Calcul multiprécision

Michel Quercia, avril 2000
quercia@cal.enst.fr

1 Introduction

Le calcul multiprécision concerne les opérations sur des nombres dépassant la capacité des variables entières ou flottantes disponibles dans les langages de programmation usuels. Il y a au moins deux domaines importants consommateurs de nombres multiprécision : la cryptographie où l'on manipule des entiers de plusieurs centaines de bits et le calcul formel où la seule limite à la taille des nombres traités qui soit tolérable est celle imposée par la capacité mémoire. L'objet de cet exposé est de présenter quelques algorithmes implémentant efficacement les opérations de base sur les nombres multiprécision : multiplication, division et racine carrée.

La représentation naturelle d'un nombre multiprécision en mémoire est son écriture dans une certaine base, qui sera notée x, et quelques informations annexes telles que le signe ou la position d'une virgule flottante. Une conséquence pratique de cette représentation est que les algorithmes de calcul ont accès aux chiffres de leurs opérandes et doivent fournir leur résultat sous forme de liste de chiffres. Une conséquence théorique est que l'on peut approcher le modèle de nombre entier par celui de polynôme conceptuellement plus simple : il n'y a pas de problème de retenues venant obscurcir l'exposé d'un algorithme. Dans certains cas on raisonnera sur des séries formelles plutôt que sur des polynômes pour s'abstraire des contingences de degré.

2 Multiplication

La multiplication est le composant fondamental des bibliothèques de calcul multiprécision : les algorithmes présentés aux chapitres suivants (division et racine carrée) se ramènent tous à des suites d'additions/soustractions et de multiplications. L'addition et la soustraction s'implémentent de façon triviale et efficace par simple balayage des chiffres des opérandes, mais il n'en va pas de même pour la multiplication et les performances de l'algorithme de multiplication utilisé ont une influence déterminante sur les performances des algorithmes « clients ».

Daniel Bernstein recense dans [1] une vingtaine d'algorithmes de multiplication avec indications historiques et interprétation en termes de morphismes d'anneaux.

2.1 La multiplication de Karatsuba

L'algorithme de Karatsuba est à l'heure actuelle l'algorithme de choix pour multiplier deux opérandes dont la taille va de quelques coefficients jusqu'à plusieurs milliers de coefficients. Au delà, la multiplication de Toom et les multiplications par transformée de Fourier discrète sont plus efficaces. L'identité de Karatsuba consiste à calculer le produit de deux polynômes de degré au plus un à l'aide de trois multiplications entre scalaires au lieu des quatre que l'on devrait faire si l'on développait naïvement le produit :

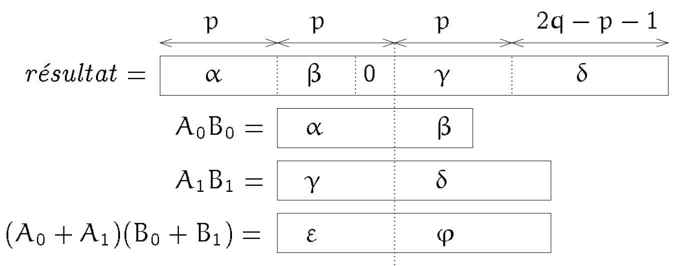
$$(a_0 + a_1x)(b_0 + b_1x) = a_0b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1)x + a_1b_1x^2.$$

Utilisée récursivement selon le principe diviser pour régner, l'identité de Karatsuba conduit à un algorithme de multiplication de deux polynômes de longueur n avec $\Theta(n^{\log_2(3)})$ opérations entre coefficients. Pour cela, on divise les polynômes A et B à multiplier en deux « moitiés » :

$$\begin{aligned} A &= a_0 + \ldots + a_{n-1}x^{n-1} \\ &= (a_0 + \ldots + a_{\lfloor n/2\rfloor-1}x^{\lfloor n/2\rfloor-1}) + (a_{\lfloor n/2\rfloor} + \ldots + a_{n-1}x^{\lceil n/2\rceil-1})x^{\lfloor n/2\rfloor} \\ &= A_0 + A_1x^{\lfloor n/2\rfloor} \end{aligned}$$

et de même pour B, puis on calcule le produit $(A_0 + A_1x^{\lfloor n/2\rfloor})(B_0 + B_1x^{\lfloor n/2\rfloor})$ à l'aide de l'identité de Karatsuba en évaluant les produits A_0B_0, A_1B_1 et $(A_0 + A_1)(B_0 + B_1)$ selon la même méthode.

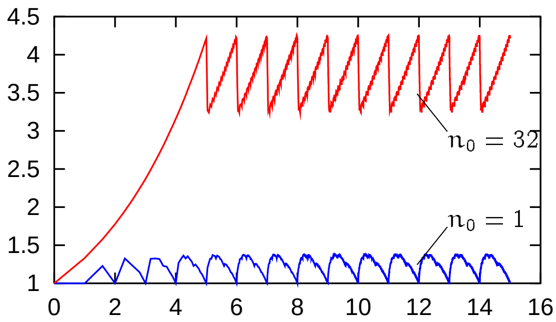
Karamul$(A, B, n) = (n = \text{longueur}(A) = \text{longueur}(B))$
  si $n \leqslant n_0$ *effectuer le produit par développement complet*
  sinon :
    soient $p = \lfloor n/2\rfloor$ *et* $q = \lceil n/2\rceil$
    décomposer A *et* B *en* $A_0 : A_1$ *et* $B_0 : B_1$ *de longueurs* p, q, p, q
    décomposer le résultat en $\alpha : \beta : 0 : \gamma : \delta$ *selon la figure ci-dessous*
    allouer une mémoire auxiliaire $\varepsilon : \varphi$ *de taille* $2q - 1$
    calculer $A_0 + A_1$ *et* $B_0 + B_1$ *dans la mémoire résultat*
    $\varepsilon : \varphi \longleftarrow$ *Karamul*$(A_0 + A_1, B_0 + B_1, q)$
    $\alpha : \beta \longleftarrow$ *Karamul*(A_0, B_0, p)
    $\gamma : \delta \longleftarrow$ *Karamul*(A_1, B_1, q)
    $\gamma \longleftarrow \gamma - \beta$
    $\beta : 0 \longleftarrow \varepsilon - \alpha - \gamma$
    $\gamma : \delta \longleftarrow \gamma : \delta - \delta + \varphi$
  fin sinon



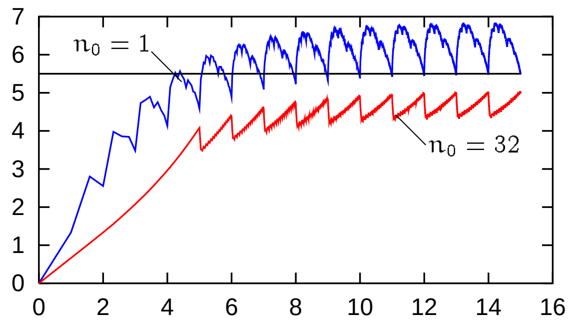
Si $K_{\text{add}}(n)$ et $K_{\text{mul}}(n)$ désignent les nombres d'additions/soustractions et de multiplications entre coefficients effectuées lors du calcul récursif de ab et si $K_{\text{mem}}(n)$ désigne la quantité de mémoire auxiliaire nécessaire alors on a les relations de récurrence :

$$\begin{aligned} K_{\text{mul}}(n) &= K_{\text{mul}}(\lfloor n/2\rfloor) + 2K_{\text{mul}}(\lceil n/2\rceil)\ ; \\ K_{\text{add}}(n) &= K_{\text{add}}(\lfloor n/2\rfloor) + 2K_{\text{add}}(\lceil n/2\rceil) + 3\lfloor n/2\rfloor + 4\lceil n/2\rceil - 3\ ; \\ K_{\text{mem}}(n) &= \max(K_{\text{mem}}(\lfloor n/2\rfloor), K_{\text{mem}}(\lceil n/2\rceil)) + 2\lceil n/2\rceil - 1. \end{aligned}$$

avec $K_{\text{mul}}(n) = n^2$, $K_{\text{add}}(n) = n(n-1)$ et $K_{\text{mem}}(n) = 0$ pour $n \leqslant n_0$, n_0 étant le seuil choisi expérimentalement où l'on arrête la récursion pour effectuer les multiplications par développement complet.



$y = K_{\text{mul}}(2^x)/3^x$



$y = K_{\text{add}}(2^x)/3^x$

Comme l'illustrent les courbes ci-dessus, on a $K_{\text{mul}}(n) = \Theta(n^{\log_2(3)})$ et $K_{\text{add}}(n) = \Theta(n^{\log_2(3)})$ mais les rapports $K_{\text{mul}}(n)/n^{\log_2(3)}$ et $K_{\text{add}}(n)/n^{\log_2(3)}$ *ne convergent pas* quand n tend vers l'infini. Par ailleurs, on a par récurrence : $K_{\text{mem}}(n) \leqslant 2n + \lfloor \log_2(n) \rfloor$; donc la quantité de mémoire auxiliaire nécessaire au calcul de AB est du même ordre que la taille du résultat.

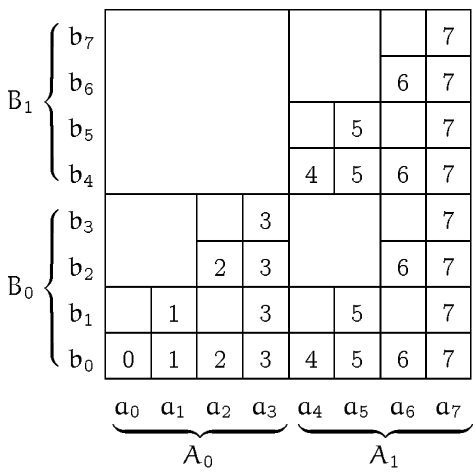
On peut aussi décomposer les polynômes à multiplier en parties paire et impaire :

$$A = (a_0 + a_2x^2 + \dots) + x(a_1 + a_3x^2 + \dots) = A'_0(x^2) + xA'_1(x^2)$$

et appliquer l'identité de Karatsuba à cette décomposition pour calculer AB, les produits des polynômes en x^2 étant calculés récursivement. Il n'y a pas d'intérêt pratique à utiliser cette décomposition pair-impair plutôt que la décomposition officielle en deux blocs adjacents, c'est même contre-indiqué en termes d'efficacité de la mémoire cache, mais elle donne lieu à une constatation amusante : les deux méthodes de décomposition conduisent à effectuer *exactement les mêmes multiplications* entre coefficients si l'on descend jusqu'au plus bas niveau de récursion (c'est-à-dire avec $n_0 = 1$).

