1} \tag{22}$$

  Cette relation permet de définir un candidat pour la similarité :

$$L = V_*^{-1} U_j \tag{23}$$
$$R = T_*^{-1} U_j^{-1} V_* \tag{24}$$

- Enfin, nous utilisons le fait que toute matrice Z du centralisateur de S_* dans le groupe des matrices inversibles, induit une nouveau candidat ZL pour la similarité.

Globalement nous avons remplacé une recherche exhaustive parmi les matrices inversibles (donc potentiellement q^{2n^2} matrices) par une recherche parmi les matrices

d'un des sous-espaces considérés (de taille q^n) et parmi le centralisateur d'une des matrices de l'autre sous-espace (souvent de taille q^n).

Le test d'un couple de matrices de d'équivalence est simplement la recherche d'associations sur une base de l'ensemble, comme présenté dans l'algorithme 27. Ensuite les idées précédentes pour le test d'équivalence d'espaces sont résumées dans l'algorithme 28.

Algorithme 27 TestCandidat : dn^ω

Entrées : – Deux $n \times n$ matrices inversibles $Left$ et $Right$.
– Deux sous-espaces S et T, donnés par une base, de dimension d, contenant l'identité, de matrices inversibles $n \times n$.
Sorties : – Soit $Left \cdot S \cdot Right = T$, soit "Non".

Pour Toute matrice A de la base de S **Faire**
  Si $Left \cdot A \cdot Right \notin T$ **Alors** Retourner "Non"; **Fin Si**
Fin Pour
Retourner "Oui";

Algorithme 28 TestEquivalence : $q^d C^* dn^\omega$

Entrées : – Deux sous-espaces S et T, donnés par une base, de dimension d, contenant l'identité, de matrices inversibles $n \times n$.
Sorties : – Les matrices $Left$ et $Right$ si les espaces sont équivalents, "Non" sinon.

{Formes Frobenius, puis taille par la formule [132 - Green (1955), Lemme 2.4] : $q^d n^\omega$}
Calculer C^* la taille du plus petit centralisateur de $GL(n,q)$ de matrices de S et T;
À renommage près, soit S^* la matrice de plus petit centralisateur.

$\{q^d n^\omega\}$

Soit $\mathcal{F}_1 := \{\}$;
Pour Toutes les matrices C de S **Faire**
  $\mathcal{F}_1 := \mathcal{F}_1 \cup \{\text{Forme de Frobenius de } C\}$;
Fin Pour
Soit F^* et V^* la forme de Frobenius et la matrice de passage de S^*;

$\{q^{2d} n^\omega\}$

Soit $\mathcal{R} := \{\}$;
Pour Toutes les matrices A_i de T **Faire**
  $\mathcal{R}_i := \{\}$; $\mathcal{F}_{2,i} := \{\}$;
  Pour Toutes les matrices B_j de T **Faire**
    $(F_{i,j}, U_{i,j})$:=FormeFrobenius-ChangeBase$(B_j A_i^{-1})$;
    $\mathcal{F}_{2,i} := \mathcal{F}_{2,i} \cup \{F_{i,j}\}$;
    Si $F_{i,j} == F^*$ **Alors** $\mathcal{R}_i := \mathcal{R}_i \cup \{(A_i, U_{i,j})\}$ **Fin Si**
  Fin Pour
  Si $\mathcal{F}_1 == \mathcal{F}_{2,i}$ **Alors** $\mathcal{R} := \mathcal{R} \cup \mathcal{R}_i$; **Fin Si**
Fin Pour

Si $\mathcal{R} == \emptyset$ **Alors** Retourner "Non"; **Fin Si** $\{\forall i,\ \mathcal{F}_1 \neq \mathcal{F}_{2,i}\}$

$\{q^d C^* dn^\omega\}$

Pour Toutes les paires de matrices (A, U) de R **Faire**
  $X := V^* \cdot U^{-1}$;
  Pour Toutes les matrices Y du centralisateur dans $GL(n,q)$ de S^* **Faire**
    $Left := Y \cdot X$;
    $Right := (Left \cdot A)^{-1}$;

24 : **Si** TestCandidat$(Left, Right, S, T)$ **Alors** Retourner $(Left, Right)$; **Fin Si**
25 : **Fin Pour**
26 : **Fin Pour**
27 : Retourner “Non”;

Ce test d'équivalence nous a par exemple permis également de démontrer que si il y a 12 classes de conjugaison d'espaces de matrices 3×3 de rang 2 dans $\mathbb{Z}/2\mathbb{Z}$ [90 - Boston (2010)], il n'y a au contraire qu'un seul espace à équivalence près. En particulier, les exemples 1 et 4 de [82 - Beasley (1999)] présentés comme inéquivalents sont en fait reliés par les matrices de passage suivantes :

Avec $a = x, b = w + y, c = x + y + z, d = y,$ on obtient

$$\left\{ \begin{bmatrix} w & 0 & y \\ z & w+x & 0 \\ 0 & y+z & x \end{bmatrix}, w, x, y, z \in \mathbb{Z}/2\mathbb{Z} \right\} =$$
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \left\{ \begin{bmatrix} a & c & c \\ d & a+b & c \\ d & d & b \end{bmatrix}, a, b, c, d \in \mathbb{Z}/2\mathbb{Z} \right\} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (25)$$

- L'algorithme 28, associé au calcul efficace de la forme de Frobenius [108 - Eberly (2000)], accélère alors la méthode de [185 - Rúa et al. (2009)] pour la classification des semi-corps et nous permet donc d'envisager d'attaquer les ordres 243 ou 625.
- En effet, l'algorithme 28 fait principalement intervenir des calculs de polynômes caractéristiques et de formes de Frobenius, variantes de l'algorithme 20. En pratique, C^* est en général inférieur à q^n et donc la complexité attendue est d'ordre $q^{d+n}dn^\omega$, soit de l'ordre de seulement 2^{25} calculs, alors que le nombre de matrices inversibles est de l'ordre de 2^{39}.
- Les premières matrices et les premières colonnes des matrices étant fixées, la recherche s'effectuera ensuite sur un espace de taille $3^{20} \approx 2^{32}$.
- De même, la classification des semi-corps d'ordre 2^7 devrait être envisageable à court terme.

4 Algorithmes symboliques-numériques

Un problème ouvert majeur en calcul scientifique est de prendre en compte un bruitage des coefficients ; nous voulons nous y intéresser en calcul matriciel. Dans de nombreux cas, les méthodes numériques classiques (itérations de Newton, de type Krylov, ...) peuvent ne pas converger ou converger trop lentement et il devient difficile de préserver la structure sous-jacente (matrices de résultants, etc.) durant le processus. Il est alors crucial de développer des algorithmes symboliques-numériques, des algorithmes à précision arbitraire et des algorithmes de certification de résultat en calcul exact. Ce type de calcul symbolique-numérique est indispensable quand des quasi-singularités de systèmes d'équations découlent de propriétés physiques, comme par exemple le déplacement d'une plate-forme de Gough-Stewart ; des matrices de Toeplitz (plus proches matrices singulières) sont utilisées en synthèse d'images ; des méthodes de calcul hybride permettent d'analyser des systèmes dynamiques mal conditionnés en biologie.

Ces dernières années, les progrès considérables de l'algorithmique linéaire exacte (les travaux de A. Storjohann en 2003 puis E. Kaltofen et G. Villard en 2004 sur le déterminant entier, ceux de C. Pernet depuis 2004 et A. Steel en 2007 sur le polynôme caractéristique et la forme échelon ; notamment) ont permis au domaine d'atteindre en pratique des vitesses de calcul comparables au numérique. Il s'agit alors de définir un nouveau paradigme de calcul tirant parti de ces dernières avancées. Qu'elle soit numérique ou exact, l'arithmétique machine est très rapide par rapport au calcul à précision variable (calcul par intervalles, précision arbitraire, etc.). Il faut donc créer le calcul de demain ni formel, ni numérique, adapté aux nouvelles architectures (majoritairement multi-cœurs, GPU, etc). Par exemple, la résolution de systèmes linéaires raides en calculs d'éléments finis nécessite des méthodes du type $A = A_1 + \epsilon A_2$ [188 - Schatzman (2005), P45] où la résolution sur A_1 est exacte, puis raffinée numériquement avec le résidu A_2. Un autre exemple est la certification de résolutions numériques pour garantir des propriétés sur le résultat des calculs [204 - Villard (2007)].

Il s'agit alors dans un premier temps d'étendre les techniques binaires récentes au cas rationnel, avec la mise en place de préconditionneurs et l'utilisation des techniques récentes de reconstruction rapide. Il sera ensuite nécessaire de trouver un paradigme adapté à un calcul hybride numérique/formel combinant efficacement par exemple les techniques d'arithmétique d'intervalles [172 - Nguyen et Revol (2008)], de précision arbitraire [99 - Demmel et al. (2006)], de certification [204 - Villard (2007)], etc.

Nous présentons dans la suite des applications où les méthodes symboliques-numériques sont indispensables pour obtenir des solutions. Tout d'abord dans le cadre des systèmes dynamiques hybrides, où une dynamique continue est guidée par des automates discrets, nous montrons qu'une approche de linéarisation par morceaux permet de donner des solutions exactes au problème approché et par la même une bonne stabilité de l'approximation. Par ailleurs, nous avons adapté les techniques de calcul de polynôme caractéristique au calcul de formes normales de Kalman utilisées pour le contrôle de ces systèmes. Enfin, nous utilisons des techniques modulaires et rationnelles d'algèbre linéaire pour obtenir des développements limités numériques utilisés pour la découverte de motifs dans l'ADN.

4.1 Algorithmes pour le contrôle symbolique/numérique de systèmes dynamiques affines

Nous considérons le système dynamique général non-linéaire dont l'état est décrit par les solutions de l'EDO suivante :

$$\begin{cases} \dot{X}(t) = f(X(t), u(t)) \\ X(0) = X_0 \end{cases} \tag{26}$$

Nous présentons un algorithme générique pour contrôler le système (26) d'un état initial X_0 au temps $t = 0$ à un état final X_f à un temps non spécifié t_f où les contrôles admissibles sont des fonctions mesurables bornées à valeurs dans un sous-ensemble U_m de $\mathbb{R}^m$ et où la fonction coût suivante est minimisée :

$$J(X_0, u) = \int_0^{t_f} l(X(t), u(t))dt.$$

L'approche de la thèse de Aude Rondepierre [183 - Rondepierre (2006)] dans le cadre des projets CC [P48] et FCSDH [P51] est d'utiliser des systèmes hybrides pour résoudre ce problème : la dynamique complexe est remplacée par des approximations affines par morceaux, solvables analytiquement [128 - Girard (2002)]. La séquence des modèles affines forme alors une séquence d'états d'un automate hybride. Étant donnée une séquence optimale d'états, nous sommes alors capables de traverser l'automate jusqu'à la cible en assurant une optimalité locale [T67].

Un outil important est donc la résolution du problème linéaire suivant : $\dot{X}(t) = AX(t) + Bu(t)$. Nous avons proposé dans [A14] des algorithmes combinant des modules numériques et symboliques pour en particulier donner un sous-ensemble garanti du domaine contrôlable. Plus précisément, cette garantie est obtenue grâce à un nouvel algorithme *exact* calculant les frontières des domaines où le contrôle optimal est constant. Des méthodes efficaces pour calculer la forme canonique de Kalman de A et B, caractérisant la contrôlabilité, et pour déterminer les solutions optimales sont également présentées. L'algorithme pour la forme de Kalman a ensuite été amélioré, pour donner une borne de complexité de $\mathcal{O}^{\sim}(n^{\omega})$, par C. Pernet et A. Rondepierre en collaboration avec G. Villard [178 - Pernet et al. (2005)].

Ensuite, la thèse de A. Rondepierre donne le cadre théorique général de la résolution de problèmes de contrôle optimal par des méthode hybrides, en particulier sur l'évaluation de la qualité de l'approximation affine par morceaux, ainsi que des algorithmes effectifs pour le calcul à la volée d'un maillage de l'espace état-contrôle et la manipulation efficace du modèle hybride, *en toute dimension*. Elle donne également deux approches pour la contrôlabilité : par la définition d'un principe du maximum hybride et par la résolution des équations d'Hamilton-Jacobi-Bellman. Pour chacune de ces approches, la convergence des solutions du problème hybride vers les solutions du problème initial est étudiée et l'exploitation des conditions nécessaires d'optimalité fournies par le principe du maximum hybride permettent notamment de déduire la structure générale des contrôles optimaux sous forme de boucles de rétroaction affines par morceaux.

- La difficulté restante est de déterminer la séquence optimale d'état de l'automate. Différents algorithmes permettent de s'en sortir : par exemple une exploration locale des ensembles atteignables, combinée à un retour-sur-traces (*backtracking*), permet de "viser" la cible à atteindre. Des études plus fines de ces algorithmes restent à faire.

- C. Pernet a réduit le calcul de la forme normale de Frobenius à la multiplication de matrices en transformant les algorithmes de Keller-Gehrig pour calculer des formes intermédiaires décalées (*k-uniform shifted form* [180 - Pernet et Storjohann (2007b)]). Il semble possible d'appliquer ces transformations également à la forme de Kalman pour aller vers une complexité de seulement n^ω.