Multiplication en ligne

La multiplication de Karatsuba peut être effectuée *en ligne*, c'est-à-dire que l'on peut déterminer le n-ème coefficient du produit AB dès que les n premiers coefficients de A et de B sont connus. Il « suffit » d'organiser les calculs de la manière suivante :



- A l'étape 0 on calcule a_0b_0.
- A l'étape 1 on calcule a_1b_1 et $(a_0 + a_1)(b_0 + b_1)$ ce qui permet d'en déduire le deuxième coefficient, $a_0b_1 + a_1b_0$, de AB.
- A l'étape 2 on calcule a_2b_2 et $(a_0 + a_2)(b_0 + b_2)$ d'où l'on déduit le troisième coefficient de AB : $(a_0b_2 + a_2b_0) + a_1b_1$.
- A l'étape 3 on calcule a_3b_3, $(a_2 + a_3)(b_2 + b_3)$ d'où l'on déduit $a_2b_3 + a_3b_2$, puis on calcule $(a_1+a_3)(b_1+b_3)$ d'où l'on déduit $a_1b_3+a_3b_1$, et enfin on calcule $(a_0+a_1+a_2+a_3)(b_0+b_1+b_2+b_3)$

auquel on retranche $(a_2+a_3)(b_2+b_3)$, $(a_0+a_1)(b_0+b_1)$, $a_0b_2+a_2b_0$ et $a_1b_3+a_3b_1$ déjà calculés pour obtenir $a_0b_3+a_1b_2+a_2b_1+a_3b_0$ qui est le quatrième coefficient de AB.
- Les étapes 4 à 7 calculent A_1B_1 et $(A_0+A_1)(B_0+B_1)$ de la même manière, et $A_0B_1+A_1B_0$ par différence, tout celà en ligne.
- etc.

L'intérêt du calcul en ligne est de permettre d'effectuer des multiplications pour lesquelles les opérandes sont incomplètement connus au départ, leurs coefficients étant calculés en fonction des premiers coefficients du produit. Par exemple on peut obtenir le début du développement en série entière de la fonction tangente par intégration de l'équation différentielle : $y' = 1+y^2$. On calcule y^2 par multiplication en ligne et on primitive $1+y^2$ ce qui fournit au multiplicateur en ligne les coefficients de y juste avant la première fois où il en a besoin.

Joris Van der Hoeven présente dans [10] plusieurs algorithmes de multiplication en ligne (qu'il appelle *multiplication détendue*), dont celui exposé ci-dessus, et leurs applications aux opérations usuelles sur les séries formelles : composition et intégration d'équations différentielles. Tel qu'il est décrit, l'algorithme de multiplication de Karatsuba en ligne calcule les n premiers coefficients du produit de deux séries en temps $\Theta(n^{\log_2(3)})$ et en mémoire $\Theta(n\ln n)$. Van der Hoeven présente aussi une autre décomposition, dans le cadre de la multiplication par transformation de Fourier discrète, permettant d'effectuer une multiplication en ligne en mémoire $\Theta(n)$. Si l'on utilise la multiplication de Karatsuba au lieu de la multiplication par FFT alors cette deuxième méthode n'effectue que $\frac{3}{2}K_{\mathrm{mul}}(n)$ multiplications pour fournir les n premiers coefficients du produit (lorsque n est de la forme 2^k-1), donc est a priori plus intéressante que l'implémentation en ligne décrite plus haut, compte tenu du moindre besoin en mémoire auxiliaire.

Multiplication de nombres entiers

Lorsque l'on applique l'algorithme de Karatsuba aux produits de grands entiers, il faut tenir compte des retenues possibles lors du calcul de A_0+A_1 et B_0+B_1. Cela ne présente pas de difficulté particulière, mais on peut aussi éviter complètement l'apparition de retenues en utilisant l'identité de Knuth :

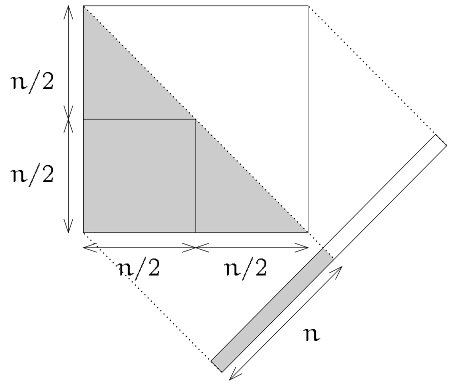
$$(a_0+a_1x)(b_0+b_1x) = a_0b_0 + (a_0b_0+a_1b_1-(a_0-a_1)(b_0-b_1))x + a_1b_1x^2.$$

Cette relation ne provoque pas de retenues dans le calcul de A_0-A_1 et B_0-B_1, en contrepartie il faut tenir compte des signes de ces différences. Knuth présente sa méthode dans [7] en signalant qu'un certain Karatsuba avait suggéré *a similar but more complicated method* sans préciser ni la méthode de Karatsuba ni en quoi elle était plus compliquée. Van der Hoeven donne l'identité de Karatsuba en indiquant que celle publiée dans l'article de Karatsuba et Ofman est *similar but slightly more complicated.* Ce sont les deux seuls auteurs parmi ceux que j'ai consultés qui suggèrent que l'identité de Karatsuba se serait pas d'origine. N'ayant pas eu accès à l'article de Karatsuba et Ofman ([5]) je ne peux pas me prononcer.

La méthode de Karatsuba a été généralisée par Toom en 1963 qui a remarqué que l'identité de Karatsuba revenait en fait à déterminer les coefficients d'un polynôme $P = AB$ de degré au plus deux à l'aide de trois valeurs de P, à savoir $P(0)$, $P(1)$ et $P(\infty)$ (c'est-à-dire le coefficient dominant de P), chacune de ces valeurs étant obtenue par une multiplication scalaire. On peut de la même manière calculer les coefficients d'un polynôme $P = AB$ de degré au plus r à l'aide des valeurs de ce polynôme en $r+1$ points avec donc $r+1$ multiplications scalaires. En choisissant r variable avec n, Toom obtient ainsi un algorithme de multiplication de nombres ou de polynômes de longueur n avec une complexité $n\exp(O(\sqrt{\ln n}))$ donc quasi-linéaire. Les méthodes de multiplication par transformée de Fourier discrète relèvent aussi de la même philosophie : calculer les valeurs du polynôme produit en des points distincts bien choisis et en déduire les coefficients de ce polynôme par interpolation de Lagrange.

2.2 La multiplication tronquée de MULDERS

Lorsque l'on travaille sur des développements limités ou sur des nombres codés en virgule flottante, le calcul du produit complet de deux polynômes ou de deux nombres de longueur n n'est pas pertinent : il fournit $2n-1$ ou $2n$ termes dont seuls les n premiers sont réellement significatifs. Le problème de la multiplication tronquée est de calculer ces n premiers termes plus rapidement que si l'on calculait le produit complet.



Une idée simple est de décomposer le produit tronqué $n \times n$ à calculer en un produit complet $n/2 \times n/2$ et deux produits tronqués $n/2 \times n/2$. Cette idée est hélas trop simple. Si l'on note M_n le temps de calcul d'un produit complet $n \times n$ et T_n le temps de calcul d'un produit tronqué de même taille alors on a l'équation de récurrence :

$$T_n = M_{n/2} + 2T_{n/2} + O(n)$$

en négligeant les détails dûs aux divisions inégales lorsque n est impair. On en déduit :

$$T_n = M_{n/2} + 2M_{n/4} + 4M_{n/8} + \ldots + O(n \ln n).$$

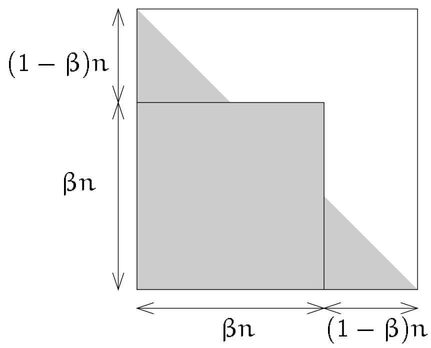
Si l'on utilise la multiplication de KARATSUBA, on a $M_{n/2^k} \approx M_n/3^k$ et donc :

$$T_n \approx M_n(\frac{1}{3} + \frac{2}{9} + \frac{4}{27} + \ldots) + O(n \ln n) = M_n + O(n \ln n).$$

Le gain en temps par rapport au calcul du produit complet est donc nul ! Avec une multiplication par transformation de FOURIER rapide c'est même pire :

$$M_{n/2^k} \approx M_n/2^k \Longrightarrow T_n \approx M_n \log_2(n).$$

C'est en 1998 que THOM MULDERS ([8]) a l'idée de déborder de la diagonale :



β est un paramètre compris entre $\frac{1}{2}$ et 1 à choisir en fonction de l'algorithme de multiplication utilisé pour les produits complets. On a alors l'équation de récurrence :

$$\mathrm{Muld}_n = M_{\beta n} + 2\mathrm{Muld}_{(1-\beta)n} + O(n).$$

Dans le cas de la multiplication de KARATSUBA, en considérant que $M_{\beta n} \sim \beta^{\log_2(3)} M_n$, MULDERS obtient :

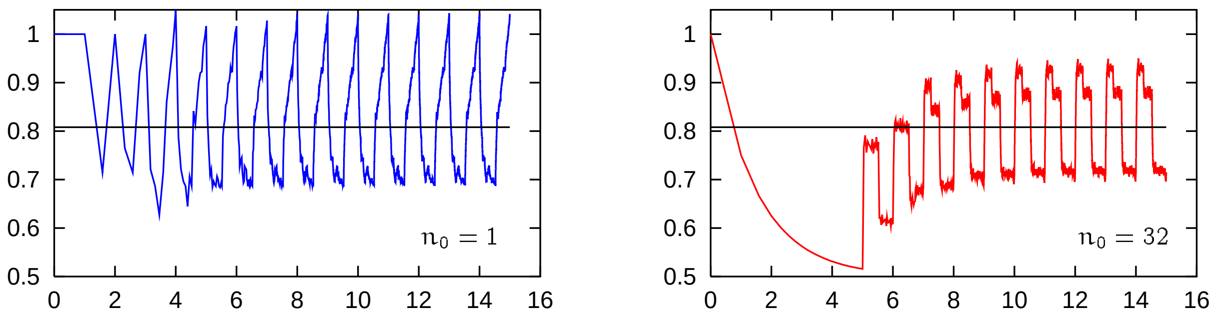
$$\mathrm{Muld}_n \sim \frac{\beta^{\log_2(3)}}{1 - 2(1-\beta)^{\log_2(3)}} M_n = f(\beta) M_n,$$

et un calcul élémentaire montre que $f(\beta)$ est minimal pour $\beta = 1 - 2^{-1/(\log_2(3)-1)} \approx 0.694$ et vaut environ 0.808 dans ce cas (on atteint presque le minimum avec la valeur plus simple $\beta = \frac{2}{3}$: $f(\frac{2}{3}) \approx 0.81$). Ainsi, avec la décomposition de MULDERS, on peut théoriquement calculer un produit tronqué en 81% du temps de calcul du produit complet. Dans le cas de la multiplication par transformation de FOURIER rapide la valeur optimale de β est 1, c'est-à-dire qu'il reste préférable de calculer le produit complet.