4.2 Modélisation par système hybride du potentiel électrique du neurone

Au sein du projet CNRS SQUASH [P50], nous avons proposé dans [B27] un système hybride modélisant le potentiel électrique émis par un neurone en réponse à un courant continu externe. Expérimentalement, Hodgkin et Huxley [135 - Hodgkin et Huxley (1952)] ont construit un système non-linéaire de dimension 4 pour simuler cette activité. Notre idée est d'utiliser une nouvelle approximation affine par morceaux comme modèle hybride de la dynamique de Hodgkin-Huxley en combinant les approches de FitzHugh-Nagumo [123 - Gerstner et Kistler (2002), §3.1.1] et de Tonnelier [196 - Tonnelier (2001), §1.4.3]. Nous avons proposé une alternative aux interpolations multi-dimensionnelles définies par A. Girard [129 - Girard (2004)], par l'utilisation de linéarisation par morceaux de représentations implicites des solutions. Après réduction de modèle à deux dimensions, nous avons donc obtenu un automate hybride en 2D de dynamique

$$\begin{cases} \frac{dv}{dt} = \tilde{p}(v) - w + I \\ \\ \frac{dw}{dt} = bv - \tilde{\chi}(w) \end{cases} \tag{M}$$

où $\tilde{p}(v)$ et $\tilde{\chi}(w)$ sont affines par morceaux.

Ainsi, ce sont à la fois les simulations et l'analyse de notre nouveau modèle qui montrent les deux caractéristiques du comportement du neurone :

- le potentiel d'excitabilité.
- le cycle limite correspondant à une stimulation périodique.

En outre, nous avons formellement démontré que l'intensité de courant appliquée est un paramètre de bifurcation du système linéarisé.

- Avec cette étude de cas, nous avons montré que des systèmes hybrides *continus* et affines par morceaux peuvent être utilisés pour calculer analytiquement des portraits de phase de systèmes dynamiques complexes sans perte de propriétés intrinsèques.
- Nous pouvons automatiquement calculer les points fixes ou les invariants de stabilité, mais les cycles limites nécessitent parfois des entrées manuelles. Une combinaison de différents types de linéarisations pourrait permettre de la multi-résolution et une analyse semi-automatique.

4.3 Reconstruction rationnelle pour la distribution de motifs dans l'ADN

La distribution de motifs dans des séquences aléatoires générées par des sources markoviennes a des applications dans une large gamme de domaines, de l'assurance à la bioinformatique. Dans ce dernier domaine, une application courante est la détection de motifs d'intérêt dans les séquences biologiques que sont l'ADN ou les protéines. Cette approche a permis par exemple la reconstruction de signaux biologiques (motifs CHI, signatures PROSITE) tout comme l'identification de nouvelles propriétés fonctionnelles (motifs régulateurs en régions hautes, sites de liaisons, etc.). Dans

[T65], nous obtenons un plongement optimal de chaîne de Markov d'un problème de motifs dans des séquences ADN par des automates finis déterministes minimaux [174 - Nuel (2010)].

Dans ce cadre, la fonction génératrice du nombre d'occurrences d'un motif donné, N_ℓ sur une séquence de taille ℓ, peut être explicitée par : $g_\ell(y) = \sum_{k=0}^{\ell} P(N_\ell = k) y^k$.

Nous voulons calculer des portions de ce développement autour d'un nombre d'occurrences donné, i.e. les coefficients $g_{\ell,k} = P(N_\ell = k)$ pour ℓ et $k \in [\mu, \nu]$ donnés, dans un petit intervalle $[\mu, \nu]$, mais où ℓ est potentiellement grand.

Cette fonction génératrice peut être obtenue comme solution du système linéaire suivant, où les matrices P, Q et les vecteurs u et v sont issus des proportions connues d'éléments dans la séquence d'ADN :

$$\begin{aligned} G(y,z) &= u^T \times (I - z(P + yQ))^{-1} \times v \\ &= \sum_{\ell=0}^{\infty} \left(u^T (P+yQ)^\ell v\right) z^\ell = \sum_{\ell=0}^{\infty} g_\ell(y) z^\ell \end{aligned}$$

Comme ℓ est grand, pour calculer les coefficients qui nous intéressent, il faut éviter de calculer la totalité du développement de Taylor de la fraction rationnelle solution.

L'idée est donc d'utiliser le *High-order lifting* de A. Storjohann [114 - Fiduccia (1985), 193 - Storjohann (2002)] pour atteindre rapidement les grands ordres et utiliser des interpolations par restes chinois et la reconstruction rationnelle sur y et z.

Plus précisément, si $G(y,z) = \frac{B(y,z)}{A(y,z)}$ avec $B, A \in \mathbb{Q}[y,z]$, nous obtenons dans un premier temps les polynômes exacts B et A de la manière suivante :

i. Calcul de la série en z modulo plusieurs nombres premiers de la taille du mot machine et modulo plusieurs points d'évaluation y_i.

ii. Reconstruction rationnelle univariée en z pour chaque point d'évaluation, sur chaque corps premier.

iii. Interpolation des polynômes en y sur chaque corps premier.

iv. Restes chinois et reconstruction rationnelle des coefficients rationnels.

On obtient alors une borne de complexité de

$$d^3\Omega$$

pour le calcul de cette première phase où d est le degré des polynômes A et B et Ω le nombre total d'éléments non nuls des matrices creuses P et Q.

On utilise ensuite la remontée de Storjohann ou directement la méthode de Fiduccia pour obtenir les coefficients de Taylor de haut degré de la fraction $BA^{-1}(z)$ en arithmétique modulo y^ν, et avec des coefficients flottants pour éviter un grossissement des coefficients. Au final, on obtient une borne de complexité

$$\log(\ell) d^2 k^2 \ \left(\text{ou } \log(\ell) d^{1+\epsilon} k^{1+\epsilon} \text{avec multiplication rapide de polynômes}\right)$$

pour le calcul de cette deuxième phase. Cela améliore par exemple l'algorithme utilisé par [173 - Nicodème et al. (2002), Algorithme 8], qui avait une borne de complexité $\log(\ell) d^3 k^2$. Une implémentation a été réalisée dans GIVARO.

Les tables 13 et 14 donnent les temps de calcul de cette approche (*High-order*) ainsi qu'une approche numérique directe (*récursion partielle*, voir [T65] pour plus

Facteurs de Transcription	n	degrés	$Eigen$	$High-order$
CGCACCC	21	18/19	**0.32s**	3.65s
TCCGTGGA	22	19/20	**0.33s**	3.63s
AACAACAAC	23	21/22	**0.40s**	5.73s
(A|C)TAAA(C|T)AA	25	20/20	**0.44s**	4.89s
(A|T){3}TTTGCTC(A|G)	30	23/23	**0.50s**	6.13s
A{24}	38	36/37	**0.56s**	19.61s
TA(A|T){4}TAG(A|C)	54	21/22	**0.66s**	7.23s
(C|T)CCN(C|T)TN(A|G){2}CCGN	66	24/25	**1.06s**	9.59s
GCGCN{6}GCGC	228	54/55	**4.08s**	67.09s
CGGN{8}CGG	419	81/82	**10.71s**	284.93s
TTGACAN{17}TATAAT	2068	173/173	**37.97s**	MT
TTGACAN{16, 18}ATATAAT	2904	253/253	**54.04s**	MT
GCGCN15GCGC	6158	1079/1080	**215.10s**	MT

Table 13: Recherche de facteurs de transcription dans le chromosome humain 5 (temps en secondes, MT signifie dépassement mémoire).

de détails), sur deux applications en biologie : des facteurs de transcription (pour la régulation de l'expression de gènes) sur le chromosome humain 5 et des signatures PROSITE (motifs fonctionnels) du protéome humain.

Dans la table 13, les facteurs de transcription s'expriment sur l'alphabet ADN $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ utilisant la notation IUPAC, $\mathtt{N} = (\mathtt{A}|\mathtt{C}|\mathtt{G}|\mathtt{T})$. n est la dimension de la matrice de transition de l'automate associé au facteur de transcription ; "degrés" correspond aux degrés en z de la fraction rationnelle reconstruite, "Eigen" est le temps de calcul du développement à l'ordre 131 624 728 (nombre de bases dans la séquence ADN) de l'approche numérique, "High-order" est le temps de calcul de la reconstruction et du développement par remontée.

signature PROSITE	n	degrés	$Eigen$	$High-order$
PILI_CHAPERONE	46	15/18	1.16s	**0.39s**
EFACTOR_GTP	52	14/16	0.84s	**0.42s**
ALDEHYDE_DEHYDR_CYS	67	11/12	2.04s	**0.44s**
SIGMA54_INTERACT_2	85	0/16	2.53s	**0.40s**
ADH_ZINC	87	37/41	**3.31s**	4.50s
SUGAR_TRANSPORT_1	188	17/18	4.13s	**1.10s**
THIOLASE_1	254	37/38	6.44s	**2.18s**
FGGY_KINASES_2	463	26/30	13.60s	**4.12s**
PTS_EIIA_TYPE_2_HIS	756	77/80	**20.88s**	31.33s
SUGAR_TRANSPORT_2	1152	149/151	**43.73s**	113.28s

Table 14: Recherche de motifs fonctionnels dans le protéome humain

Dans la table 14, les motifs fonctionnels (Signatures PROSITE) s'expriment sur l'alphabet des acides-aminés. "Eigen" est le temps de calcul du développement de Taylor à l'ordre 9 884 385 (nombre d'acides aminés dans le protéome) de l'approche

numérique, "High-order" est le temps de calcul de la reconstruction et du développement par remontée.

- En pratique, l'algorithme de reconstruction/remontée est très rapide quand la fraction rationnelle est de petit degré d par rapport à la taille des matrices.
- Toutefois, le facteur limitant est le pré-calcul de la série bivariée rationnelle à reconstruire, tant en temps qu'en mémoire. Une reconstruction symbolique-numérique à partir de coefficients approchés de la série rationnelle à l'aide des méthodes de [152 - Kaltofen et Yang (2007)] pourrait induire des améliorations significatives.

5 Conception et modélisation logicielles

Cette section aborde deux aspects de la production logicielle. Un aspect plus technique proposant des modèles de conception génériques issus principalement des expériences GIVARO, LINBOX ou KAAPI; puis un aspect plus théorique allant vers une formalisation des techniques utilisées, en particulier des paradigmes orientés objet.

Au niveau des modèles de conception, l'idée est de suivre la démarche des *design pattern* [117 - Gamma et al. (1994)] et plus généralement des *algorithmic patterns* [92 - Buschmann et al. (1996), 125 - Ginat (2004)] pour des applications en calcul formel. Nous avons proposé un certain nombre de concepts novateurs, comme les archétypes pour le contrôle de l'explosion de code, des itérateurs parallèles de boîtes noires, des restes chinois génériques, etc. qui ont des applications plus larges que leurs bibliothèques d'origine. Nous proposons donc une classification, une structuration d'outils génériques pour le calcul formel, appelés *Computer Algebra Patterns*.

Au niveau formalisation, nous avons proposé un nouveau langage de modélisation diagrammatique, utilisant la théorie des catégories et plus particulièrement les sommes amalgamées, pour décrire de nombreuses constructions orientées-objets. Plusieurs paradigmes impératifs plus classiques comme le mécanisme des exceptions, l'état implicite de la mémoire, les effets de bord en général, nous ont également semblé nécessiter des approfondissements. Nous avons ainsi proposé une formalisation par de nouvelles catégories cartésiennes à effets de ces mécanismes, afin de donner des bases cohérentes au travail sur l'orienté-objet. Ainsi, les formalisations de l'état, des exceptions ou encore des constructeurs/destructeurs semblent être de nature similaire. Notre approche est théorique mais nous gardons la volonté de décrire des implémentations réelles et la définition de DML (un nouveau langage de modélisation diagrammatique, voir section 5.5) permet par exemple de traiter la plupart des structures complexes de LINBOX.

5.1 Computer algebra patterns

Nous proposons dans cette section des éléments réutilisables de programmation orientée-objet pour le calcul formel. En effet, de nombreuses bibliothèques réutilisent les mêmes structures d'algorithmes, parfois les mêmes algorithmes avec à chaque fois des solutions différentes. L'idée est de proposer des solutions standard pour répondre à des problèmes d'architecture, de conception et d'algorithmique du calcul formel. Il s'agit donc de fournir des patrons de conception (*design patterns*) augmentés de parties algorithmiques. Dans ce cadre, le projet //INBOX a pour objectif de rassembler des chercheurs des communautés du calcul formel et du calcul parallèle autour de nouvelles problématiques liées au développement de bibliothèques de calcul effectives sur des architectures parallèles, allant des grilles aux architectures de calcul ambiant [P40, B32]. Il s'agit d'identifier des concepts de programmation clefs sur lesquels fonder cette interaction au niveau du développement logiciel, ainsi que les verrous théoriques et pratiques. Ces recherches et les solutions envisagées orienteront alors aussi bien l'évolution des intergiciels de programmation parallèle que des bibliothèques de calcul exact.

Il semble que le niveau intéressant ici soit le niveau intergiciel adaptatif et l'on peut alors par exemple :

- Coupler des bibliothèques optimisées avec des objets mathématiques de plus haut niveau. Les interfaces LINBOX-Gap[4] [S59, C39] (réalisée durant mon doc-

[4] http://www.cis.udel.edu/~dumas/Homology, http://www.gap-system.org/~gap

torat), LINBOX-Maple de Pascal Giorgi, ou LINBOX-SAGE de Clément Pernet, sont un point de départ.

- S'adapter efficacement aux données et aux ressources. Le cadre générique proposé en section 5.2 permet de définir des choix algorithmiques et de les ordonnancer indépendamment de leur finalité.

Les patrons de conception sont principalement de trois types : créationnels (ils définissent comment faire l'instanciation et la configuration des classes et des objets), structuraux (ils définissent comment organiser les classes d'un programme dans une structure plus large, séparant l'interface de l'implémentation), comportementaux (ils définissent comment organiser les objets pour que ceux-ci collaborent, comment distribuer les responsabilités, et expliquent le fonctionnement des algorithmes impliqués). Nous donnons quelques-unes des évolutions, appelées *Computer Algebra Patterns*, issues de la bibliothèque LINBOX :

- Création
  - `Stream` (matrice, premier, itérateur aléatoire) est une évolution du patron *Fabrique*.
- Structure
  - `Composer` est une évolution de *Composite*.
  - `BlackBox` est une évolution de *Pont*.
  - `Envelope` est une évolution d'*Adapteur*.
  - `Archetype` est une évolution de *Proxy*.
  - `Mapper` (homomorphisme).
  - `Rebinder` (présent dans la STL[171 - Musser et Saini (1996)]).
- Comportement
  - `CRT` pour les restes chinois est une évolution de *Méthode socle*.
  - `ParallelIterator` est bien sûr une évolution d'*Iterator*.
  - `Domain` (pour les coefficients) est une évolution de *Visiteur*.
  - `Adaptive` est une évolution de *Stratégie*.
  - `Interface` (e.g. Maple, GAP, SAGE) est une évolution de *Fabrique/Visiteur*.

`Envelope`, `Archetype` sont explicités en section 5.5. Les patrons de structure sont la plupart du temps des évolutions des patrons de conception classiques [117 - Gamma et al. (1994)]. L'`Archetype`, par exemple, ajoute une solution algorithmique pour réduire l'explosion de code généré [B28]. `Adaptive` est explicitée section 5.2 et nous présentons `CRT` en section 5.1.2. En exemple, nous donnons l'extraction d'un patron de conception de composition à partir de la classe `Compose` de la bibliothèque LINBOX.

5.1.1 Le *CA Pattern* de composition

La méthode de composition de matrices boîtes noires de LINBOX s'écrit comme dans la classe 29 et on voit se dégager un patron général de composition de fonctions donné dans la classe 30.

Code 29 `linbox/blackbox/compose.h`

```
template <class BB1, class BB2> class Compose {
   const BB1& _A; const BB2& _B; vector _z;
public:
   Compose(const BB1& A, const BB2& B) : _A(A), _B(B) {
      linbox_check( A.coldim()==B.rowdim() );
      _z.resize( A.coldim() );
   }

   template <class OutVector, class InVector>
   OutVector& Apply(OutVector& y, const InVector& x) const {
      return _A.Apply( y, _B.Apply( z, x ) );
   }

   size_t rowdim() const         { return _A.rowdim(); }
   size_t coldim() const         { return _B.coldim(); }
};
```

Code 30 CA pattern : Composer

```
template <class F1, class F2> class Composer {
   const F1& _A; const F2& _B;
   // Type z; // à remplir, ou utiliser TypeChooser
public:
   Composer(const F1& A, const F2& B) : _A(A), _B(B) {
        // à remplir
   }

   template <class Out, class In>
   Out& operator()(Out& y, const In& x) const {
      return _A( y, _B( z, x ) );
   }
};
```

Cela donne une composition générique prédéfinie à laquelle peuvent s'ajouter des outils automatiques, souvent des Traits et de la meta-programmation : par exemple `Mapper`/`Rebinder` facilite la déclaration et l'application d'homomorphismes, des traits, ou `TypeChooser` [149 - Kaltofen et al. (2005)], automatisent la gestion, les choix automatiques ou non du type intermédiaire, etc.

5.1.2 Design générique de reconstruction par restes chinois

Un autre exemple de patron de conception est l'algorithme des restes chinois. Ici le patron inclut également le code effectif sur les entiers, et propose une implémentation générique, indépendante de l'algorithme qui utilise les projections, et indépendante du parallélisme éventuel sous-jacent. Nous avons proposé un tel patron dans [B19] avec une structure de données dédiée, et des outils pour gérer la terminaison anticipée, y compris dans un contexte parallèle. Ces travaux ont été pour partie effectués dans le cadre du projet ANR Safescale [P44].

La structure de données que nous proposons est un tableau dans lequel les éléments d'indice i ont une taille 2^i. Les résidus des restes chinois à reconstruire, de taille 1 entrent dans le tableau et sont ensuite combinés deux à deux en remontant le tableau jusqu'à arriver dans une case vide. Nous avons appelé cette structure une échelle de résidus (*radix ladder*) et elle permet de stocker efficacement l'arbre des résidus indépendamment de l'algorithme de reconstruction choisi (Lagrange, Newton, diviser pour régner etc.).

Ensuite, une autre possibilité de généricité réside dans la stratégie choisie pour la reconstruction : il peut s'agir d'une reconstruction déterministe, où des bornes fines sur les coefficients permettent de garantir que l'on a suffisamment de résidus, ou des stratégies de terminaison anticipée, où l'on arrête la reconstruction avant terme si les entiers produits se stabilisent [112 - Emiris (1998), A17, 147 - Kaltofen (2002)]. Nous appelons ce module, décidant la stratégie et utilisant par exemple les échelles de résidus pour sa reconstruction, un constructeur (*builder*).

Enfin, nous proposons une structure de contrôle, cette fois-ci générique par rapport au constructeur *et* par rapport à l'algorithme dont on veut remonter les entiers, nous l'appelons *CRA-control*. L'interface d'un contrôleur est réduite à un constructeur à partir d'un *builder* et une fonction classe itérant une boîte noire (comme le déterminant modulo, le polynôme minimal modulo, etc.). Le CA-pattern associé définissant l'interface de *builder* est donné dans l'algorithme 31 et la figure 11 présente l'architecture générale du patron de reconstruction par restes chinois.

Algorithme 31 CRA-CONTROL

1 : $Builder$.**initialize**();
2 : **Tant que** $Builder$.**notTerminated**() **Faire**
3 : $p := Builder$.**nextCoPrime**();
4 : $v := BlackBox$.**apply**(p);
5 : $Builder$.**update**(v, p);
6 : **Fin Tant que**
7 : Return $Builder$.**reconstruct**();

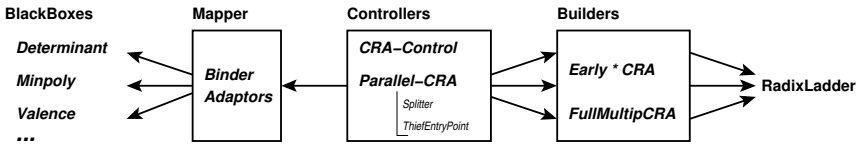


Figure 11: Schéma générique de reconstruction par restes chinois

- Dans [B19] nous présentons également un prototype de terminaison anticipée parallèle utilisant KAAPI [122 - Gautier et al. (2007)]. La difficulté est ici de minimiser le nombre d'appels à la boîte noire, tout en réalisant un regroupement non-bloquant des résidus afin de tester la terminaison. Dans le cas d'une stratégie de terminaison anticipée, et quand le coût de la boîte noire n'est pas prédominant, des techniques d'amortissement deviennent indispensables [83 - Beaumont et al. (2004)]. La table 15 montre des exemples d'accélérations obtenues sur 16 cœurs en remplaçant uniquement le contrôleur donné dans l'algorithme 31 par des contrôleurs parallèles (synchrone avec openMP, puis asynchrone avec Kaapi).

Matrice	$ex-1$	$ex-3$	$t-150$	$t-300$	$t-500$	$t-700$	$t-2000$
Taille	560	2600	150	300	500	700	2000
Ω	8736	71760	2040	4678	8478	12654	41907
OMP	**1.38**	10.66	2.10	8.52	11.29	12.55	12.48
Kaapi	1.35	**10.74**	**5.78**	**10.52**	**11.56**	12.55	**12.59**

Table 15: Accélérations pour le calcul du déterminant sur $\mathbb{Z}$ obtenues sur 8 double-cœurs (Opteron 875, 1MB L2 cache, 2.2Ghz).

- Dans [B32], nous généralisons cette construction par terminaison anticipée à un nouvel algorithme, reprenant les constructions d'algorithmes de la STL[§]. Nous proposons `accumulate_until` qui abstrait la notion de terminaison anticipée pour fournir un algorithme générique donc la sémantique est donnée dans le code 32.

  L'algorithme `accumulate_until` prend un tableau v de taille N (entre les itérateurs `beg` et `end`), un opérateur unaire $func$ devant être appliqué sur les éléments de v et une fonction binaire spécifique `Accumulator`. Cet accumulateur, dont un exemple est donné à partir d'un prédicat, se comporte comme une addition en place (`a+=b`) et retourne `true` pour indiquer que suffisamment de valeurs ont été accumulées. Soit $S_k = \sum_{i=0,..,k} func(v[i])$ avec $k \in \{0, N\}$. L'algorithme calcule et renvoie $n \leq N$ et S_n tels que, soit une accumulation durant le calcul de S_n a renvoyé `true`, soit $n = N$. L'usage attendu est que toute accumulation supplémentaire renverrait également `true`.

- Enfin, une évolution de ce patron vers la reconstruction rationnelle a été amorcée dans la thèse de A. Urbańska-Marzalek.

[§] www.sgi.com/tech/stl

Code 32 ACCUMULATE_UNTIL

```
template<class T, class Predicate> struct DefaultAccumulator {
    bool operator()(T& a, const T& x) {
        a += x; return _pred(a);
    }

    DefaultAccumulator(const Predicate& P) : _pred(P) {}
    Predicate _pred;
};


template <class T, class Iterator, class Function,
          class Accumulator>
size_t accumulate_until(T& res, Iterator beg, Iterator end,
                        Function func, Accumulator accu) {
    size_t cnt = 0;
    for( ; beg != end; ++beg,++cnt) {
        typename Function::result_type r; func(r, *beg);
        if (accu(res, r)) break;
    }
    return cnt;
}
```

5.2 Cadre générique pour l'adaptativité

Les algorithmes adaptatifs présentés par exemple en sections 3.2, 3.4, 3.7 ou encore 3.8.2, reposent sur des couplages de différents algorithmes, souvent récursifs, et s'adaptant automatiquement aux ressources et aux données. Nous avons proposé en 2005 le projet AHA [P47] sur le développement d'un cadre générique pour le calcul adaptatif. Il s'agit de construire des algorithmes qui s'adaptent automatiquement au contexte d'exécution (à la fois au niveau des données et au niveau de l'architecture matérielle). Ces travaux sont appliqués au calcul fiable, à l'ordonnancement et affectation par optimisation combinatoire et aux problèmes inverses de vision artificielle (multi-caméras et temps-réel). En particulier, dans [B23], nous proposons un processus fournissant un schéma générique de couplage adaptatif reposant sur des choix récursifs à l'exécution.

5.2.1 Représentation récursive

Soit f un problème d'ensemble d'entrées I et d'ensemble de sorties O. Pour le calcul de f, un algorithme adaptatif va composer différents algorithmes $(f_i)_{i=1,\ldots,k}$, chacun pouvant résoudre le problème f. Les algorithmes séquentiels ou parallèles des sections précédentes sont tous des instances f_i distinctes.

Ces algorithmes sont de deux sortes : soit ils incluent au moins un appel récursif soit ils n'en incluent pas et sont alors appelés terminaux. Un appel récursif signifie dans ce cadre que l'algorithme f_i résout une instance de f en la réduisant à des sous-calculs d'au moins une instance de f de plus petite taille. L'adaptativité consiste alors à choisir pour chacun de ces sous-calculs l'algorithme f_j le mieux adapté. Ce que nous proposons est donc que l'appel récursif ne se fasse pas avec le même programme f_i mais avec une routine générale d'adaptativité, f_g dans le programme 33, qui choisit à

chaque appel le nouvel algorithme f_j suivant les paramètres actuels de données et de ressources.

Code 33 Description récursive d'un algorithme *adaptatif* f_i

```
1 Algorithme f_i ( int n, I input, O output, ... ) {
2     ...
3         f_g ( n - 1, ...);
4     ...
5         f_g ( n / 2, ...);
6     ...
7 };
```

Ce choix peut être implémenté de différentes façons. Par exemple, f_g peut être une méthode virtuelle pure, chaque f_i étant des spécialisations héritées.

- Le bénéfice principal de cette approche est que le travail déjà effectué est automatiquement réutilisé par la nouvelle variante prenant le relais.
- En outre, le surcoût de la stratégie d'adaptation est limité aux seuls tests de f_g exécutés à chaque appel récursif.
- Enfin ce schéma récursif peut également induire des améliorations de complexité sur l'algorithme global, comme cela est le cas par exemple pour le déterminant sur les entiers ou pour le polynôme caractéristique de matrices creuses.