Malheureusement, comme le remarque PAUL ZIMMERMANN dans [12] un an plus tard, le raisonnement de MULDERS repose sur l'hypothèse $M_{\beta n} \sim \beta^{\log_2(3)} M_n$ *qui est fausse* car la suite $(M_n / n^{\log_2(3)})$ n'est pas convergente. En prenant la relation de récurrence exacte pour le compte de multiplications :

$$\mathrm{Muld}_{\mathrm{mul}}(n) = K_{\mathrm{mul}}(\lceil \beta n \rceil) + 2\mathrm{Muld}_{\mathrm{mul}}(n - \lceil \beta n \rceil) \text{ si } n > n_0, \qquad \mathrm{Muld}_{\mathrm{mul}}(n) = \frac{n(n+1)}{2} \text{ si } n \leqslant n_0,$$

on obtient les courbes suivantes pour le rapport $\mathrm{Muld}_{\mathrm{mul}}(n)/K_{\mathrm{mul}}(n)$:



$y = \mathrm{Muld}_{\mathrm{mul}}(2^x)/K_{\mathrm{mul}}(2^x)$ $\qquad$ $y = \mathrm{Muld}_{\mathrm{mul}}(2^x)/K_{\mathrm{mul}}(2^x)$

Pour $n_0 = 1$ le rapport $\mathrm{Muld}_{\mathrm{mul}}(n)/K_{\mathrm{mul}}(n)$ oscille entre 0.7 et 1.04 donc l'accélération de MULDERS peut très bien se trouver être en fait un ralentissement, mais aussi, heureuse surprise, une accélération meilleure que prévu. Pour $n_0 = 32$ la situation est plus régulière et le rapport $\mathrm{Muld}_{\mathrm{mul}}(n)/K_{\mathrm{mul}}(n)$ oscille entre 0.7 et 0.93, au moins y a-t-il toujours accélération. A l'heure actuelle on ne sait pas si la méthode de MULDERS est réellement efficace, le compte du nombre de multiplications n'est pas nécessairement représentatif du temps de calcul, et des mesures effectives de temps de calcul seraient trop dépendantes de la machine utilisée pour fournir des indications incontestables.

Multiplication tronquée de nombres entiers

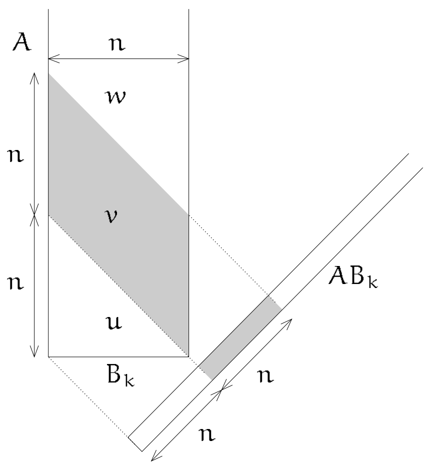
Le cas des nombres entiers soulève un autre problème : du fait que la décomposition de MULDERS déborde de la diagonale, les n premiers chiffres du produit tronqué contiennent des retenues indésirables provenant des chiffres suivants. Il a été suggéré de considérer les entiers à multiplier comme étant des polynômes, chaque chiffre étant un coefficient, et de conduire les calculs de manière polynomiale c'est-à-dire sans propager les retenues. A la fin du calcul il suffit de tronquer le polynôme obtenu à n coefficients et de le convertir en nombre par propagation des retenues locales. Cette méthode a toutefois l'inconvénient de nécessiter une mémoire auxiliaire $\Theta(n \ln n)$ puisqu'il faut réserver $\log_2(n)$ bits de retenue pour chaque coefficient du produit calculé.

2.3 Le produit médian de Hanrot et Zimmermann

La notion de produit médian a été introduite en 1999 par Guillaume Hanrot et Paul Zimmermann dans [12] à propos de l'approximation de l'inverse d'une série formelle par la méthode de Newton. Soit $A = \sum_{i=0}^{\infty} a_i x^i$ une série formelle avec $a_0 \neq 0$. On calcule la série formelle $B = 1/A$ par approximations successives à l'aide de la suite récurrente :

$$B_0 = 1/a_0, \qquad B_{k+1} \equiv B_k + B_k(1 - AB_k) \pmod{x^{2^{k+1}}} \quad \text{avec } \deg(B_{k+1}) < 2^{k+1}.$$

On a alors : $1 - AB_{k+1} \equiv (1 - AB_k)^2 \bmod x^{2^{k+1}}$ d'où par récurrence : $\mathrm{val}(1 - AB_k) \geqslant 2^k$, c'est-à-dire que les coefficients de B_k sont égaux aux 2^k premiers coefficients de $1/A$ (voir la section **3.2** pour une description complète de l'algorithme d'inversion). Notons $n = 2^k$. Pour calculer $1 - AB_k \bmod x^{2^{k+1}}$ il suffit de calculer la partie v grisée ci-dessous :



En effet le produit tronqué, u, de A et de B_k est égal à 1 car $\mathrm{val}(1 - AB_k) \geqslant n$ et la partie w est de valuation au moins $2n$, donc est nulle modulo x^{2n}. Pour calculer v on a au moins trois stratégies :

- Calculer le produit complet $A_k B_k$ où A_k est la troncature à l'ordre $2n$ de A et en extraire les n coefficients du milieu. Cette méthode est à l'origine de l'appellation « produit médian $n \times 2n$ » pour v. Le coût de la méthode est $2M_n$, M_n étant le temps de calcul d'un produit complet $n \times n$.

- Décomposer v en deux produits tronqués tête-bêche et les calculer par la méthode de Mulders. Le coût est environ $1.62M_n$ en admettant la loi de Mulders : $\mathrm{Muld}_n \approx 0.81M_n$ pour la multiplication de Karatsuba. Cette loi est inexacte, mais elle n'était pas encore contestée à l'époque.

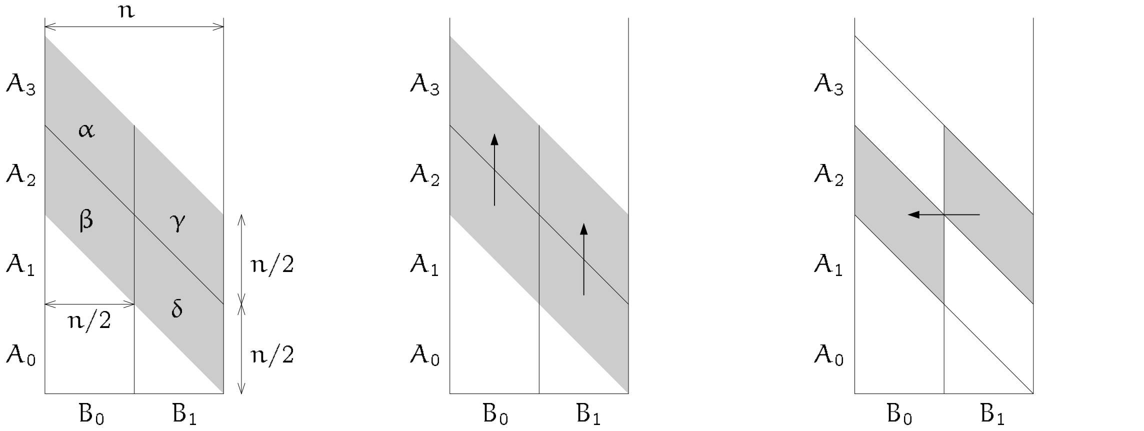
- Calculer le produit médian à l'aide de l'algorithme RMP (recursive middle product) de Hanrot et Zimmermann exposé ci-dessous. Le coût est de l'ordre de M_n.

Hanrot et Zimmermann remarquent que le produit médian d'une série A par un polynôme B de longueur n, noté $A * B$, est bilinéaire par rapport à A et B et ils en déduisent une décomposition permettant de ramener le calcul d'un produit médian $n \times 2n$ à trois calculs de produits médians $n/2 \times n$ (voir figure page suivante). Si n est une puissance de 2 on a les relations de récurrence suivantes pour les nombres $R_{\mathrm{mul}}(n)$ et $R_{\mathrm{add}}(n)$ de multiplications et d'additions/soustractions effectuées et la taille mémoire auxilliaire $R_{\mathrm{mem}}(n)$ nécessaire pour calculer récursivement un produit médian $n \times 2n$:

$$\begin{aligned} R_{\mathrm{mul}}(n) &= 3R_{\mathrm{mul}}(n/2)\ ; \\ R_{\mathrm{add}}(n) &= 3R_{\mathrm{add}}(n/2) + 3n\ ; \\ R_{\mathrm{mem}}(n) &= R_{\mathrm{mem}}(n/2) + n. \end{aligned}$$

Dans les mêmes conditions les équations de récurrence pour la multiplication de Karatsuba sont :

$$\begin{aligned} K_{\mathrm{mul}}(n) &= 3K_{\mathrm{mul}}(n/2)\ ; \\ K_{\mathrm{add}}(n) &= 3K_{\mathrm{add}}(n/2) + \frac{7}{2}n - 3\ ; \\ K_{\mathrm{mem}}(n) &= K_{\mathrm{mem}}(n/2) + n - 1. \end{aligned}$$



$$\alpha + \beta = (A_1{:}A_2 + A_2{:}A_3) * B_0$$
$$\gamma + \delta = (A_0{:}A_1 + A_1{:}A_2) * B_1$$

$$\beta - \gamma = (A_1{:}A_2) * (B_0 - B_1)$$

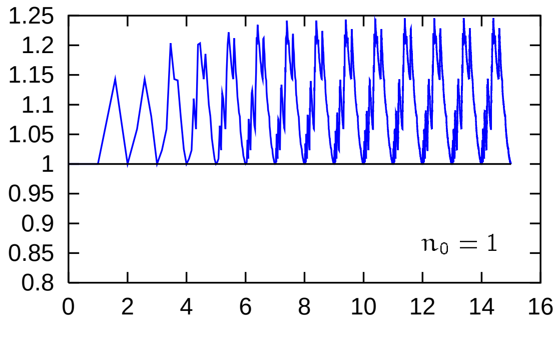
$$\begin{aligned} A * B &= (\beta + \delta) : (\alpha + \gamma) \\ &= ((\beta - \gamma) + (\gamma + \delta)) : ((\alpha + \beta) - (\beta - \gamma)) \end{aligned}$$

et les conditions initiales sont les mêmes : $R_{\mathrm{mul}}(n) = K_{\mathrm{mul}}(n) = n^2$, $R_{\mathrm{add}}(n) = K_{\mathrm{add}}(n) = n(n-1)$, $R_{\mathrm{mem}}(n) = K_{\mathrm{mem}}(n) = 0$ pour $n \leqslant n_0$. Donc, *lorsque* n *est une puissance de* 2, le calcul récursif d'un produit médian $n \times 2n$ selon la décomposition Hanrot-Zimmermann requiert autant de multiplications et moins d'additions que le calcul d'un produit complet $n \times n$ par la méthode de Karatsuba ; les besoins en mémoire auxilliaire des deux algorithmes sont comparables : $2n + O(\ln(n))$.

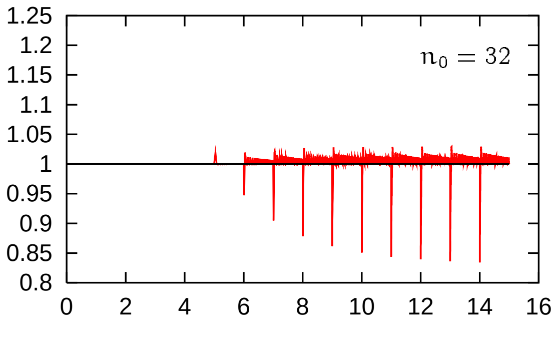
Lorsque n n'est pas une puissance de 2 l'analyse est plus délicate. En particulier si n est impair il y a au moins trois stratégies possibles :

- prolonger les opérandes par des zéros et calculer un produit médian $(n+1) \times (2n+2)$;
- calculer un produit médian $(n-1) \times (2n-2)$ et calculer à part les $2n-1$ produits manquants ;
- décomposer le produit médian en deux produits médians $\lceil n/2 \rceil \times 2\lceil n/2 \rceil$ et un produit médian $\lfloor n/2 \rfloor \times 2\lfloor n/2 \rfloor$ et calculer les $\lfloor n/2 \rfloor$ produits manquants.

Chacune de ces stratégies conduit à un algorithme de complexité $\Theta(n^{\log_2(3)})$, mais qui peut dépasser notablement celle de la multiplication de Karatsuba. En sélectionnant pour chaque valeur de n la stratégie donnant le moins de multiplications on obtient *expérimentalement* $R_{\mathrm{mul}}(n) \leqslant 1.25 K_{\mathrm{mul}}(n)$ lorsque $n_0 = 1$ et $R_{\mathrm{mul}}(n) \leqslant 1.02 K_{\mathrm{mul}}(n)$ lorsque $n_0 = 32$.



$y = R_{\mathrm{mul}}(2^x)/K_{\mathrm{mul}}(2^x)$



$y = R_{\mathrm{mul}}(2^x)/K_{\mathrm{mul}}(2^x)$

Produit médian de nombres entiers

L'application de l'algorithme RMP aux nombres entiers soulève le problème des retenues, comme dans le cas du produit tronqué de Mulders, mais cette fois lors de la phase de décomposition. En effet dans le cas de polynômes on a :

$$(A_1{:}A_2) + (A_2{:}A_3) = (A_1 + A_2) : (A_2 + A_3),$$

mais ceci n'est plus vrai pour des nombres en multiprécision, où A_1, A_2, A_3 représentent des blocs de chiffres, s'il y a une retenue lors du calcul de $A_1 + A_2$. Ceci fait que les relations sur $\alpha + \beta$, $\gamma + \delta$ et $\beta - \gamma$ présentées plus haut n'ont plus lieu. On peut résoudre le problème de la même manière que pour le produit tronqué en conservant les retenues au niveau de chaque chiffre, au prix d'une complexité spatiale $\Theta(n \ln n)$.

3 Division euclidienne

La division euclidienne est la deuxième opération fondamentale en calcul multiprécision après la multiplication. Elle est omniprésente dans les problèmes d'arithmétique modulaire, en particulier en cryptographie, et aussi dans les calculs en virgule flottante où la plupart des fonctions sont implémentées par application de la méthode de NEWTON ou par approximations rationnelles.

Les analyses de complexité pour les algorithmes de division supposent en général que la longueur du dividende est double de celle du diviseur à quelques unités près (division $2n$ par n). Si le dividende est plus court on peut le prolonger par des zéros ou court-circuiter certaines étapes de l'algorithme de division. S'il est plus long on peut le décomposer en tranches de longueur n et procéder à une division en cascade, chaque étape effectuant une division de deux tranches consécutives par le diviseur et remplaçant la tranche de poids faible par le reste de cette division.

Le problème de la division euclidienne de A par B (nombres ou polynômes) consiste à calculer le quotient Q et le reste R tels que $A = BQ + R$ et $R < B$ ou $\deg R < \deg B$. La *division sans reste* ne calcule que le quotient Q. Il n'existe pas à l'heure actuelle de division générale sans quotient où l'on ne calculerait que R (la méthode consistant à calculer Q et R puis à oublier Q n'est pas considérée comme une division sans quotient car elle n'économise aucun calcul).

3.1 La division selon KNUTH

Le théorème de la division euclidienne pour les polynômes est constructif et fournit un algorithme simple pour calculer le quotient Q et le reste R de la division de A de longueur p par B de degré n :

Divise$(A, B, p, n) =$
    $Q \longleftarrow 0$; $R \longleftarrow A$
    tant que $p \geqslant n$ *faire :*
        $q_{p-n} \longleftarrow r_{p-1}/b_{n-1}$; $R \longleftarrow R - q_{p-n}x^{p-n}B$; $p \longleftarrow p - 1$
    fin tant que

Le temps de calcul est $\Theta(n(p-n))$ soit $\Theta(n^2)$ pour une division « standard » $2n$ par n.

La division des nombres entiers enseignée à l'école suit le même principe à cette différence près que l'on prend la partie entière de r_{p-1}/b_{n-1} ou $(r_{p-1}x + r_{p-2})/b_{n-1}$ comme estimation initiale pour le chiffre suivant du quotient et que l'on corrige cette estimation au vu de la différence $R - q_{p-n}x^{p-n}B$. Par chance les enfants comptent en base 10 donc ils trouvent le bon chiffre en au plus 5 corrections (car le quotient estimé est au plus deux fois trop grand), ce n'est pas trop fastidieux. Pour une machine travaillant en base 2^{32} il peut y avoir jusqu'à 2^{31} corrections à effectuer, ce qui est plus embêtant. KNUTH donne dans [7] (algorithme D) une méthode efficace pour obtenir un quotient très proche du quotient exact :

1 multiplier A *et* B *par une puissance convenable de* 2 *de sorte que* $b_{n-1} \geqslant x/2$ *où* x *est la base de numération*
2 $q_{p-n} \longleftarrow \min\left(\left\lfloor \dfrac{r_{p-1}x + r_{p-2}}{b_{n-1}} \right\rfloor, x - 1\right)$
3 tant que $r_{p-1}x^2 + r_{p-2}x + r_{p-3} > q_{p-n}(b_{n-1}x + b_{n-2})$ *faire* $q_{p-n} \longleftarrow q_{p-n} - 1$.

On dit que B est *normalisé* lorsque $b_{n-1} \geqslant x/2$. Lorsque B n'est pas normalisé on le normalise en début de division par décalage, on applique le même décalage à A et on procède à la division des nombres décalés. Si $2^k A = (2^k B)Q + R$ avec $0 \leqslant R < 2^k B$ alors R est divisible par 2^k et l'on a $A = BQ + R/2^k$ avec $0 \leqslant R/2^k < B$ donc on reconstitue reste de la division initiale par décalage inverse de celui effectué lors de l'étape de normalisation, le quotient est inchangé. Le but de cette normalisation est d'assurer que tous les bits de b_{n-1} sont significatifs ; KNUTH démontre sous cette hypothèse que le quotient calculé en *2* n'est jamais trop petit et est au plus trop grand de deux unités. L'étape *3* effectue une première correction en basse précision et avec au plus deux itérations. A la fin de cette étape q_{p-n} est correct ou trop grand d'une unité, cette dernière éventualité ayant une probabilité inférieure à $2/x$ donc très faible en pratique. Lorsque $p - n$ est petit devant n il peut être avantageux de décaler uniquement les chiffres de poids fort de R et de B à chaque étape de la division, on économise ainsi le décalage complet de B et sa restauration ainsi que la gestion des conflits d'accès à B dans un programme multi-thread.

3.2 Division par inversion

Comme $a/b = a \times (1/b)$ on peut toujours ramener le calcul d'un quotient au calcul d'un inverse suivi d'une multiplication. Plus précisément, supposons que l'on veuille calculer le quotient Q et le reste R de la division euclidienne d'un polynôme A de longueur inférieure ou égale à $2n$ par un polynôme B de degré $n - 1$. On peut procéder comme suit :

- *effectuer la division sans reste de* x^{2n-1} *par* B *c'est-à-dire déterminer le polynôme* C *tel que* $\deg(x^{2n-1} - BC) < n - 1$ *;*
- *calculer* $Q = \mathscr{E}(AC/x^{2n-1})$ *puis* $R = A - BQ$ *où* $\mathscr{E}(F)$ *désigne la partie entière d'une fraction rationnelle* F.

En effet, $Ax^{2n-1} = B(AC) + A(x^{2n-1} - BC) = BQx^{2n-1} + B(AC - Qx^{2n-1}) + A(x^{2n-1} - BC)$ donc le polynôme $S = B(AC - Qx^{2n-1}) + A(x^{2n-1} - BC)$ est divisible par x^{2n-1} et son degré est inférieur ou égal à $3n - 3$ d'où $\deg(S/x^{2n-1}) < \deg(B)$. Le coût de cette méthode, hors calcul de C, est d'une multiplication $(n + 1) \times (n + 1)$ pour calculer la partie entière de AC/x^{2n-1} et d'une multiplication $n \times (n + 1)$ pour calculer BQ soit environ $2M_n$ en notant M_n le temps de calcul d'un produit $n \times n$.

La division euclidienne de nombres entiers peut être faite de la même manière en prenant pour x la base de numération et l'on obtient : $S/x^{2n-1} < 2x^n \leqslant 2xB$ donc il suffit de corriger le quotient et le reste trouvés par une « petite division ». Si l'on ne veut que le quotient Q alors dans le cas de la division de polynômes il suffit de ne pas calculer R, soit une économie de M_n. Dans le cas de la division de nombres entiers on peut calculer en temps constant quelques chiffres de poids fort de BQ ce qui permet en général de corriger Q sauf si le reste réel est trop petit.