5.2.2 Vol de travail et borne sur le surcoût d'adaptativité

Un cas crucial dans le couplage d'algorithmes parallèles et séquentiels est celui où l'on peut obtenir un nombre non borné de choix possibles. Nous présentons une solution utilisant du vol de travail [88 - Blumofe et Leiserson (1994)] pour coupler un algorithme séquentiel f_{seq} et un algorithme parallèle f_{par} résolvant le même problème f. Suivant [96 - Daoudi et al. (2005), 182 - Roch et al. (2006)], nous supposons que l'algorithme séquentiel exécute une première partie du calcul séquentiel (appelée *ExtractSeq*) et ensuite exécute un appel récursif terminal à f_g pour terminer le calcul. L'opération qui consiste à extraire une partie du calcul séquentiel en cours et à l'exécuter en parallèle est appelée *ExtractPar*. Pour réduire le surcoût de choix dans f_g entre f_{seq} et f_{par}, f_{seq} est le choix par défaut et f_{par} est choisi uniquement si des processeurs deviennent inactifs. Le surcoût est donc restreint aux seuls appels à *ExtractPar*. Avec le modèle de Cilk-5 [116 - Frigo et al. (1998)] et KAAPI [142 - Jafar et al. (2005)] bornant le nombre de requêtes de vol dans un ordonnancement de type vol de travail, on obtient la borne indiquée dans le théorème 34.

Théorème 34

Soient $T_1^{(\text{seq})}$ *(resp.* $T_1^{(\text{par})}$*) le temps d'exécution sur un processeur séquentiel (i.e. le travail) de* f_{seq} *(resp.* f_{par}*), et soit* $T_\infty^{(\text{par})}$ *le temps d'exécution de* f_{par} *sur un nombre non borné de processeurs identiques. Si le programme* adaptatif *est exécuté sur une machine comportant* m *processeurs identiques, alors le nombre moyen de préférences à* f_{par} *par rapport à* f_{seq} *dans* f_g *est borné par* $(m-1).T_\infty^{(\text{par})}$

- En conséquence, pour un parallélisme à grain fin vérifiant $T_\infty^{(\text{par})} \ll T_1^{(\text{seq})}$ et même quand l'adaptativité est a priori non bornée, le surcoût dû aux choix adaptatifs est négligeable par rapport au travail global à effectuer.
- Ce paradigme a été appliqué par exemple pour la résolution de systèmes à dégénération parallèle, section 3.4.3.

5.3 Modélisation catégorique des effets de bord

La production de logiciel de qualité est un autre défi important. Plusieurs formalismes contribuent à répondre à ce défi, aussi bien au niveau du langage de programmation (syntaxe, sémantiques, etc.), au niveau des langages de spécification (logiques, automates, etc.) que des techniques de preuves (démonstration (semi-)automatique, model-cheking, typages, techniques de tests, etc.). Souvent, les formalismes suscités sont définis à l'aide de la théorie des ensembles. Cependant, les solutions proposées sont parfois relativement complexes. La théorie des catégories, introduite par les mathématiciens [160 - Mac Lane (1997)], offre des moyens formels assez puissants pour fournir des définitions et des résultats sous une forme concise et simple. Plusieurs utilisations de la théorie des catégories en informatique fondamentale existent déjà. Parmi d'autres applications, on peut citer par exemple la sémantique des types abstraits, la définition des sémantiques opérationnelles utilisant la transformation (réécriture), la correspondance entre le lambda-calcul simplement typé et les catégories cartésiennes fermées, qui prolonge la correspondance de Curry et Howard [81 - Barr et Wells (1995), 158 - Lambek et Scott (1986), 76 - Asperti et Longo (1991)].

Une sémantique catégorique pour un langage de programmation associe en général un objet à chaque type, un morphisme pour chaque terme et utilise la composition et les produits pour traiter la substitution de termes.

Programme		Catégorie	
Type	X	Objet	X
Fonction	$Y\ f(X\ x);$	Morphisme	$f : X \to Y$
Arguments	$Y\ f(X\ x,\ Y\ y);$	Produit	$f : X \times Y \to Z$
Substitution	$f(g(x))$	Composition	$f \circ g$

Table 16: Sémantique catégorique pour un langage de programmation

Cela suffit pour traiter le simple cadre équationnel mais nécessite des adaptations dès que des effets de bord sont présents. Les effets considérés sont par exemple la mise

à jour de l'état implicite dans un langage impératif, la levée d'exception, les boucles infinies, etc.

En général, il y a donc deux types de termes : les termes génériques qui peuvent produire des effets et les termes *purs*, sans effet. Comme dans [165 - Moggi (1991)], un terme général peut être vu comme un *programme* qui retourne une *valeur* qui est pure.

Dans [A1] nous définissons des effets et des catégories à effet, puis nous nous intéressons au problème de la séquentialité : les produits catégoriques n'ont pas d'ordre d'évaluation des arguments, alors que cet ordre peut modifier la sémantique d'un programme en présence d'effets. Nous introduisons donc des *catégories cartésiennes à effet* comportant un *produit séquentiel* généralisant le produit cartésien, pour résoudre ce problème.

D'autres approches ont été proposées : les monades fortes [164 - Moggi (1989)], les catégories de Freyd [181 - Power et Robinson (1997)], et les Arrows [138 - Hughes (2000)]. Ces trois méthodes sont relativement similaires [134 - Heunen et Jacobs (2006), 78 - Atkey (2008)] alors que la nôtre est plus précise :

Théorème 35

Les catégories cartésiennes à effets sont des catégories de Freyd.

Théorème 36

Les catégories cartésiennes à effets induisent des Arrows.

Théorème 37

Une catégorie cartésienne avec une monade forte et les produits de Kleisli est une catégorie cartésienne à effets si et seulement si la force de la monade est consistante avec l'identité.

- Ensuite, il serait intéressant de définir une forme de clôture pour essayer de généraliser les travaux de [95 - Curien et Obtulowitz (1989)] sur la partialité à d'autres effets ; il serait également intéressant de comparer notre approche avec la logique d'évaluation de [166 - Moggi (1995)] ; les effets dans les continuations devraient également entrer dans notre cadre ; le problème de la combinaison d'effets [140 - Hyland et al. (2006)] pourrait aussi bénéficier de l'approche des catégories à effet.
- Nous sommes en train d'étendre cette approche catégorique à un langage de programmation impérative et allons vers la modélisation des objets.

5.4 Sémantique avec état implicite pour l'impératif

Les catégories à effets permettent de manipuler plusieurs niveaux d'abstraction car leur sémantique devient *décorée* [T62, §3.1] : les morphismes peuvent être purs ou

génériques et plus généralement paramétrés par une décoration qui permet de différencier leurs actions [102 - Dominguez et Duval (2010)]. Cette sémantique permet par exemple de masquer les effets, afin d'avoir une vision plus abstraite, tout en conservant des effets implicites. Nous utilisons en particulier la catégorie à effets de l'état pour décrire le fonctionnement d'un langage de programmation impérative simple: IMP [209 - Winskel (1993)]. Nous étudions les liens entre la syntaxe, la mémoire (l'état implicite) et l'exécution. Par exemple la sémantique du point-virgule ($C_1; C_2$, d'abord la commande C_1, puis la commande C_2) est donnée directement par un produit séquentiel. L'étude des branchements (`if-then-else`) des boucles et points fixes (`while`) est en cours ; à terme des équivalences entre sémantiques décorée, opérationnelle, dénotationnelle ou axiomatique seront établies [T61].

5.5 Vers une modélisation orientée objet par les diagrammes

Au sein du projet INCA [P49] nous avons proposé dans [B24] un nouveau langage de modélisation diagrammatique, DML. Le paradigme utilisé est celui de la théorie des catégories et en particulier des sommes amalgamées (ou "pushouts"). La plupart des structures orientées objet peuvent être décrites avec ce langage, de l'héritage et du polymorphisme à la généricité (par "templates"). Par ailleurs, la bibliothèque C++ d'algèbre linéaire LINBOX a été conçue pour allier efficacité et généricité. Elle requiert donc de nombreuses structures génériques et polymorphes. Grâce à DML, nous avons proposé une description simple des structures complexes de cette bibliothèque.

L'idée générale est ici de donner une sémantique à au moins une partie des constructions objet, grâce aux catégories.

Dans le but de modéliser la structure d'un logiciel C++, nous décrivons une catégorie $\mathcal{C}_{dml}$ de façon informelle. La catégorie $\mathcal{C}_{dml}$ fournit une vision synthétique d'une architecture C++, en ne faisant pas de distinction entre classe et instance, par exemple. C'est cette vision qui va permettre d'obtenir une description simple des mécanismes complexes mis en jeu dans le logiciel LINBOX.

Les points de la catégorie $\mathcal{C}_{dml}$ sont appelés les *spécifications*. Parmi les spécifications figurent tous les *types* de C++, aussi bien les types prédéfinis que les classes, et aussi les "typenames". Une spécification peut aussi être une valeur dans un type prédéfini ou une instance d'une classe. Cela peut aussi être une variable représentant un type, une valeur ou une instance. Une spécification A détermine un ensemble de *modèles* $\mathcal{M}od(A)$. Typiquement, si la spécification est une classe A, ses modèles sont les instances de la classe A, et si la spécification est une instance a, son unique modèle est lui-même. Ainsi, on peut voir une spécification soit du point de vue *syntaxique*, comme un morceau de code C++, soit du point de vue *sémantique*, comme un ensemble de modèles.

Les flèches de la catégorie $\mathcal{C}_{dml}$ sont appelés les *morphismes de spécifications*. Ces morphismes regroupent des relations de nature variée entre spécifications. Un morphisme $\varphi : A \to B$ permet de considérer le code de A comme une partie du code de B. Par exemple, un morphisme de spécifications peut être un *héritage*, entre deux classes. Quand B hérite de A, la classe B contient tous les membres (attributs et méthodes) de la classe A, plus éventuellement quelques nouveaux, il s'agit donc d'un morphisme $\varphi : A \to B$. Une *paramétrisation générique* (ou "*template*") est aussi un morphisme de spécifications. Quand une classe générique T est un paramètre générique pour une classe B, les membres de T peuvent être utilisés dans B, donc il y a un morphisme $T \to B$. Une *instanciation* est une autre espèce de morphisme de spécifications. Quand un objet a est créé comme instance d'une classe A, les membres de A sont instanciés dans a, ce qui peut être vu comme un morphisme $A \to a$. Une

implémentation d'une classe abstraite A par une classe B est aussi un morphisme $A \to B$.

Les spécifications peuvent être construites progressivement, par des constructions systématiques, grâce aux pushouts. Nous avons montré par exemple que les pushouts de la catégorie $\mathcal{C}_{dml}$ correspondent à des constructions fondamentales en C++ : l'héritage virtuel, le passage de paramètres, le passage de paramètres template, l'instanciation d'objets et le polymorphisme.

Nous proposons l'illustration suivante : les spécifications utilisées par LINBOX pour calculer sur un corps (ici le corps à deux éléments), à l'exception de la classe `Field::Element` ; elles sont représentées sur le diagramme de la figure 12.

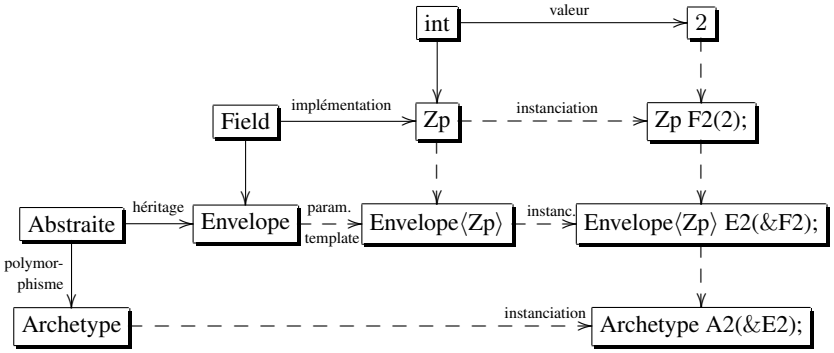


Figure 12: Un diagramme DML pour l'architecture des corps dans LINBOX, avec une enveloppe à recopie

Ce diagramme peut être lu de la gauche vers la droite, c'est-à-dire de l'abstraction vers les instances. À l'exception de la première ligne, les spécifications (les boîtes) de la colonne de droite sont des objets, et les autres sont des classes (`Zp` dénote une classe de corps finis premiers), et les flèches horizontales les plus à droite sont des instanciations. La première ligne est légèrement différente des autres : elle comporte un type prédéfini `int` à la place d'une classe, et une valeur `2` de type `int` au lieu d'une instance. Les trois objets `F2`, `E2` et `A2` représentent tous les corps à deux éléments, de trois points de vue différents. Les trois pushouts à droite, de sommets `F2`, `E2` et `A2`, construisent des instanciations. Le pushout du milieu, de sommet `Envelope⟨Zp⟩`, correspond à un passage de paramètre template. La flèche horizontale de `Abstract` vers `Envelope⟨Zp⟩ E2(&F2);` est composée de trois morphismes de nature différente : d'abord un héritage, ensuite un passage de paramètre template, et finalement une instanciation.

- La construction des archétypes et des enveloppes en LINBOX découle de particularités de C++ et de la volonté de conserver une efficacité maximale. Néanmoins, cet effort de conciliation de généricité et d'efficacité semble mener pour tout langage à un ensemble d'objets ou de modules interagissant de manière complexe. Un autre exemple est le logiciel de topologie algébrique de F. Sergeraert, écrit en Common Lisp [186 - Rubio et al. (1998), 157 - Lambán et al. (2003)].

- Il est probable que cette modélisation fonctionne bien sur une bibliothèque de calcul formel, ou plus généralement, sur toute bibliothèque reposant sur des bases mathématiques stables (l'algèbre linéaire) et une utilisation limitée des

multiples possibilités sémantiques de C++. Cependant, d'autres mécanismes de la programmation orientée objet peuvent être appréhendés par des méthodes diagrammatiques, en particulier par des pushouts, et ainsi des extensions de DML sont envisageables. Ces extensions pourraient concerner, par exemple, l'étude de la surcharge des fonctions, en tenant compte du fait que la redéfinition des types d'arguments mélange les liaisons statique et dynamique. On pourrait également les appliquer à l'étude du mécanisme des exceptions [107 - Duval et Reynaud (2005), T62], voire une analyse de l'exécution d'un programme [106 - Duval et Reynaud (1994)].

6 Perspectives

Au-delà des problématiques évoquées tout au long de cette synthèse et des questions ouvertes en algorithmique exacte ainsi qu'en modélisation et conception logicielle, je souhaite étendre mes recherches autour du développement de logiciels, pour les codes et la cryptologie, adaptés aux nouvelles architectures multi-cœurs. Deux types d'applications à court terme sont visées :

- Développer les codes spécifiques à l'algèbre linéaire exacte en petite caractéristique pour les codes correcteurs.
- Développer une arithmétique efficace, à précision fixée, spécialisée pour les architectures multi-cœurs et la cryptologie sur jacobiennes de courbes.

Le prototype de bibliothèque d'algèbre linéaire en très petite caractéristique, M4RI, est prometteur pour le premier point. Sur un exemple de code LDPC donnant lieu à une matrice 4385×4333 modulo 2, le décodage par les algorithmes spécifiques de V. Roca et M. Cunche demandait 3s sur un PC de bureau. Un prototype couplant un code LINBOX d'élimination de Gauß creuse générique avec un solveur spécialisé de M4RI, obtient déjà un temps de calcul de $0.8 + 0.05 = 0.85$ secondes. La spécialisation, par exemple, des algorithmes de factorisation pour les matrices LDPC avec les techniques M4RI devrait donc nous permettre de gagner un ordre de grandeur sur les techniques existantes de décodage, en séquentiel. Par ailleurs, des expériences de calcul sur processeurs graphiques montrent que l'algèbre linéaire numérique haute performance est dorénavant possible sur ces multi-cœurs. Un défi technologique est ici de combiner l'approche numérique/exacte de FFLAS avec la caractéristique 2, les formats spécifiques de matrices compressées pour GPU et les matrices de codes. Des progrès algorithmiques dans ce domaine sur les formes normales nous permettent également de découvrir de nouveaux codes correcteurs définis sur des sous-espaces de matrices et d'envisager, par des techniques équivalentes, des classifications de semi-corps jusqu'alors impossibles.

Pour le deuxième point, les dernières générations de calculateurs intègrent des processeurs spécialisés comme des FPGA (Field-programmable gate array) reprogrammables et des GPU (Graphics Processing Unit) afin d'atteindre des performances importantes. Malheureusement, les logiciels actuels d'arithmétique et de cryptologie ne tirent en général pas parti de ces accélérateurs matériels. Par ailleurs, une tendance actuelle en cryptologie à clef publique est de privilégier les techniques de courbes elliptiques ou plus généralement de jacobiennes de courbes ou de couplages. Des logiciels comme Crypto++, BorZoi ou openSSL sont spécialisés pour la cryptologie, mais un logiciel générique de précision arbitraire comme GMP est souvent meilleur en terme de performances (voir par exemple `http://gmplib.org/32vs64.html`). Il semble donc réaliste d'optimiser ces bibliothèques au moins pour la cryptologie des jacobiennes à précision fixée sur architecture spécifique. Plusieurs pistes sont envisagées :

- Modifier les structures de données, c'est ce que nous avons fait avec la bibliothèque PALOALTO.
- Coupler d'autres arithmétiques : par exemple, sur FPGA, il existe des bases RNS (*residue number system*) adaptées ; comment peut-on coupler efficacement les FPGA avec des GPU et des CPU ?
- Étudier d'autres courbes : de nombreux travaux théoriques sur les différentes représentations des courbes existent, il faudra identifier ou construire des représentations et des courbes les plus adaptées aux multi-cœurs.

- Continuer à travailler sur les couplages (par exemple Tate-Weyl) qui semblent prometteurs ; qu'en est-il au niveau logiciel embarqué ?
- Enfin, s'attacher à développer la résistance des systèmes embarqués aux attaques par perturbations algébriques, par exemple, est vital.

Les thèmes de recherche que je veux développer dans ce cadre sont un pont entre mon domaine principal, le calcul exact (l'algèbre linéaire, principalement, et les corps finis) et la thématique dans laquelle je veux m'engager, plus orientée vers les codes (cryptologie, codes correcteurs).

Un défi à plus long terme pour la communauté sera de fédérer les développements arithmétiques autour des corps finis, des entiers, des rationnels, des polynômes, puis leurs applications par exemple pour la cryptographie, la cryptanalyse et les codes correcteurs.

Plus généralement, les projets GIVARO et LINBOX démontrent qu'il est possible d'allier efficacité, généricité et développement pérenne pour l'algorithmique du calcul exact. Ainsi, le couplage efficace et durable d'algorithmes sur des ressources aujourd'hui majoritairement orientées multi-cœurs, tout comme le besoin de garanties et de certification, même pour des domaines où l'approximation est indispensable, sont des problèmes majeurs qui dépassent largement le cadre du calcul exact.

Publications

Publications de rang A

Revues internationales avec comité de lecture

[A1] Jean-Guillaume Dumas, Dominique Duval et Jean-Claude Reynaud. – Cartesian effect categories are Freyd-categories. *Journal of Symbolic Computation*, 2010. – to appear. {66}

[A2] Jean-Guillaume Dumas, Laurent Fousse et Bruno Salvy. – Simultaneous modular reduction and Kronecker substitution for small finite fields. *Journal of Symbolic Computation*, 2010. – to appear. {7, 8, 9, 12, 29}

[A3] Jean-Guillaume Dumas, Rod Gow, Gary McGuire et John Sheekey. – Subspaces of matrices with special rank properties. *Linear Algebra and its Applications*, volume 433, n° 1, juillet 2010, pages 191–202. {48}

[A4] Jean-Guillaume Dumas, Pascal Giorgi et Clément Pernet. – Dense linear algebra over prime fields. *ACM Transactions on Mathematical Software*, volume 35, n° 3, novembre 2008, pages 1–42. {7, 26, 34, 38, 39}

[A5] Jean-Guillaume Dumas. – Bounds on the coefficients of the characteristic and minimal polynomials. *Journal of Inequalities in Pure and Applied Mathematics*, volume 8, n° 2, avril 2007, pages 6 pp, art. 31. {43, 45, 46}

[A6] Jacques Dubrois et Jean-Guillaume Dumas. – Efficient polynomial time algorithms computing industrial-strength primitive roots. *Information Processing letters*, volume 97, n° 2, janvier 2006, pages 41–45. {16, 17}

[A7] Jean-Guillaume Dumas et Jean-Louis Roch. – On parallel block algorithms for exact triangularizations. *Parallel Computing*, volume 28, n° 11, novembre 2002, pages 1531–1548. {25, 38}

[A8] Jean-Guillaume Dumas, B. David Saunders et Gilles Villard. – On efficient sparse integer matrix Smith normal form computations. *Journal of Symbolic Computation*, volume 32, n° 1/2, juillet–août 2001, pages 71–99. {25, 47}

Revues nationales avec comité de lecture

[A9] Jean-Guillaume Dumas. – Caractérisation des quenines et leur représentation spirale. *Mathématiques et Sciences Humaines*, volume 184, n° 4, 2008, pages 9 – 23. {19}

Conférence ISSAC (ACM International Symposium on Symbolic and Algebraic Computation)

[A10] Brice Boyer, Jean-Guillaume Dumas, Clément Pernet et Wei Zhou. – Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm. Dans : *ISSAC'2009* [162 - May (2009)], pages 135–143. {29, 31}

[A11] Jean-Guillaume Dumas, Clément Pernet et B. David Saunders. – On finding multiplicities of characteristic polynomial factors of black-box matrices. Dans : *ISSAC'2009* [162 - May (2009)], pages 55–62. {44}

[A12] Jean-Guillaume Dumas. – Q-adic transform revisited. Dans : *ISSAC'2008* [144 - Jeffrey (2008)], pages 63–69. {7, 15}

[A13] Jean-Guillaume Dumas, Clément Pernet et Zhendong Wan. – Efficient computation of the characteristic polynomial. Dans : *ISSAC'2005* [154 - Kauers (2005)], pages 140–147. {43, 47}

[A14] Jean-Guillaume Dumas et Aude Rondepierre. – Algorithms for symbolic/numeric control of affine dynamical systems. Dans : *ISSAC'2005* [154 - Kauers (2005)], pages 277–284. {53}

[A15] Jean-Guillaume Dumas, Pascal Giorgi et Clément Pernet. – FFPACK: Finite field linear algebra package. Dans : *ISSAC'2004* [133 - Gutierrez (2004)], pages 119–126. {38}

[A16] Jean-Guillaume Dumas, Thierry Gautier et Clément Pernet. – Finite field linear algebra subroutines. Dans : *ISSAC'2002* [167 - Mora (2002)], pages 63–74. {26}

[A17] Jean-Guillaume Dumas, B. David Saunders et Gilles Villard. – Integer Smith form via the Valence: experience with large sparse matrices from Homology. Dans : *ISSAC'2000* [197 - Traverso (2000)], pages 95–105. {47, 61}

Publications de rang B

Autres conférences internationales avec comité de lecture

[B18] Brice Boyer, Jean-Guillaume Dumas et Pascal Giorgi. – Exact sparse matrix-vector multiplication on gpu's and multicore architectures. Dans : *PASCO 2010* [168 - Moreno-Maza et Roch (2010)], pages 80–88. {41}

[B19] Jean-Guillaume Dumas, Thierry Gautier et Jean-Louis Roch. – Generic design of chinese remaindering schemes. Dans : *PASCO 2010* [168 - Moreno-Maza et Roch (2010)], pages 26–34. {61, 62}

[B20] Alexandre Berzati, Cécile Canovas, Jean-Guillaume Dumas et Louis Goubin. – Fault attacks on RSA public keys: Left-to-right implementations are also vulnerable. Dans : *CTRSA'2009* [115 - Fischlin (2009)], pages 414–428. {21, 23}

[B21] Jean-Guillaume Dumas, Laurent Fousse et Bruno Salvy. – Compressed modular matrix multiplication. Dans : *Milestones in Computer Algebra 2008, Tobago*, édité par Mark Giesbrecht et Stephen Watt, 1–3 mai 2008, pages 1–8. {29}

[B22] Jean-Guillaume Dumas, Pascal Giorgi, Philippe Elbaz-Vincent et Anna Urbańska. – Parallel computation of the rank of large sparse matrices from algebraic k-theory. Dans : *PASCO 2007* [169 - Moreno-Maza et Watt (2007)], pages 43–52. { 40, 41}

[B23] Van-Dat Cung, Vincent Danjean, Jean-Guillaume Dumas, Thierry Gautier, Guillaume Huard, Bruno Raffin, Christophe Rapine, Jean-Louis Roch et Denis Trystram. – Adaptive and hybrid algorithms: classification and illustration on triangular system solving. Dans : *TC'2006* [105 - Dumas (2006b)], pages 131–148. { 63}

[B24] Jean-Guillaume Dumas et Dominique Duval. – Towards a diagrammatic modeling of the linbox C++ linear algebra library. Dans : *LMO'2006, Langages et Modèles à Objets, Nîmes, France*, édité par Roger Rousseau, Christelle Urtado et Sylvain Vauttier, 22–24 mars 2006. Pages 117–132. – Hermes science. { 67}

[B25] Jean-Guillaume Dumas et Anna Urbańska. – An introspective algorithm for the determinant. Dans : *TC'2006* [105 - Dumas (2006b)], pages 185–202. { 41}

[B26] Jean-Guillaume Dumas. – Efficient dot product over finite fields. Dans : *CASC'2004, Proceedings of the seventh International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine*, édité par Victor G. Ganzha, Ernst W. Mayr et Evgenii V. Vorozhtsov, 12–19 juillet 2004. Pages 139–154. – Technische Universität München, Germany. { 7, 10}

[B27] Jean-Guillaume Dumas et Aude Rondepierre. – Modeling the electrical activity of a neuron by a continuous and piecewise affine hybrid system. *Lecture Notes in Computer Science (HSCC'2003, Proceedings of the 2003 Hybrid Systems: Computation and Control, Prague, The Czech Republic)*, volume 2623, avril 2003, pages 156–171. { 54}

[B28] Jean-Guillaume Dumas, Thierry Gautier, Mark Giesbrecht, Pascal Giorgi, Bradford Hovinen, Erich Kaltofen, B. David Saunders, Will J. Turner et Gilles Villard. – LinBox: A generic library for exact linear algebra. Dans : *ICMS'2002, Proceedings of the 2002 International Congress of Mathematical Software, Beijing, China*, édité par Arjeh M. Cohen, Xiao-Shan Gao et Nobuki Takayama, 17–19 août 2002. Pages 40–50. – World Scientific Pub. { 59}

[B29] Jean-Guillaume Dumas et Gilles Villard. – Computing the rank of sparse matrices over finite fields. Dans : *CASC'2002* [118 - Ganzha et al. (2002)], pages 47–62. { 25, 41}

[B30] Jean-Guillaume Dumas et Jean-Louis Roch. – A fast parallel block algorithm for exact triangularization of rectangular matrices. Dans : *SPAA'01. Proceedings of the Thirteenth ACM Symposium on Parallel Algorithms and Architectures, Kreta, Greece*, édité par Pierre Fraigniaud, juillet 2001, pages 324–325. { 38}

[B31] Jean-Guillaume Dumas. – Calcul parallèle du polynôme minimal entier en Athapascan-1 et Linbox. Dans : *RenPar'2000. Actes des douzièmes rencontres francophones du parallélisme, Besançon, France*, édité par Jean-Marc Nicod, 19-22 juin 2000, pages 119–124. { 40}

Conférences internationales avec actes

[B32] Jean-Guillaume Dumas, Thierry Gautier, Clément Pernet et B. David Saunders. – LinBox founding scope allocation, parallel building blocks, and separate compilation. Dans : *ICMS'2010, Proceedings of the 2010 International Congress of Mathematical Software, Kobe, Japan*, édité par Komei Fukuda, Joris van der Hoeven et Michael Joswig, 13–17 septembre 2010. – LNCS. ..... { 58, 62 }

[B33] Jean-Guillaume Dumas, Clément Pernet et Jean-Louis Roch. – Adaptive triangular system solving. Dans : *Challenges in Symbolic Computation Software*, édité par Wolfram Decker, Mike Dewar, Erich Kaltofen et Stephen M. Watt, octobre 2006. Pages 1 – 18. – Dagstuhl Seminar proceedings 06271. ..... { 36 }

Monographies et chapitres de livres

[C34] Jean-Guillaume Dumas, Jean-Louis Roch, Eric Tannier et Sébastien Varrette. – *Foundations of Coding: Compression, Encryption, Error-Correction*. – 2010, 374 pages. Traduction de Romain Xu et Rodney Coleman.