Le calcul de C se ramène, en lisant les coefficients de B par degrés décroissants, au calcul des $n+1$ premiers coefficients de l'inverse d'une série formelle de valuation nulle. On note $B = b_0 x^{n-1} + \ldots + b_{n-1}x^0$, $\widetilde{B} = b_0 + b_1 y + \ldots + b_{n-1}y^{n-1} + 0y^n + \ldots$ la série associée et on calcule $\widetilde{B}^{-1}$ par application de la méthode de NEWTON à l'équation :

$$f(U) = \frac{1}{U} - \widetilde{B} = 0$$

c'est-à-dire que l'on calcule les termes successifs de la suite récurrente :

$$U_0 = \frac{1}{b_0}, \qquad U_{k+1} \equiv U_k - \frac{f(U_k)}{f'(U_k)} \equiv U_k + U_k(1 - \widetilde{B}U_k) \pmod{y^{2^{k+1}}}.$$

On a par récurrence $\mathrm{val}(1 - \widetilde{B}U_k) \geqslant 2^k$ donc on obtient $\widetilde{C}$ puis C en au plus $\lceil \log_2(n + 1) \rceil$ itérations. Le temps de calcul de U_{k+1} à partir de U_k est approximativement égal à $3M_{2^k}$ (deux multiplications $2^k \times 2^k$ pour calculer $\widetilde{B}U_k \bmod y^{2^{k+1}}$ et une seule multiplication $2^k \times 2^k$ pour multiplier le résultat par U_k car $\mathrm{val}(1 - \widetilde{B}U_k) \geqslant 2^k$). Le temps de calcul de C par cette méthode est :

$$3(M_{2^{p-1}} + M_{2^{p-2}} + M_{2^{p-3}} + \ldots) \approx KM_{2^p}$$

avec $p = \lceil \log_2(n + 1) \rceil$ et $K = \frac{3}{2}$ pour la multiplication de Karatsuba, $K = 3$ pour la multiplication par FFT. Le temps total de calcul de Q et R est donc de l'ordre de $\frac{3}{2}M_{2^p} + 2M_n$, quantité comprise entre $\frac{7}{2}M_n$ et $\frac{13}{2}M_n$ si l'on utilise la multiplication de KARATSUBA. Pour la multiplication par FFT on obtient un temps $3M_{2^p} + 2M_n$, soit entre $5M_n$ et $8M_n$. De plus, si l'on doit effectuer plusieurs divisions avec le même diviseur B, alors le coût marginal de chaque division est $2M_n$. Cet algorithme de division rapide est décrit (en version nombres entiers et non polynômes) par KNUTH dans [7] qui en attribue l'idée à COOK. Il date probablement du milieu des années 60. Il semble qu'ainsi la messe est dite : *une division équivaut à quelques multiplications.* Pourtant de nouveaux algorithmes vont apparaître au cours des années 90 améliorant le coefficient « quelques ».

En 1993 KARP et MARKSTEIN proposent dans [6] de modifier la dernière étape de calcul de C en introduisant le dividende A ou plus exactement la série formelle associée, $\widetilde{A}$. Soit U_{p-1} l'approximation de $1/\widetilde{B}$ obtenue à l'avant-dernière étape de l'algorithme d'inversion. On calcule :

$$V_{p-1} \equiv \widetilde{A}U_{p-1} \pmod{y^{2^{p-1}}} \qquad \text{et} \qquad V_p \equiv V_{p-1} + U_{p-1}(\widetilde{A} - \widetilde{B}V_{p-1}) \pmod{y^{n+1}}.$$

On a donc :

$$\begin{aligned} &\widetilde{A} - \widetilde{B}V_{p-1} \equiv \widetilde{A}(1 - \widetilde{B}U_{p-1}) && \pmod{y^{2^{p-1}}}\ ; && \mathrm{val}(\widetilde{A} - \widetilde{B}V_{p-1}) \geqslant 2^{p-1}\ ; \\ &\widetilde{A} - \widetilde{B}V_p \equiv (\widetilde{A} - \widetilde{B}V_{p-1})(1 - \widetilde{B}U_{p-1}) && \pmod{y^{n+1}}\ ; && \mathrm{val}(\widetilde{A} - \widetilde{B}V_p) \geqslant n+1. \end{aligned}$$

en particulier les $n+1$ premiers coefficients de V_p sont ceux de Q. On échange ainsi le calcul du produit AC de coût M_n contre le calcul de $\widetilde{A}U_{p-1}$ de coût $M_{2^{p-1}}$. Si l'on utilise la multiplication de KARATSUBA et si $n+1 = 2^p$, le temps de calcul de la division sans reste de A par B passe ainsi de $\frac{5}{2}M_n$ à $\frac{11}{6}M_n$ soit une économie de l'ordre de 27%.

En 1997 JEBELEAN conteste dans [3] l'efficacité de la méthode par inversion en prétendant qu'une division par cette méthode est environ 30 fois plus lente qu'une multiplication ; il fait référence à une analyse de lui-même et de KRANDICK publiée en 1996, mais la version électronique de l'article cité en référence, disponible sur la page personnelle de KRANDICK, ne contient aucune analyse sur ce sujet ???

En 1999 HANROT et ZIMMERMANN inventent l'algorithme RMP décrit au chapitre précédent et l'utilisent pour calculer $1 - \widetilde{B}U_k$ en temps M_{2^k} au lieu de $2M_{2^k}$ (avec la multiplication de KARATSUBA). Le temps de calcul de U_{p-1} est donc réduit d'un tiers et le temps de calcul de la division sans reste passe à $\frac{4}{3}M_n$. Ils remarquent aussi que les produits $U_k(1 - \widetilde{B}U_k)$, $\widetilde{A}U_{p-1}$ et $U_{p-1}(\widetilde{A} - \widetilde{B}V_{p-1})$ peuvent être calculés avec l'algorithme de multiplication tronquée de MULDERS, soit une économie (contestée depuis) de 19% sur ces produits et un temps de division sans reste de l'ordre de $1.17M_n$.

3.3 La division selon BURNIKEL et ZIEGLER

Curieusement, il aura fallu attendre jusqu'en 1998 pour découvrir une méthode récursive de division rapide dont l'idée était pourtant en germe dans l'algorithme D de KNUTH. On rappelle que KNUTH résolvait le problème de « deviner » le prochain chiffre du quotient par l'algorithme :

1 multiplier A *et* B *par une puissance convenable de* 2 *de sorte que* $b_{n-1} \geqslant x/2$ *où* x *est la base de numération*
2 $q_{p-n} \longleftarrow \min\left(\left\lfloor \frac{r_{p-1}x + r_{p-2}}{b_{n-1}} \right\rfloor, x-1\right)$
3 tant que $r_{p-1}x^2 + r_{p-2}x + r_{p-3} > q_{p-n}(b_{n-1}x + b_{n-2})$ *faire* $q_{p-n} \longleftarrow q_{p-n} - 1$.

c'est-à-dire que KNUTH effectue sans le dire une division du nombre triple précision $r_{p-1}x^2 + r_{p-2}x + r_{p-3}$ par le nombre double précision $b_{n-1}x + b_{n-2}$ au moyen d'une division double précision par simple précision et zéro, une ou deux corrections. Cette idée sera reprise par BURNIKEL et ZIEGLER dans [2] en remplaçant la notion de chiffre par celle de bloc de chiffres :

Divise_21$(A, B, n) = (\text{longueur}(A) = 2n,\ \text{longueur}(B) = n)$
  si $n \leqslant n_0$ *effectuer la division de manière classique*
  sinon :
    décomposer A *en blocs de longueur* $n/2$: $A = A_3{:}A_2{:}A_1{:}A_0$, A_3 *étant le bloc de poids fort*
    décomposer B *en blocs de longueur* $n/2$: $B = B_1{:}B_0$, B_1 *étant le bloc de poids fort*

$(Q_1, R_2{:}R_1) \longleftarrow$ *Divise_32* $(A_3{:}A_2{:}A_1,\ B_1{:}B_0,\ n/2)$
$(Q_0, R_1{:}R_0) \longleftarrow$ *Divise_32* $(\ R_2{:}R_1{:}A_0,\ B_1{:}B_0,\ n/2)$
retourner $Q = Q_1{:}Q_0$ *et* $R = R_1{:}R_0$
fin sinon

Divise_32 $(A, B, n) =$ (longueur$(A) = 3n$, longueur$(B) = 2n$)
si $n \leqslant n_0$ *effectuer la division de manière classique*
sinon :
décomposer A *en blocs de longueur* n : $A = A_2{:}A_1{:}A_0$, A_2 *étant le bloc de poids fort*
décomposer B *en blocs de longueur* n : $B = B_1{:}B_0$, B_1 *étant le bloc de poids fort*
si $A_2 < B_1$ *alors* $(Q, R) \longleftarrow$ *Divise_21* $(A_2{:}A_1,\ B_1,\ n)$
sinon $(Q, R) \longleftarrow (x^n - 1,\ A_2{:}A_1 - (x^n - 1)B_1)$
$R \longleftarrow R{:}A_0 - Q \times B_0$
tant que $R < 0$ *faire* : $Q \longleftarrow Q - 1$; $R \longleftarrow R + B$
retourner Q *et* R
fin sinon

Burnikel et Ziegler ont tout simplement remplacé la base de numération x par x^n, il suffisait d'y penser ! La démonstration de correction de Knuth reste donc valable et le nombre d'étapes de correction dans *Divise_32* est majoré par 2 sous l'hypothèse $B_1 \geqslant x^n/2$. La division avec reste d'un nombre de longueur $2n$ par un nombre de longueur n se résume donc à deux divisions avec reste n par $n/2$ et deux multiplications $n/2 \times n/2$ plus quelques opérations de complexité négligeable, soit un temps de calcul :

$$D_n = 2M_{n/2} + 4M_{n/4} + 8M_{n/8} + \ldots \approx 2M_n$$

en prenant la multiplication de Karatsuba. Avec la multiplication par FFT le temps de division est de l'ordre de $\frac{1}{2}\log_2(n)M_n$ donc moins bon que celui de la division par inversion. Notons que l'algorithme de Burnikel et Ziegler s'applique aussi à la division de polynômes en remplaçant *Divise_32* par :

Divise_32 $(A, B, n) =$ (longueur$(A) = 3n$, longueur$(B) = 2n$)
si $n \leqslant n_0$ *effectuer la division de manière classique*
sinon :
décomposer A *en blocs de longueur* n : $A = A_2{:}A_1{:}A_0$, A_2 *étant le bloc de plus haut degré*
décomposer B *en blocs de longueur* n : $B = B_1{:}B_0$, B_1 *étant le bloc de plus haut degré*
$(Q, R) \longleftarrow$ *Divise_21* $(A_2{:}A_1,\ B_1,\ n)$
$R \longleftarrow R{:}A_0 - Q \times B_0$
retourner Q *et* R
fin sinon

Division sans reste

Dans *Divise_21* il est calculé deux restes : le reste interne $R_2{:}R_1$ permettant de calculer Q_0 et le reste final, $R_1{:}R_0$. Si l'on veut effectuer une division sans reste il suffit d'omettre le calcul de ce reste final :