[C35] Jean-Guillaume Dumas, Jean-Louis Roch, Eric Tannier et Sébastien Varrette. – *Théorie des codes : Compression, Cryptage, Correction*. – Dunod, 2007, 352 pages.

[C36] Pascal Bouvry, Jean-Guillaume Dumas, Roland Gillard, Jean-Louis Roch et Sébastien Varrette. – Cryptographie à clef secrète. Dans : *Cryptographie et sécurité des systèmes et réseaux*, édité par T. Ebrahimi, F. Leprevost et B. Warusfeld, pages 23–102. – Hermes, 2006.

[C37] Jean-Guillaume Dumas, Franck Leprevost, Jean-Louis Roch, Valentin Savin et Sébastien Varrette. – Cryptographie à clef publique. Dans : *Cryptographie et sécurité des systèmes et réseaux*, édité par T. Ebrahimi, F. Leprevost et B. Warusfeld, pages 103–186. – Hermes, 2006.

[C38] Jean-Guillaume Dumas, Franck Leprevost, Jean-Louis Roch et Sébastien Varrette. – Architectures pki. Dans : *Cryptographie et sécurité des systèmes et réseaux*, édité par T. Ebrahimi, F. Leprevost et B. Warusfeld, pages 187–210. – Hermes, 2006.

[C39] Jean-Guillaume Dumas, Frank Heckenbach, B. David Saunders et Volkmar Welker. – Computing simplicial homology based on efficient smith normal form algorithms. Dans : *Algebra, Geometry and Software Systems*, édité par Michael Joswig et Nobuki Takayama, pages 177–206. – Springer, mars 2003. { 58 }

Contrats de recherche

[P40] L. Informatique de Grenoble, L. d'Informatique, Robotique, Microélectronique de Montpellier, L. Jean Kuntzmann et L. d'Informatique et du

Parallélisme. – //INBOX : *Outils logiciels pour le calcul algébrique haute performance*. – 12 k€, CNRS-PEPS, 2010-2011. {40, 58}

[P41] L. Informatique de Grenoble, Verimag, L. Jean Kuntzmann, Institut Fourier, Communication & Systems, Netheos, iWall/Mataru, EasyiiC et CEA/Leti. – SHIVA : *Secured Hardware Immune Versatile Architecture*. – 2200 k€, Ministère de l'industrie, 2009-2011. {6, 24}

[P42] L. Jean Kuntzmann et Institut Fourier. – PALO-ALTO : *Plate-forme d'Attaques LOgicielles par ALgorithmes et Techniques Optimisés pour architectures Multi-Cœurs Parallèles*. – 57 k€, UJF-Pôle MSTIC, 2008-2009. {6, 18, 70}

[P43] Communication & Systems, L. Informatique de Grenoble, L. Jean Kuntzmann, Institut Fourier et Verimag. – EAU : *Formations à la cryptologie et à la sécurité, mise en place d'infrastructures sécurisées.* – 3000 k€, Contrat industriel, 2006-2010. {6}

[P44] L. d'Informatique de Paris Nord, L. Informatique de Grenoble, É. N. Supérieure des Télécommunications de Bretagne, Institut Fourier et L. Jean Kuntzmann. – SAFESCALE : *Certification et tolérance aux fautes sur grille de calcul*. – 120 k€, ANR, 2006-2009. {61}

[P45] L. Jean Kuntzmann. – CHPID : *Nouveaux outils mathématiques pour le calcul scientifique*. Dans : *Calcul Hautes Performances et Informatique Distribuée*. – 14 k€, Cluster ISLE de la Région Rhône-Alpes, 2005-2008. {52}

[P46] L. Jean Kuntzmann et L. d'InfoRmatique en Image et Systèmes d'information. – CALCEL : *Calcul Cellulaire*. – 120 k€, Région Rhône-Alpes, 2005-2008.

[P47] L. Jean Kuntzmann et L. Informatique de Grenoble. – AHA : *Algorithmes Hybrides Adaptatifs*. – 80 k€, IMAG, 2005-2007. {63}

[P48] L. Jean Kuntzmann. – CC : *Contrôle Hybride*. Dans : *Computation and Control*. – 4 k€, Projet Européen, 2003-2004. {53}

[P49] L. Jean Kuntzmann et L. Logiciels Systèmes Réseaux. – INCA : *Interfaces pour le calcul formel*. – 30 k€, IMAG, 2003-2004. {67}

[P50] L. Jean Kuntzmann. – SQUASH : *Analyse qualitative des systèmes hybrides*. – 10 k€, CNRS, 2002-2003. {54}

[P51] U. de Strasbourg, Tsinghua U. et U. de Grenoble. – FCSDH : *Laboratoire Franco-Chinois sur les systèmes dynamiques hybrides*. – 2002-2006. {53}

Logiciels

[S52] Brice Boyer et Jean-Guillaume Dumas. – FFSpMv: Finite field sparse matrix-vector product on multi-cores, 2010. https://ljkforge.imag.fr/projects/ffspmvgpu. {40}

[S53] Brice Boyer et Jean-Guillaume Dumas. – Galet: Matrix multiplication schedule generator, janvier 2009. http://ljk.imag.fr/CASYS/LOGICIELS/Galet. {29}

[S54] Jean-Guillaume Dumas, Laurent Fousse et Pascal Giorgi. – Recint: Recursive fixed precision integers, 2009. https://www.ljkforge.imag.fr/projects/paloalto. {6, 18, 70}

[S55] Jean-Guillaume Dumas et Clément Pernet. – Exact linear system resolution in M4RI, novembre 2008. http://m4ri.sagemath.org. {29, 48}

[S56] Jean-Guillaume Dumas, Pascal Giorgi et Clément Pernet. – FFLAS-FFPACK: Finite field linear algebra subroutine/package, février 2006. {39}

[S57] Thierry Gautier, Gilles Villard, Jean-Louis Roch, Jean-Guillaume Dumas, Pascal Giorgi et Clément Pernet. – Givaro, a C++ library for computer algebra: exact arithmetic and data structures, octobre 2005. {5, 6, 7, 18, 24, 55, 58, 71}

[S58] The LINBOX group. – LINBOX 1.0, juillet 2005. {5, 26, 39, 40, 58, 59, 60, 67, 68, 70, 71}

[S59] Jean-Guillaume Dumas, Frank Heckenbach, B. David Saunders et Volkmar Welker. – *Simplicial Homology, a (proposed) share package for GAP*, mars 2000. Manual, http://www.cis.udel.edu/~dumas/Homology. {58}

Rapports de recherche et prépublications soumises

[T60] Jean-Guillaume Dumas. – *Les rayons des permutations spirales*. – Rapport technique, IMAG-hal-00447415, 2010. http://hal.archives-ouvertes.fr/hal-00447415. {19, 21}

[T61] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse et Jean-Claude Reynaud. – *A state-free semantics for an imperative language*. – Rapport technique, IMAG-CCSD, mars 2010. {67}

[T62] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse et Jean-Claude Reynaud. – *States and exceptions are dual effects*. – Rapport technique, IMAG-hal-00445873, arXiv: cs.LO/1001.1662, janvier 2010. http://hal.archives-ouvertes.fr/hal-00445873. {66, 69}

[T63] Jean-Guillaume Dumas, Philippe Elbaz-Vincent, Laurent Fousse, Pascal Giorgi et Jérôme Plût. – *Fixed precision arithmetic for curves*. – Rapport technique, https://www.ljkforge.imag.fr/projects/paloalto, 2010. {18}

[T64] Jean-Guillaume Dumas, Rod Gow et John Sheekey. – *Subspaces of symmetric matrices with constant rank*. – Rapport technique, 2010. {48}

[T65] Jean-Guillaume Dumas et Grégory Nuel. – *Sparse approaches for the exact distribution of patterns in long multi-states sequences generated by a Markov source*. – Rapport technique, IMAG-hal-00492738, arXiV math.Pr/1006.3246, 2010. http://hal.archives-ouvertes.fr/hal-00492738. {55}

[T66] Jean-Guillaume Dumas et John Sheekey. – *Classification of semifields of order* 5^4 *and* 3^5. – Rapport technique, 2010. {48}

[T67] Jean-Guillaume Dumas et Aude Rondepierre. – *A hybrid system approach to nonlinear optimal control problems*. – Rapport technique, IMAG-hal-00004191, arXiv math.OC/0502172, 2008. http://hal.archives-ouvertes.fr/hal-00004191. {53}

[T68] Jacques Dubrois et Jean-Guillaume Dumas. – *Polynomial time algorithms for probable primitive roots*. – Rapport technique, arXiv cs.SC/0409029, http://hal.archives-ouvertes.fr/hal-00002828, septembre 2004. {16}

[T69] Jean-Guillaume Dumas. – *Algorithmes parallèles efficaces pour le calcul formel : algèbre linéaire creuse et extensions algébriques*. – Thèse de Doctorat en mathématiques appliquées, Institut National Polytechnique de Grenoble, France et University of Delaware, USA, 20 décembre 2000. http://tel.archives-ouvertes.fr/tel-00002742. {25, 38, 47}

Bibliographie

[70] John Abbott, Manuel Bronstein et Thom Mulders. – Fast deterministic computation of determinants of dense matrices. Dans : *ISSAC 99: July 29–31, 1999, Simon Fraser University, Vancouver, BC, Canada: proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, édité par Sam Dooley, 1999, pages 197–204. {41}

[71] Journaïdi Abdeljaoued et Gennadi I. Malaschonok. – Efficient algorithms for computing the characteristic polynomial in a domain. *Journal of Pure and Applied Algebra*, volume 156, 2001, pages 127–145. {47}

[72] Alfred V. Aho, John E. Hopcroft et Jeffrey D. Ullman. – *The Design and Analysis of Computer Algorithms*. – Addison-Wesley, 1974. {29, 37}

[73] Martin Albrecht et Gregory Bard. – *The M4RI Library*. – The M4RI Team, 2008. http://m4ri.sagemath.org. {29}

[74] Vladimir L. Arlazarov, Yefim Dinitz, A. Kronrod, M et I. A. Faradjev. – On economical construction of the transitive closure of a directed graph. *Soviet Mathematics Doklady*, volume 11, n° 5, 1970, pages 1209–1210. {29}

[75] Jörg Arndt. – *Matters Computational*. – 2010. http://www.jjj.de/fxt/#fxtbook, to appear. {19}

[76] Andrea Asperti et Giuseppe Longo. – *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist*. – Cambridge, MA, MIT Press, 1991, *Foundations of Computing*. {65}

[77] Peter R. J. Asveld. – *Permuting Operations on Strings: Their Permutations and Their Primes*. – Rapport technique, TR-CTIT-09-26, Centre for Telematics and Information Technology, University of Twente, Enschede, 2009. {19}

[78] Robert Atkey. – What is a categorical model of arrows? Dans : *Mathematically Structured Functional Programming (MSFP'08)*, 2008. {66}

[79] Eric Bach. – How to generate factored random numbers. *SIAM Journal on Computing*, volume 17, n° 2, avril 1988, pages 179–193. – Special issue on cryptography. {17}

[80] Eric Bach. – Comments on search procedures for primitive roots. *Mathematics of Computation*, volume 66, n° 220, octobre 1997, pages 1719–1727. {16}

[81] Michael Barr et Charles Wells. – *Category Theory for Computing Science*. – Prentice Hall, 1995, seconde édition, *Prentice Hall International Series in Computer Science*. {65}

[82] LeRoy B. Beasley. – Spaces of rank-2 matrices over gf(2). *ELECTRONIC Journal of LINEAR ALGEBRA*, volume 5, janvier 1999, pages 11–18. {48, 51}

[83] Olivier Beaumont, El Mostafa Daoudi, Nicolas Maillard, Pierre Manneback et Jean-Louis Roch. – Tradeoff to minimize extra-computations and stopping criterion tests for parallel iterative schemes. Dans : *3rd International Workshop on Parallel Matrix Algorithms and Applications (PMAA04)*, 18–22 octobre 2004. – CIRM, Marseille, France. {62}