Divise_21_sr $(A, B, n) =$ (longueur$(A) = 2n$, longueur$(B) = n$)
si $n \leqslant n_0$ *effectuer la division de manière classique*
sinon :
décomposer A *en blocs de longueur* $n/2$: $A = A_3{:}A_2{:}A_1{:}A_0$, A_3 *étant le bloc de poids fort*
décomposer B *en blocs de longueur* $n/2$: $B = B_1{:}B_0$, B_1 *étant le bloc de poids fort*
$(Q_1, R_2{:}R_1) \longleftarrow$ *Divise_32* $(A_3{:}A_2{:}A_1,\ B_1{:}B_0,\ n/2)$
$Q_0 \longleftarrow$ *Divise_32_sr* $(R_2{:}R_1{:}A_0,\ B_1{:}B_0,\ n/2)$
retourner $Q = Q_1{:}Q_0$
fin sinon

Divise_32_sr $(A, B, n) =$ (longueur$(A) = 3n$, longueur$(B) = 2n$)
si $n \leqslant n_0$ *effectuer la division de manière classique*
sinon :
décomposer A *en blocs de longueur* n : $A = A_2{:}A_1{:}A_0$, A_2 *étant le bloc de poids fort*
décomposer B *en blocs de longueur* n : $B = B_1{:}B_0$, B_1 *étant le bloc de poids fort*

$Q \longleftarrow$ *Divise_21_sr*$(A_2{:}A_1,\ B_1,\ n)$
retourner Q
fin sinon

Pour les polynômes le résultat est exact. Pour les nombres entiers chaque quotient retourné par *Divise_32_sr* est trop grand d'au plus deux unités, donc compte tenu des appels récursifs le quotient final est trop grand d'au plus $2\log_2(n)$ unités. Burnikel et Ziegler démontrent qu'en calculant les quatre coefficients du milieu dans le produit BQ on peut déterminer un encadrement de l'erreur sur le quotient, les deux bornes de cet encadrement étant égales dans la plupart des cas (la probabilité d'avoir des bornes inégales sur une machine 32 bits est majorée par 2^{-23}). Le calcul de ces quatre termes a une complexité linéaire en n donc négligeable dans le temps total de division.

Ainsi, pour une division sans reste de polynômes ou pour une division sans reste d'entiers dans le cas favorable, la méthode de Burnikel et Ziegler économise un temps de calcul de l'ordre de :

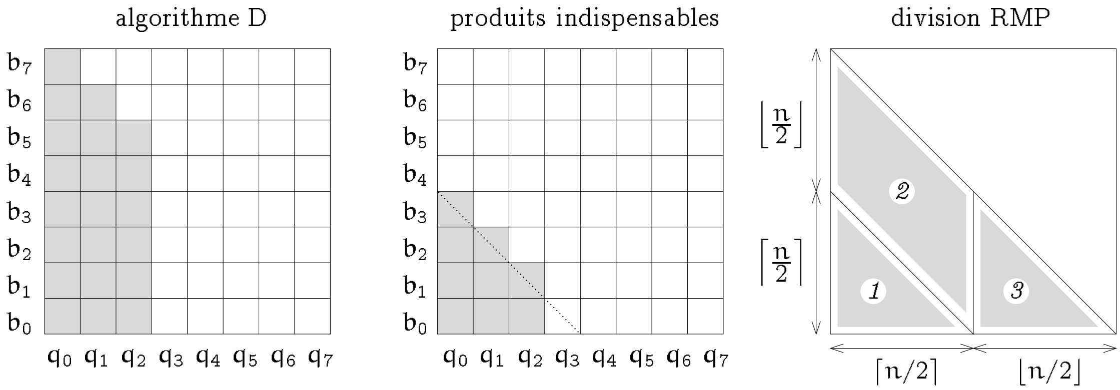
$$M_{n/2} + M_{n/4} + M_{n/8} + \ldots \approx \frac{1}{2} M_n$$

donc fournit le quotient en temps $\frac{3}{2}M_n$. Dans le cas défavorable pour les nombres entiers il faut calculer le reste $A - BQ$ pour déterminer l'erreur sur Q et le temps total de calcul monte à $\frac{5}{2}M_n$. Michel Quercia décrit dans [9] un algorithme de division rapide itérative construit sur une variante de la multiplication détendue de Joris Van der Hoeven qui se trouve, après examen comparatif, n'être autre qu'une version dérécursifiée de l'algorithme de Burnikel et Ziegler. L'intérêt de cet algorithme, outre qu'il est itératif, est que la stratégie d'évitement du calcul du reste est plus simple : on arrête les calculs dès que l'on est sûr de la correction du quotient, compte tenu d'une évaluation pessimiste des termes non encore calculés. Avec cet algorithme le temps de calcul d'une division sans reste est $\frac{3}{2}M_n$ dans le cas favorable et $2M_n$ dans le cas défavorable, c'est-à-dire qu'il n'y a pas de pénalité par rapport à la division avec reste.

Notons pour conclure ce paragraphe que Krandick et Jebelean avaient décrit dans [3] un an avant Burnikel et Ziegler un autre algorithme de division récursive avec reste en temps $2M_n$ dans la plupart des cas mais $\Theta(n^2)$ dans les rares cas défavorables. Par ailleurs, Krandick et Jebelean sont aussi les inventeurs d'une méthode de division amusante dans le cas où l'on sait a priori que le reste est nul *(division exacte)* : la division bidirectionelle décrite dans [4]. Cette méthode consiste à calculer la moitié haute du quotient par division classique suivant les puissances décroissantes de la base et la moitié basse de façon moins classique par division suivant les puissances croissantes de cette base, pourvu que le chiffre des unités du diviseur soit premier à la base, ce qui est facile à réaliser lorsque la base est une puissance de 2. Ces deux divisions peuvent être effectuées en parallèle sur une machine biprocesseur et il est en général possible d'éviter de calculer les deux restes. Inventée en 1994, cette méthode utilisait un algorithme de division de complexité $\Theta(n^2)$ mais elle pourrait être ressortie des cartons et adaptée à la division de Burnikel et Ziegler pour fournir un algorithme de division exacte en temps $2 \times \frac{3}{2}M_{n/2} = M_n$ ou aussi à la division RMP qui suit, en temps $2 \times M_{n/2} = \frac{2}{3}M_n$. Enfin, il est possible d'effectuer une division avec reste en temps M_n à l'aide de la multiplication détendue de Van der Hoeven décrite au chapitre précédent, mais cette méthode nécessite une mémoire auxiliaire $\Theta(n \ln n)$ alors que toutes les autres méthodes de division décrites ici tournent en mémoire $O(n)$.

3.4 La division par RMP

C'est en Janvier 2000 qu'est atteint pour la première fois le seuil psychologique de $1.00000M_n$ pour calculer les n premiers coefficients du quotient de deux séries formelles avec une mémoire auxiliaire $O(n)$, grâce à une utilisation inattendue de l'algorithme RMP ([12]). L'idée de départ est d'effectuer une multiplication détendue du diviseur B par le quotient Q tout en retranchant au fur et à mesure le produit calculé au dividende A pour découvrir les coefficients suivants du quotient.



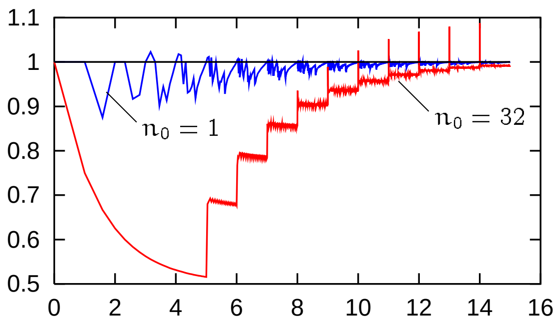
La figure ci-dessus illustre les principes de l'algorithme D de Knuth et de l'algorithme de division par RMP. Soient $A = \sum_{i=0}^{\infty} a_i x^i$, $B = \sum_{i=0}^{\infty} b_i x^i$ deux séries formelles avec $b_0 \neq 0$ et $Q = \sum_{i=0}^{\infty} q_i x^i$ le quotient de A par B. L'algorithme D calcule les n premiers coefficients de Q coefficient par coefficient en retranchant du dividende tous les produits $q_i b_j$ dès que q_i est déterminé, j étant limité à la précision finale du calcul. Au moment du calcul de q_3 on a calculé tous les produits $q_i b_j$ avec $i < 3$ et $i + j < n$ figurant dans les trois premières colonnes du premier tableau. Or, seuls les produits $q_i b_j$ tels que $i+j \leqslant 3$ sont nécessaires au calcul de q_3, c'est la zone grisée du deuxième tableau. Les autres produits $q_i b_j$ avec $i < 3$ n'interviendront que lors du calcul des coefficients suivants de Q ; leur calcul peut être différé pour être effectué plus rapidement en bloc. C'est le principe de l'algorithme de division par RMP :

Divise_rmp $(A, B, n) =$ (*calcule les* n *premiers coefficients de* A/B)
  si $n \leqslant n_0$ *calculer* $q_0 \ldots q_{n-1}$ *de façon classique*
  sinon :
    soit $p = \lceil n/2 \rceil$
    1 $q_0 \ldots q_{p-1} \longleftarrow$ *Divise_rmp* (A, B, p)
    2 $a_p \ldots a_{n-1} \longleftarrow a_p \ldots a_{n-1} -$ *RMP* $(b_1 \ldots b_{n-1},\ q_0 \ldots q_{p-1},\ n-p)$
    3 $q_p \ldots q_{n-1} \longleftarrow$ *Divise_rmp* $(a_p \ldots a_{n-1} \ldots,\ B, n-p)$
  fin sinon

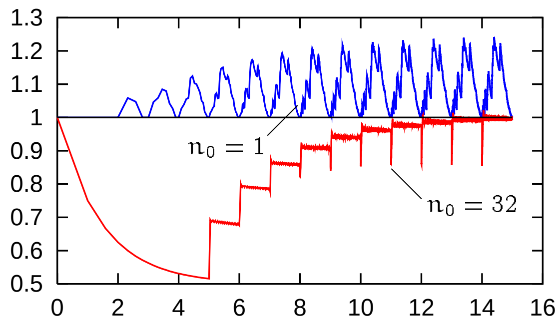
Le nombre de multiplications scalaires pour calculer les n premiers coefficients de A/B vérifie l'équation de récurrence :

$$D^R_{\mathrm{mul}}(n) = D^R_{\mathrm{mul}}(\lceil n/2 \rceil) + D^R_{\mathrm{mul}}(\lfloor n/2 \rfloor) + R_{\mathrm{mul}}(\lceil n/2 \rceil),$$

avec $D^R_{\mathrm{mul}}(n) = n(n+1)/2$ si $n \leqslant n_0$. Pour $n_0 = 1$ on obtient ainsi $D^R_{\mathrm{mul}}(n) = R_{\mathrm{mul}}(n) = K_{\mathrm{mul}}(n)$ lorsque n est une puissance de 2 ; la situation est plus chaotique pour n quelconque :