[84] Stuart J. Berkowitz. – On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.*, volume 18, n° 3, 1984, pages 147–150. {47}

[85] Alexandre Berzati, Cécile Canovas et Louis Goubin. – Perturbating RSA public keys: An improved attack. Dans : *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C, USA, August 10-13, 2008.*, édité par Elisabeth Oswald et Pankaj Rohatgi, 2008. – *Lecture Notes in Computer Science*, volume 5154, pages 380–395. – Springer. {22}

[86] Dario Bini et Victor Pan. – *Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms.* – Boston, Birkhauser, 1994. {38}

[87] Manuel Blum et Silvio Micali. – How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, volume 13, n° 4, novembre 1984, pages 850–864. {17}

[88] Robert D. Blumofe et Charles E. Leiserson. – Scheduling multithreaded computations by work stealing. Dans : *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, édité par Shafi Goldwasser, novembre 1994. Pages 356–368. – Los Alamitos, CA, USA. {64}

[89] Tomas J. Boothby et Robert W. Bradshaw. – *Bitslicing and the Method of Four Russians Over Larger Finite Fields*. – Rapport technique, U. of Washington, USA, janvier 2009. http://arxiv.org/abs/0901.1413. {11, 29}

[90] Nigel Boston. – Spaces of constant rank matrices over gf(2). *ELECTRONIC Journal of LINEAR ALGEBRA*, volume 20, janvier 2010, pages 1–5. {51}

[91] Christopher W. Brown (éditeur). – *ISSAC'2007, Proceedings of the 2007 ACM International Symposium on Symbolic and Algebraic Computation, Waterloo, Canada*, July 29 – August 1 2007. – ACM Press, New York. {83, 86, 88, 90}

[92] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad et Michael Stal. – *Pattern-oriented software architecture: a system of patterns*. – New York, NY, USA, John Wiley & Sons, Inc., 1996. {58}

[93] Don Coppersmith et Shmuel Winograd. – Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, volume 9, n° 3, 1990, pages 251–280. {27}

[94] Pierre Courrieu. – Fast computation of Moore-Penrose inverse matrices. *Neural Information Processing - Letters and Reviews*, volume 8, n° 2, août 2005, pages 25–29. {39}

[95] Pierre-Louis Curien et A. Obtulowitz. – Partiality, cartesian closedness and toposes. *Information and Computation*, volume 80, 1989, pages 50–95. {66}

[96] El-Mostafa Daoudi, Thierry Gautier, Aicha Kerfali, Rémi Revire et Jean-Louis Roch. – Algorithmes parallèles à grain adaptatif et applications. *Technique et Science Informatiques*, volume 24, 2005, pages 1—20. {64}

[97] Philippe Delsarte. – Bilinear forms over a finite field, with applications to coding theory. *Journal of Combinatorial Theory, Series A*, volume 25, n° 3, 1978, pages 226–241. {48}

[98] Philippe Delsarte et Jean-Marie Goethals. – Alternating bilinear forms over GF(q). *Journal of Combinatorial Theory, Series A*, volume 19, n° 1, 1975, pages 26–50. {48}

[99] James Demmel, Yozo Hida, William Kahan, Xiaoye S. Li, Sonil Mukherjee et E. Jason Riedy. – Error bounds from extra-precise iterative refinement. *ACM Transactions on Mathematical Software*, volume 32, n° 2, juin 2006, pages 325–351. {52}

[100] Ulrich Dempwolff. – Semifield planes of order 81. *Journal of Geometry*, volume 89, n° 1–2, octobre 2008, pages 1–16. {49}

[101] John D. Dixon. – Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, volume 40, 1982, pages 137–141. {41}

[102] César Dominguez et Dominique Duval. – Diagrammatic logic applied to a parameterization process. *Mathematical Structures in Computer Science*, volume 20, n° 4, 2010, pages 639–654. {67}

[103] Craig C. Douglas, Michael Heroux, Gordon Slishman et Roger M. Smith. – Gemmw: A portable level 3 blas winograd variant of strassen's matrix-matrix multiply algorithm. *Journal of Computational Physics*, volume 110, 1994, pages 1–10. {29, 31}

[104] Jean-Guillaume Dumas (éditeur). – *ISSAC'2006, Proceedings of the 2006 ACM International Symposium on Symbolic and Algebraic Computation, Genova, Italy*, 9–12 juillet 2006. – ACM Press, New York. {83, 89}

[105] Jean-Guillaume Dumas (éditeur). – *TC'2006, Proceedings of Transgressive Computing 2006, Granada, España*, 24–26 avril 2006. – Universidad de Granada, Spain. {75}

[106] Dominique Duval et Jean-Claude Reynaud. – Sketches and computation I: Basic definitions and static evaluation. *Mathematical Structures in Computer Science*, volume 4, n° 2, 1994, pages 185–238. {69}

[107] Dominique Duval et Jean-Claude Reynaud. – Diagrammatic logic and exceptions: an introduction. Dans : *Dagstuhl Seminar Proceedings – MAP05, Mathematics, Algorithms, Proofs*, janvier 2005. {69}

[108] Wayne Eberly. – Black box frobenius decomposition over small fields. Dans : *ISSAC'2000* [197 - Traverso (2000)], pages 106–113. {51}

[109] Wayne Eberly, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann et Gilles Villard. – Solving sparse rational linear systems. Dans : *ISSAC'2006* [104 - Dumas (2006a)], pages 63–70. {40}

[110] Wayne Eberly, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann et Gilles Villard. – Faster inversion and other black box matrix computations using efficient block projections. Dans : *ISSAC'2007* [91 - Brown (2007)], pages 143–150. {40}

[111] Wayne Eberly, Mark Giesbrecht et Gilles Villard. – Computing the determinant and Smith form of an integer matrix. Dans : *FOCS 2000, Proceedings of The* 41^{st} *Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, California*, édité par Avrim Blum, 12–14 novembre 2000. {41, 42}

[112] Ioannis Z. Emiris. – A complete implementation for computing general dimensional convex hulls. *International Journal of Computational Geometry and Applications*, volume 8, n° 2, avril 1998, pages 223–253. {61}

[113] Ioannis Z. Emiris et Victor Y. Pan. – Improved algorithms for computing determinants and resultants. *Journal of Complexity*, volume 21, n° 1, février 2005, pages 43–71. {45}

[114] Charles M. Fiduccia. – An efficient formula for linear recurrences. *SIAM Journal on Computing*, volume 14, n° 1, février 1985, pages 106–112. {55}

[115] Marc Fischlin (éditeur). – *CTRSA'2009, Proceedings of the RSA Conference 2009, Cryptographers' Track, San Francisco, USA*, 20–24 avril 2009. – *Lecture Notes in Computer Science*, volume 5473. {74, 90}

[116] Matteo Frigo, Charles E. Leiserson et Keith H. Randall. – The implementation of the Cilk-5 multithreaded language. Dans : *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation*, juin 1998, pages 212–223. – Montréal, Québec. {64}

[117] Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides. – *Design Patterns: Elements of Reusable Object-Oriented Software*. – Massachusetts, Addison Wesley, 1994. {58, 59}

[118] Victor G. Ganzha, Ernst W. Mayr et Evgenii V. Vorozhtsov (éditeurs). – *CASC'2002, Proceedings of the fifth International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine*, 22–27 septembre 2002. – Technische Universität München, Germany. {75, 84}

[119] Joachim von zur Gathen et Jürgen Gerhard. – *Modern Computer Algebra*. – New York, NY, USA, Cambridge University Press, 1999. {26}

[120] Joachim von zur Gathen et Igor Shparlinski. – Orders of Gauss periods in finite fields. *Applicable Algebra in Engineering, Communication and Computing*, volume 9, 1998, pages 15–24. {16}

[121] Pierrick Gaudry et Emmanuel Thomé. – The mpFq library and implementing curve-based key exchanges. Dans : *SPEED: Software Performance Enhancement for Encryption and Decryption*, 11-12 juin 2007, pages 49–64. – Amsterdam, the Netherlands. {19}

[122] Thierry Gautier, Xavier Besseron et Laurent Pigeon. – KAAPI: A thread scheduling runtime system for data flow computations on cluster of multiprocessors. Dans : *PASCO 2007* [169 - Moreno-Maza et Watt (2007)], pages 15–23. {62}

[123] Wulfram Gerstner et Werner M. Kistler. – *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. – Cambridge University Press, août 2002. {54}

[124] Mark Giesbrecht et Arne Storjohann. – Computing rational forms of integer matrices. *Journal of Symbolic Computation*, volume 34, n° 3, septembre 2002, pages 157–172. {45}

[125] David Ginat. – Algorithmic patterns and the case of the sliding delta. *SIGCSE Bulletin*, volume 36, n° 2, 2004, pages 29–33. {58}

[126] Pascal Giorgi. – *Arithmétique et algorithmique en algèbre linéaire exacte pour la bibliothèque LinBox*. – Thèse de Doctorat, École normale supérieure de lyon, 20 12 2004. {40}

[127] Pascal Giorgi, Claude-Pierre Jeannerod et Gilles Villard. – On the complexity of polynomial matrix computations. Dans : *ISSAC'2003, Proceedings of the 2003 ACM International Symposium on Symbolic and Algebraic Computation, Philadelphia, Pennsylvania, USA*, édité par Rafael Sendra, 3–6 août 2003. Pages 135–142. – ACM Press, New York. {41}

[128] Antoine Girard. – Approximate solutions of ODEs using piecewise linear vector fields. Dans : *CASC'2002* [118 - Ganzha et al. (2002)], pages 107–119. {53}

[129] Antoine Girard. – *Analyse Algorithmique des Systèmes Hybrides*. – Thèse de Doctorat, Institut National Polytechnique de Grenoble, 30 septembre 2004. {54}

[130] A.J. Goldstein et R.L. Graham. – A Hadamard-type bound on the coefficients of a determinant of polynomials. *SIAM Review*, volume 15, 1973, pages 657–658. {45}

[131] Kazushige Goto et Robert A. van de Geijn. – Anatomy of a high-performance matrix multiplication. *ACM Transactions on Mathematical Software*, volume 34, n° 3, mai 2008, pages 1–25. {26}

[132] James Alexander Green. – The characters of the finite general linear groups. *Transactions of the American Mathematical Society*, volume 80, 1955, pages 402–447. {49, 50}

[133] Jaime Gutierrez (éditeur). – *ISSAC'2004, Proceedings of the 2004 ACM International Symposium on Symbolic and Algebraic Computation, Santander, Spain*, 4–7 juillet 2004. – ACM Press, New York. {74, 89, 90}

[134] Chris Heunen et Bart Jacobs. – Arrows, like monads, are monoids. *Electronic Notes in Theoretical Computer Science*, volume 158, 2006, pages 219–236. {66}

[135] Alan L. Hodgkin et Andrew F. Huxley. – A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, volume 177, 1952, pages 500–544. {54}

[136] Alston S. Householder. – *The Theory of Matrices in Numerical Analysis*. – Blaisdell, Waltham, Mass., 1964. {42}

[137] Xiaohan Huang et Victor Y. Pan. – Fast rectangular matrix multiplications and improving parallel matrix computations. Dans : *PASCO '97. Proceedings of the second international symposium on parallel symbolic computation, July 20–22, 1997, Maui, HI*, édité par ACM, 1997. Pages 11–23. – New York, NY 10036, USA. {27}

[138] John Hughes. – Generalising monads to arrows. *Science of Computer Programming*, volume 37, 2000, pages 67–111. {66}

[139] Steven Huss-Lederman, Elaine M. Jacobson, Jeremy R. Johnson, Anna Tsao et Thomas Turnbull. – Implementation of Strassen's algorithm for matrix multiplication. Dans : *Supercomputing '96 Conference Proceedings: November 17–22, Pittsburgh, PA*, édité par ACM, 1996. – New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA. http://doi.acm.org/10.1145/369028.369096. {29, 31}

[140] Martin Hyland, Gordon Plotkin et John Power. – Combining effects: Sum and tensor. *Theoretical Computer Science*, volume 357, 2006, pages 70–99. {66}

[141] Oscar H. Ibarra, Shlomo Moran et Roger Hui. – A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, volume 3, n° 1, mars 1982, pages 45–56. {38, 41}

[142] Samir Jafar, Thierry Gautier, Axel W. Krings et Jean-Louis Roch. – A checkpoint/recovery model for heterogeneous dataflow computations using work-stealing. Dans : *Euro-Par 2005, Parallel Processing, 11th International Euro-Par Conference (11th Euro-Par'05)*, édité par Jose C. Cunha et Pedro D. Medeiros, août-septembre 2005. – *Lecture Notes in Computer Science (LNCS)*, volume 3648, pages 675–684. – Lisbon, Portugal. {64}

[143] Matthias Jantzen. – The power of synchronizing operations on strings. *Theoretical Computer Science*, volume 14, n° 2, mai 1981, pages 127–154. {19}

[144] David Jeffrey (éditeur). – *ISSAC'2008, Proceedings of the 2008 ACM International Symposium on Symbolic and Algebraic Computation, Hagenberg, Austria*, 20–23 juillet 2008. – ACM Press, New York. {74}

[145] Erich Kaltofen. – On computing determinants of matrices without divisions. Dans : *ISSAC'92, Proceedings of the 1992 ACM International Symposium on Symbolic and Algebraic Computation, Berkeley, California*, édité par Paul S. Wang, 27–29 juillet 1992. Pages 342–349. – ACM Press, New York. {47}