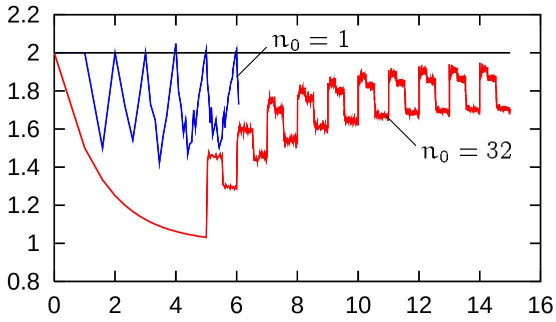
$y = D^R_{\mathrm{mul}}(2^x)/R_{\mathrm{mul}}(2^x)$



$y = D^R_{\mathrm{mul}}(2^x)/K_{\mathrm{mul}}(2^x)$

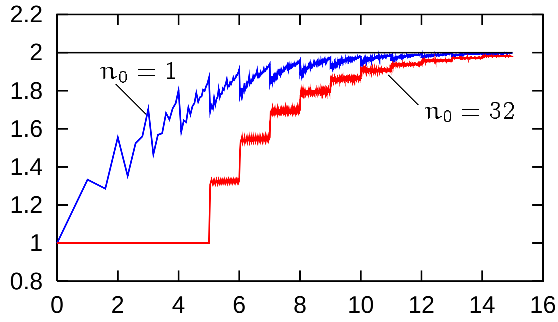
L'algorithme de division par RMP peut théoriquement servir aussi pour effectuer une division de polynômes avec reste : il suffit, lorsqu'on a terminé le calcul de Q, de calculer le produit tronqué au dessus de la diagonale avec l'algorithme de Mulders et de le retrancher au dividende, pour une complexité totale de l'ordre de $1.8 M_n$. Toutefois, compte tenu du caractère aussi chaotique du temps de calcul pour la multiplication tronquée de Mulders, il est probable que cette méthode n'est pas compétitive en comparaison de l'algorithme plus simple de Burnikel et Ziegler dont la complexité théorique, $2M_n$ est confirmée expérimentalement.

division RMP-MULDERS



$$y = \frac{D^R_{mul}(2^x) + Muld_{mul}(2^x)}{K_{mul}(2^x)}$$

division avec reste de BURNIKEL et ZIEGLER



$$y = D_{mul}(2^x)/K_{mul}(2^x)$$

4 Racine carrée

4.1 La racine carrée à l'école

Le soir au coin du feu, les anciens nous racontent qu'autrefois, il y a très longtemps, on apprenait à calculer les racines carrées à l'école primaire. La méthode ressemble à celle de la division :

on écrit le nombre dont on veut la racine à la place du dividende en ajoutant un zéro à gauche si sa longueur est impaire et on effectue la division par tranches de deux chiffres en plaçant chaque nouveau chiffre à la fois à droite du quotient et aussi à droite du diviseur. Lorsqu'on a terminé une soustraction il faut doubler le chiffre ajouté au diviseur en faisant bien attention aux retenues ...

```
03, 00 00 00 |          03, 00 00 00 | 1,        03, 00 00 00 | 2, 7      03, 00 00 00 | 3, 4 3
             |________   1,          |______     1,           |______     1,           |________
                        ___          | 1,       ___           | 1, 7     ___           | 1, 7 3
                         2, 00       |           2, 00        |           2, 00        |
                                                 1, 89        |           1, 89        |
                                                ______        |          ______        |
                                                   11 00      |             11 00      |
                                                                            10 29      |
                                                                           ______      |
                                                                              71 00    |
```

```
03, 00 00 00 | 3, 4 6 2
 1,          |__________
___          | 1, 7 3 2
 2, 00       |
 1, 89       |
______       |
    11 00    |
    10 29    |
   _______   |
       71 00 |
       69 24 |
      ______ |
        1 76 |
```

$$\boxed{03,000000 = 1,732^2 + 0,000176}$$

Il est facile de se convaincre que cet algorithme est valide (c'est la maîtresse qui l'a dit) et que sa complexité pour calculer une racine carrée avec n chiffres est $\Theta(n^2)$ (ça elle ne l'avait pas dit !).

En 1999, PAUL ZIMMERMANN présente dans [11] une version « diviser pour régner » de cet algorithme en considérant un nombre de $2n$ chiffres comme constitué de quatre blocs de $n/2$ chiffres et en faisant jouer aux blocs le rôle des chiffres dans l'algorithme scolaire :

Sqrt_rem $(A, n) =$ (longueur$(A) = 2n$, *calcule* $B = \lfloor\sqrt{A}\rfloor$ *et* $C = A - B^2$)
  si $n \leqslant n_0$ *calculer la racine carrée de manière classique*
  sinon :
    décomposer A *en blocs de longueur* $n/2$: $A = A_3{:}A_2{:}A_1{:}A_0$, A_3 *étant le bloc de poids fort*

1 $(B_1, C_2) \longleftarrow$ *Sqrt_rem* $(A_3 \colon A_2,\ n/2)$
2 $(B_0, C_1) \longleftarrow$ *Div_rem* $(C_2 \colon A_1,\ 2B_1)$
3 $B \longleftarrow B_1 \colon B_0$; $C \longleftarrow C_1 \colon A_0 - B_0^2$
4 *tant que* $C < 0$ *faire :* $C \longleftarrow C + 2B - 1$; $B \longleftarrow B - 1$ *fin tant que*
5 *retourner* B *et* C

fin sinon

Div_rem (X, Y) effectue la division euclidienne de X par Y et retourne le couple (Q, R). B_0 joue le rôle du chiffre que l'on ajoute à droite du quotient et du diviseur ; les étapes *2,3,4* déterminent la valeur correcte de B_0 et effectuent la division du reste en cours, $C_2 \colon A_1 \colon A_0$ par $(2B_1) \colon B_0$. Paul Zimmermann démontre que, sous l'hypothèse $A_3 \geqslant \frac{1}{4}x^{n/2}$, la valeur de B_0 calculée à l'étape *2* est soit exacte, soit trop grande d'une unité et donc il y a au plus une itération dans la boucle de correction *4*. L'hypothèse $A_3 \geqslant \frac{1}{4}x^{n/2}$ peut être réalisée en procédant à une normalisation préalable de A par décalage d'un nombre pair de bits.

Si $S_{mul}(n)$ désigne le nombre de multiplications entre chiffres effectuées pour calculer la racine carrée entière B et le reste C pour un nombre A de n chiffres on a l'équation de récurrence :

$$S_{mul}(n) = S_{mul}(\lceil n/2 \rceil) + D_{mul}(\lceil n/2 \rceil) + M_{mul}(\lfloor n/2 \rfloor),$$

et $S_{mul}(n) = n(n+1)/2$ si $n \leqslant n_0$. En prenant la division avec reste de Burnikel et Ziegler et la multiplication de Karatsuba on obtient : $S_{mul}(n) \approx \frac{3}{2}K_{mul}(n)$; en prenant la division par inversion et la multiplication par FFT on obtient : $S_{mul}(n) \approx 6M_{mul}(n)$.

Racine carrée sans reste

Dans les problèmes impliquant le calcul d'une racine carrée on recherche en général l'une des trois informations suivantes :

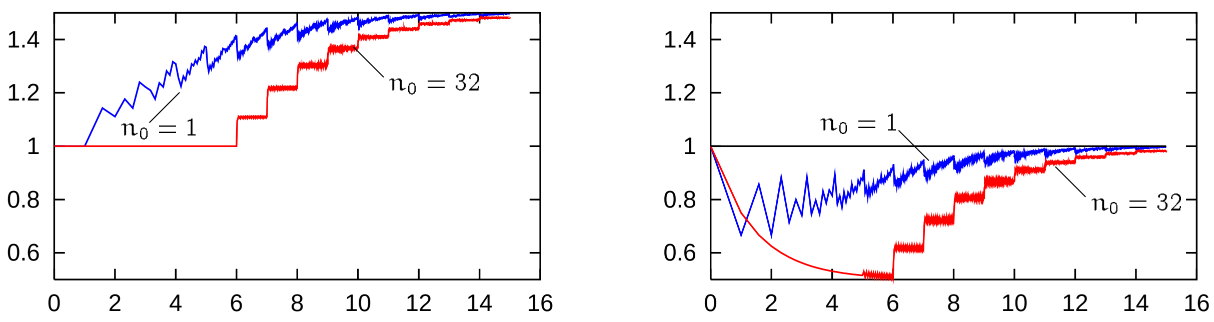
- l'arrondi par défaut, $B = \lfloor \sqrt{A} \rfloor$;
- l'arrondi au plus proche, $B' = \lfloor \sqrt{A} \rceil = \begin{cases} B & \text{si } C \leqslant 2B \text{ ;} \\ B+1 & \text{si } C > 2B \text{ ;} \end{cases}$
- est-ce que A est un carré parfait et si oui lequel ?

Dans les deux premiers cas le calcul exact de C n'est en général pas nécessaire, il suffit de connaître quelques chiffres de poids fort de C pour savoir si l'on doit effectuer une correction en *4* ou si l'on doit incrémenter B pour obtenir l'arrondi au plus proche. Donc, sauf si A est très proche du carré d'un entier ou du carré d'un demi-entier, on peut éviter les calculs de C_1 et B_0^2 dans l'appel de *Sqrt_rem* de niveau 0 et les remplacer par un calcul basse précision. Si ce calcul basse précision est concluant on a ainsi économisé $\frac{1}{2}K_{mul}(\lceil n/2 \rceil) + K_{mul}(\lfloor n/2 \rfloor) \approx \frac{1}{2}K_{mul}(n)$ multiplications entre chiffres (avec le modèle Karatsuba-Burnikel-Ziegler). S'il n'est pas concluant alors il faut calculer C_1 et B_0 après coup, pour un coût $\frac{2}{3}K_{mul}(n)$. Le temps de calcul d'une racine sans reste est donc :

$$K_{mul}(n) \quad \text{dans les cas favorables ;} \qquad \frac{5}{3}K_{mul}(n) \quad \text{dans les cas défavorables.}$$

Michel Quercia présente dans [9] une version itérative de l'algorithme *Sqrt_rem* pour laquelle l'évitement du calcul du reste se réduit à arrêter la boucle dès que possible, donc sans pénalité dans les cas défavorables.

racine carrée avec reste — racine carrée sans reste



$y = S_{mul}(2^x)/K_{mul}(2^x)$ — $y = S_{mul}(2^x)/K_{mul}(2^x)$

4.2 Les méthodes par approximations successives

Il y a au moins deux méthodes pour calculer la racine carrée d'un nombre A par approximations successives, toutes deux dérivées de la méthode de NEWTON :

– résoudre l'équation : $f(U) = U^2 - A = 0$, ce qui donne la relation de récurrence :

$$U_{k+1} = \frac{1}{2}\Big(U_k + \frac{A}{U_k}\Big) = U_k + \frac{A - U_k^2}{2U_k} \ ;$$

– calculer $1/\sqrt{A}$ à l'aide de l'équation : $g(V) = 1/V^2 - A = 0$, ce qui donne la relation de récurrence :

$$V_{k+1} = V_k + \frac{1}{2}V_k(1 - AV_k^2)$$

puis calculer $\sqrt{A} = A \times (1/\sqrt{A})$.