[146] Erich Kaltofen. – Challenges of symbolic computation: My favorite open problems. *Journal of Symbolic Computation*, volume 29, n° 6, juin 2000, pages 891–919. {42, 44}

[147] Erich Kaltofen. – An output-sensitive variant of the baby steps/giant steps determinant algorithm. Dans : *ISSAC'2002* [167 - Mora (2002)], pages 138–144. {61}

[148] Erich Kaltofen et Austin Lobo. – Distributed matrix-free solution of large sparse linear systems over finite fields. *Algorithmica*, volume 24, n° 3-4, 1999, pages 331–348. {41}

[149] Erich Kaltofen, Dmitriy Morozov et George Yuhasz. – Generic matrix multiplication and memory management in linbox. Dans : *ISSAC'2005* [154 - Kauers (2005)], pages 216–223. {60}

[150] Erich Kaltofen et B. David Saunders. – On Wiedemann's method of solving sparse linear systems. Dans : *Applied Algebra, Algebraic Algorithms and Error–Correcting Codes (AAECC '91)*, octobre 1991. – *Lecture Notes in Computer Science*, volume 539, pages 29–38. {44}

[151] Erich Kaltofen et Gilles Villard. – Computing the sign or the value of the determinant of an integer matrix, a complexity survey. *Journal of Computational and Applied Mathematics*, n° 164, 2004, pages 133–146. {47}

[152] Erich Kaltofen et Zhengfeng Yang. – On exact and approximate interpolation of sparse rational functions. Dans : *ISSAC'2007* [91 - Brown (2007)], pages 203–210. {57}

[153] William M. Kantor et Michael E. Williams. – Symplectic semifield planes and Z_4-linear codes. *TRANSACTIONS OF THE AMERICAN MATHEMATICAL SOCIETY*, volume 356, n° 3, 2004, pages 895–938. {49}

[154] Manuel Kauers (éditeur). – *ISSAC'2005, Proceedings of the 2005 ACM International Symposium on Symbolic and Algebraic Computation, Beijing, China*, 24–27 juillet 2005. – ACM Press, New York. {74, 86}

[155] Walter Keller-Gehrig. – Fast algorithms for the characteristic polynomial. *Theoretical computer science*, volume 36, 1985, pages 309–317. {42}

[156] Daniel Kunkle et Gene Cooperman. – Biased tadpoles: a fast algorithm for centralizers in large matrix groups. Dans : *ISSAC'2009* [162 - May (2009)], pages 223–230. {49}

[157] Laureano Lambán, Vico Pascual et Julio Rubio. – An object-oriented interpretation of the eat system. *Applicable Algebra in Engineering, Communication and Computing*, volume 14, n° 3, 2003, pages 187–215. {68}

[158] Joachim Lambek et Philip J. Scott. – *Introduction to Higher Order Categorical Logic*. – Cambridge University Press, 1986, *Studies in Advanced Mathematics*, volume 7. {65}

[159] Vincent Lefèvre. – *The Euclidean Division Implemented with a Floating-Point Multiplication and a Floor*. – Rapport technique, INRIA Rhône-Alpes, 2005. http://hal.inria.fr/inria-00000159. {8}

[160] Saunders Mac Lane. – *Categories for the Working Mathematician*. – New York, Springer-Verlag, 1997, 2nde édition, *Graduate Texts in Mathematics*, volume 5. (1st ed., 1971). {65}

[161] George Marsaglia. – Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences of the United States of America*, volume 61, n° 1, 15 septembre 1968, pages 25–28. {21}

[162] John P. May (éditeur). – *ISSAC'2009, Proceedings of the 2009 ACM International Symposium on Symbolic and Algebraic Computation, Seoul, Korea*, 28–31 juillet 2009. – ACM Press, New York. {74, 86}

[163] Robert T. Moenck et John H. Carter. – Approximate algorithms to derive exact solutions to systems of linear equations. Dans : *Proceedings of the International Symposium on Symbolic and Algebraic Manipulation (EUROSAM '79)*, édité par Edward W. Ng, juin 1979. – *LNCS*, volume 72, pages 65–73. – Marseille, France. {41}

[164] Eugenio Moggi. – Computational lambda-calculus and monads. Dans : *Logic In Computer Science (LICS)*, 1989. Pages 14–23. – IEEE Press. {66}

[165] Eugenio Moggi. – Notions of computation and monads. *Information and Computation*, volume 93, 1991, pages 55–92. {66}

[166] Eugenio Moggi. – A semantics for evaluation logic. *Fundamenta Informaticae*, volume 22, 1995, pages 117–152. {66}

[167] Teo Mora (éditeur). – *ISSAC'2002, Proceedings of the 2002 ACM International Symposium on Symbolic and Algebraic Computation, Lille, France*, 7–10 juillet 2002. – ACM Press, New York. {74, 86, 89}

[168] Marc Moreno-Maza et Jean-Louis Roch (éditeurs). – *Parallel Symbolic Computation'2010*, 21–23 juillet 2010. – Université de Grenoble, France. {74}

[169] Marc Moreno-Maza et Stephen Watt (éditeurs). – *Parallel Symbolic Computation'07*, 26–27 juillet 2007. – Waterloo University, Ontario, Canada. {75, 84}

[170] Ronald C. Mullin, I. M. Onyszchuk, Scott A. Vanstone et Richard Michael Wilson. – Optimal normal bases in GF(p^n). *Discrete Applied Mathematics*, volume 22, n° 2, 1989, pages 149–161. {19, 21}

[171] David R. Musser et Atul Saini. – *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. – Reading (MA), USA, Addison-Wesley, 1996. {59}

[172] Hong Diep Nguyen et Nathalie Revol. – Solving and certifying the solution of a linear system. Dans : *The 13th GAMM - IMACS International Symposium*

on Scientific Computing, Computer Arithmetic and Verified Numerical Computations, U. of Texas at El Paso, USA, édité par Martine Ceberio et Vladik Kreinovich, 29–3 octobre 2008. {52}

[173] Pierre Nicodème, Bruno Salvy et Philippe Flajolet. – Motif statistics. *Theoretical Computer Science*, volume 287, n° 2, septembre 2002, pages 593–617. {55}

[174] Grégory Nuel. – Waiting time distribution for pattern occurrence in a constrained sequence: an embedding markov chain approach. *Discrete Mathematics and Theoretical Computer Science*, 2010. – in press. {55}

[175] Victor Y. Pan et Franco P. Preparata. – Work-preserving speed-up of parallel matrix computations. *SIAM Journal on Computing*, volume 24, n° 4, 1995, pages 811–821. {37}

[176] Clément Pernet. – *Calcul du polynôme caractéristique sur des corps finis*. – Diplôme d'Études Approfondies, University of Delaware, juin 2003. {43}

[177] Clément Pernet. – *Algèbre linéaire exacte efficace : le calcul du polynôme caractéristique*. – Thèse de Doctorat, Université Joseph-Fourier - Grenoble I, 27 septembre 2006. {27, 40}

[178] Clément Pernet, Aude Rondepierre et Gilles Villard. – *Computing the Kalman form*. – Rapport technique, IMAG-ccsd-00009558, ArXiv:cs/0510014, octobre 2005. {53}

[179] Clément Pernet et Arne Storjohann. – Faster algorithms for the characteristic polynomial. Dans : *ISSAC'2007* [91 - Brown (2007)]. {43}

[180] Clément Pernet et Arne Storjohann. – *Frobenius form in expected matrix multiplication time over sufficiently large fields*. – Rapport technique, novembre 2007. http://www.cs.uwaterloo.ca/~astorjoh/cpoly.pdf. {54}

[181] John Power et Edmund Robinson. – Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, volume 7, 1997, pages 453–468. {66}

[182] Jean-Louis Roch, Daouda Traore et Julien Bernard. – On-line adaptive parallel prefix computation. Dans : *Europar 2006*, août 2006. – *LNCS 4128*. – Dresden, Germany. {64}

[183] Aude Rondepierre. – *Algorithmes hybrides pour le contrôle optimal des systèmes non linéaires*. – Thèse de Doctorat, Université Joseph-Fourier - Grenoble I, 18 juillet 2006. {53}

[184] Jacques Roubaud. – Réflexions historiques et combinatoires sur la n-ine autrement dit quenine. *La bibliothèque Oulipienne*, volume 5, n° 66, 2000, pages 99–124. – Contribution à la réunion 395 de l'Oulipo, le 17 septembre 1993. {19}

[185] Ignacio Fernández Rúa, Elías F. Combarro et José Ranilla. – Classification of semifields of order 64. *Journal of Algebra*, volume 322, n° 11, 2009, pages 4011–4029. {49, 51}

[186] Julio Rubio, Françis Sergeraert et Yvon Siret. – Overview of eat. *Symbolic and Algebraic Computation (SAC) Newsletter*, volume 3, 1998, pages 69–79. {68}

[187] David Saunders et Zhendong Wan. – Smith Normal Form of dense integer matrices fast algorithms into practice. Dans : *ISSAC'2004* [133 - Gutierrez (2004)], pages 274–281. {41, 42}

[188] Michèle Schatzman. – Les bonnes matrices sont rares. *MATAPLI, bulletin de la SMAI*, volume 78, octobre 2005, pages 41–56. {52}

[189] Victor Shoup. – Searching for primitive roots in finite fields. *Mathematics of Computation*, volume 58, n° 197, janvier 1992, pages 369–380. {16}

[190] Victor Shoup. – *A computational introduction to number theory and algebra*. – Cambridge University Press, 2005, xvi + 517 pages. {8, 23}

[191] Victor Shoup. – NTL 5.5.2: A library for doing number theory, 2009. `www.shoup.net/ntl`. {8}

[192] Arne Storjohann. – Computing the frobenius form of a sparse integer matrix. – avril 2000. Personal communication, Personal communication. {47}

[193] Arne Storjohann. – High-order lifting. Dans : *ISSAC'2002* [167 - Mora (2002)], pages 246–254. {55}

[194] Arne Storjohann. – The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity*, volume 21, n° 4, 2005, pages 609–650. {41, 42}

[195] Volker Strassen. – Gaussian elimination is not optimal. *Numerische Mathematik*, volume 13, 1969, pages 354–356. {26}

[196] Arnaud Tonnelier. – *Dynamique non-linéaire et bifurcations en neurosciences mathématiques*. – Thèse de Doctorat en mathématiques appliquées, Université Joseph Fourier, Grenoble, France, 25 octobre 2001. {54}

[197] Carlo Traverso (éditeur). – *ISSAC'2000, Proceedings of the 2000 ACM International Symposium on Symbolic and Algebraic Computation, Saint-Andrews, Scotland*, 6–9 août 2000. – ACM Press, New York. {74, 83}

[198] William J. Turner. – A block Wiedemann rank algorithm. Dans : *ISSAC'2006* [104 - Dumas (2006a)], pages 332–339. {41}

[199] Anna Urbańska-Marzalek. – *Hybrid and adaptive algorithms in exact linear algebra*. – Thèse de Doctorat, Université Joseph-Fourier - Grenoble I, 27 avril 2010. {41}

[200] Joris van der Hoeven. – The truncated Fourier transform and applications. Dans : *ISSAC'2004* [133 - Gutierrez (2004)], pages 290–296. {15}

[201] Richard S. Varga. – *Geršgorin and his circles*. – Berlin, Springer-Verlag, 2004, *Springer Series in Computational Mathematics*, volume 36, x+226 pages. {46}

[202] Pierrick Vignard. – *Anneaux finis et calcul du rang*. – Diplôme d'Études Approfondies, Université Joseph Fourier, Grenoble, juin 2003. {7}

[203] Gilles Villard. – Computing the Frobenius normal form of a sparse matrix. Dans : *CASC'00, Proceedings of the Third International Workshop on Computer Algebra in Scientific Computing, Samarkand, Uzbekistan*, édité par Victor G. Ganzha, Ernst W. Mayr et Evgenii V. Vorozhtsov, 5–9 octobre 2000, pages 395–407. {44}

[204] Gilles Villard. – Certification of the QR factor R and of lattice basis reducedness. Dans : *ISSAC'2007* [91 - Brown (2007)], pages 361–368. {52}

[205] Zhendong Wan. – *Computing the Smith Forms of Integer Matrices and Solving Related Problems*. – Thèse de Doctorat, University of Delaware, USA, 2005. {41, 42}

[206] André Weimerskirch, Douglas Stebila et Sheueling Chang Shantz. – Generic GF(2) arithmetic in software and its application to ECC. Dans : *Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003*, édité par Reihaneh Safavi-Naini et Jennifer Seberry, 2003. – *Lecture Notes in Computer Science*, volume 2727, pages 79–92. – Springer. {13}

[207] R. Clint Whaley, Antoine Petitet et Jack J. Dongarra. – Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, volume 27, n° 1–2, janvier 2001, pages 3–35. – `http://www.netlib.org/utk/people/JackDongarra/PAPERS/atlas_pub.pdf`. {26}

[208] Douglas H. Wiedemann. – Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, volume 32, n° 1, janvier 1986, pages 54–62. {44}

[209] Glynn Winskel. – *The Formal Semantics of Programming Languages: An Introduction*. – MIT Press, 1993. {67}

[210] Masayuki Yoshino, Katsuyuki Okeya et Camille Vuillaume. – Recursive double-size modular multiplications without extra cost for their quotients. Dans : *CTRSA'2009* [115 - Fischlin (2009)], pages 340–356. {18}