La première méthode est connue sous le nom de méthode de HERON. La forme $U_k + (A - U_k^2)/(2U_k)$ est en général plus commode pour les calculs car la division de $A - U_k^2$ par $2U_k$ est plus facile à réaliser que la division de A par U_k lorsque U_k est proche de $\sqrt{A}$. Pour A entier, si l'on utilise cette méthode en effectuant des divisions entières alors la suite $(U_k)_{k\geqslant 1}$ est d'abord strictement décroissante puis oscillante ou stationnaire donc on dispose d'un critère d'arrêt simple :

si k *est le premier indice non nul tel que* $U_k \leqslant U_{k+1}$ *alors* $U_k = \lfloor\sqrt{A}\rfloor$.

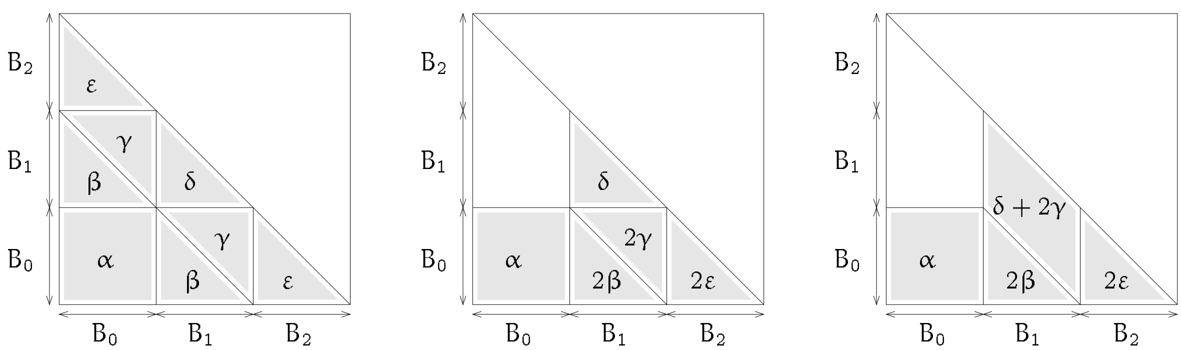
La deuxième méthode était jusqu'à présent plus intéressante d'un point de vue temps calcul car elle ne nécessite pas de divisions, mais maintenant que l'on sait faire une division aussi vite qu'une multiplication elle pourrait perdre de son attrait.

Les deux méthodes sont aussi applicables pour calculer la racine carrée d'une série formelle de valuation nulle et dont le coefficient constant est un carré. Comme pour la méthode de division par inversion, on montre que les valuations de $A - U_k^2$ et $1 - AV_k^2$ sont doublées à chaque étape donc il suffit de $\lceil \log_2(n+1)\rceil$ itérations pour obtenir les n premiers coefficients de $\sqrt{A}$.

Le temps de calcul, avec le modèle de complexité de KARATSUBA et en utilisant tous les trucs disponibles (produit tronqué, produit médian, division RMP, accélération de KARP et MARKSTEIN), est de l'ordre de $0.91M_n$ pour la méthode de HERON et $1.41M_n$ pour la deuxième méthode. Le calcul de la racine carrée sans reste par la méthode de HERON est donc *théoriquement* légèrement plus rapide que la méthode récursive de ZIMMERMANN, mais le record sera bientôt battu ...

4.3 La racine carrée par RMP

Cette dernière méthode, présentée dans [12], est construite sur une multiplication détendue et tronquée de B par lui-même, $B = B_0 + B_1x^n + B_2x^{2n}$ étant un polynôme de longueur $3n$:



$$\begin{aligned}
\operatorname{trunc}(B^2) &= \operatorname{trunc}\left\{(B_0 + B_1x^n + B_2x^{2n})^2\right\} \\
&= \alpha + 2\beta x^n + (2\gamma + \delta + 2\varepsilon)x^{2n} \\
&= B_0^2 + \operatorname{trunc}(2B_0B_1)x^n + \operatorname{rmp}\left\{(2B_0 + B_1x^n) \times B_1\right\}x^{2n} + \operatorname{trunc}(2B_0B_2)x^{2n} \\
&= B_0^2 + \operatorname{trunc}\left\{(2B_0 + B_1x^n) \times (B_1 + B_2x^n)\right\}x^n.
\end{aligned}$$

trunc et rmp désignant les produits tronqué et médian. Soit alors $A = A_0 + A_1x^n + A_2x^{2n} + \dots$ une série formelle de valuation nulle dont le terme constant est un carré et $B = \sqrt{A}$. D'après le calcul précédent on a :

$$\begin{aligned}
B_0 &\equiv \sqrt{A_0} \pmod{x^n}\ ; \\
B_1 + B_2x^n &\equiv \frac{(A - B_0^2)/x^n}{2B_0 + B_1x^n} \pmod{x^{2n}}.
\end{aligned}$$

Ces deux relations constituent presque un algorithme de calcul de B :

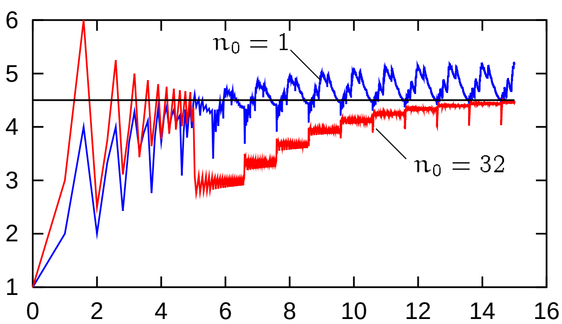
- calculer B_0 et $A - B_0^2$ à l'aide de *Sqrt_rem* ;
- calculer $B_1 + B_2x^n$ par division sans reste de $(A - B_0^2)/x^n$ par $2B_0 + B_1x^n$.

Presque car B_1 figure dans les deux membres de la division. Mais en utilisant la division par RMP on peut commencer la division puisque l'on connaît la partie basse, $2B_0$, du diviseur et lorsque la première étape de la division est achevée on a déterminé B_1 donc on peut continuer la division. La division sans reste de Burnikel et Ziegler a aussi cette propriété de calculer la première moitié du quotient avant d'utiliser la deuxième moitié du diviseur, on pourrait l'employer mais il se trouve qu'elle est plus lente que la division par RMP.

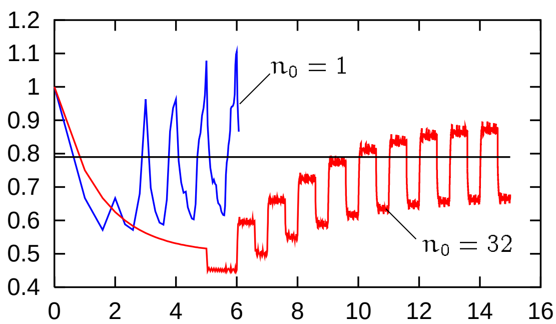
En changeant n en $n/3$ on obtient l'équation suivante pour le nombre $S^R_{\mathrm{mul}}(n)$ de multiplications nécessaires pour calculer les n premiers coefficients de $\sqrt{A}$ par la méthode RMP :

$$\begin{aligned}
S^R_{\mathrm{mul}}(n) &= S_{\mathrm{mul}}(\lceil n/3\rceil) + D^R_{\mathrm{mul}}(n - \lceil n/3\rceil) \\
&\approx \frac{3}{2}K_{\mathrm{mul}}(\lceil n/3\rceil) + K_{\mathrm{mul}}(n - \lceil n/3\rceil) \\
&\approx \frac{9}{2}K_{\mathrm{mul}}(\lceil n/3\rceil).
\end{aligned}$$

En considérant que $K_{\mathrm{mul}}(\lceil n/3\rceil) \approx K_{\mathrm{mul}}(n)/3^{\log_2(3)}$ on obtient $S^R_{\mathrm{mul}}(n) \approx 0.79K_{\mathrm{mul}}(n)$, c'est à l'heure actuelle le record de vitesse au niveau mondial pour le calcul d'une racine carrée. Mais en réalité la relation $K_{\mathrm{mul}}(\lceil n/3\rceil) \approx K_{\mathrm{mul}}(n)/3^{\log_2(3)}$ n'est qu'indicative, et comme dans le cas du produit tronqué de Mulders on constate des écarts importants par rapport à cette loi :



$y = S^R_{\mathrm{mul}}(2^x)/K_{\mathrm{mul}}(\lceil 2^x/3\rceil)$



$y = S^R_{\mathrm{mul}}(2^x)/K_{\mathrm{mul}}(2^x)$

5 Bibliographie

[1] DANIEL J. BERNSTEIN
Multidigit multiplication for mathematicians
`http://pobox.com/~djb/papers/m3.dvi`

[2] CHRISTOPH BURNIKEL, JOACHIM ZIEGLER
Fast recursive division
rapport de recherche MPI-I-98-1-022, Max Planck Institut Für Informatik, octobre 1998

[3] TUDOR JEBELEAN
Practical integer division with Karatsuba complexity
`http://www.risc.uni-linz.ac.at/people/tjebelea/shortcv.html#selected-papers`

[4] WERNER KRANDICK, TUDOR JEBELEAN
Bidirectionnal exact integer division
`http://math-www.uni-paderborn.de/~krandick/abstracts/pasco94.html`

[5] A. KARATSUBA, J. OFMAN
Multiplication of multidigit numbers on automata
Soviet Physics Doklady, 7:595-596, 1963

[6] ALAN H. KARP, PETER MARKSTEIN
High precision division and square root
`http://www.hpl.hp.com/techreports/93/HPL-93-42.html`

[7] DONALD E. KNUTH
The art of computer programming, volume 2
Addison-Wesley

[8] THOM MULDERS
On short multiplications and divisions
`http://www.inf.ethz.ch/~mulders/papers/Shortprodquo`

[9] MICHEL QUERCIA
Division et racine carrée rapides
`http://pauillac.inria.fr/~quercia/divrap.ps.gz`

[10] JORIS VAN DER HOEVEN
Relax, but don't be too lazy
`http://www.math.u-psud.fr/~vdhoeven/Publs/relax.ps`

[11] PAUL ZIMMERMANN
Karatsuba square root
Rapport de recherche INRIA n° 3805, `http://www.inria.fr/RRRT/RR3805.html`

[12] GUILLAUME HANROT, MICHEL QUERCIA, PAUL ZIMMERMANN
Speeding up the division and square root of power series
Rapport INRIA à paraître