Ecole Polytechnique **Option informatique**

MULTIPLICATION RAPIDE EN LE_LISP

Paul Zimmermann
X 84
Directeur d'option : P. Cousot

23 juillet 1991

Laboratoire d'accueil : INRIA Directeur de stage : Jean Vuillemin

Table des matières

Chapitre 1

Comment multiplier rapidement de grands entiers ?

1.1 L'algorithme naïf

A l'école primaire, nous avons appris que pour multiplier 751 par 93, il faut écrire:

```
    751
   X 93
    ---
   2253
+ 6759
  -----
  69843
```

en disant

- 3 fois 1 (plus 0) égale 3 : je pose 3 et je retiens 0
- 3 fois 5 plus 0 égale 15 : je pose 5 et je retiens 1
- 3 fois 7 plus 1 égale 22 : je pose 2 et je retiens 2
- je pose le 2 puisque j'ai épuisé les chiffres du multiplicande
- 9 fois 1 (plus 0) égale 9 : je pose 9 et je retiens 0
- 9 fois 5 plus 0 égale 45 : je pose 5 et je retiens 4
- 9 fois 7 plus 4 égale 67 : je pose 7 et je retiens 6
- je pose le 6 qui reste

puis

- 3 plus 0 (plus 0) égale 3 : je pose 3 et je retiens 0
- 5 plus 9 plus 0 égale 14 : je pose 4 et je retiens 1
- 2 plus 5 plus 1 égale 8 : je pose 8 et je retiens 0

- 2 plus 7 plus 0 égale 9 : je pose 9 et je retiens 0
- 0 plus 6 plus 0 égale 6 : je pose 6

Nous avons utilisé deux opérations de base:

1. La multiplication avec retenue d'un chiffre qui, à partir de trois chiffres n, m et q, calcule les deux chiffres p et q' tels que $nm + q = 10q' + p$. On pose p et on retient q'.
2. L'addition avec retenue qui, à partir de deux chiffres n et m, et d'une retenue r valant 0 ou 1, calcule le chiffre p et la retenue r' tels que $n + m + r = 10r' + p$. On pose p et on retient r'.

Les ordinateurs d'aujourd'hui font également toutes leurs multiplications à l'aide d'une ou deux opérations de base. La principale différence est que la base qu'ils utilisent n'est pas 10, mais une puissance de 2 (2^{16} ou 2^{32} le plus couramment). Ceci nous amène à préciser la notion de chiffre.

Définition 1 *En base B, un entier n est un* chiffre *si $0 \leq n < B$.*

Définition 2 *On dit qu'une instruction est élémentaire si tous ses opérandes sont des chiffres.*

Supposons donc que l'on ait une machine disposant des instructions élémentaires suivantes:

- MULADD(p,n,m,q) qui remplace p par p' et retourne q', où p' et q' sont donnés par $p + nm + q = Bq' + p'$. Comme la valeur maximale de $p + nm + q$ est $B - 1 + (B - 1)^2 + B - 1 = B^2 - 1$, cette instruction a bien un sens.
- ADD(p,q) qui remplace p par p' et retourne r où $p + q = Br + p'$ (r vaut 0 ou 1).

Nous disposons de deux entiers N et M ayant respectivement k et l chiffres dans la base B, et d'un entier P ayant $s \geq k + l$ chiffres dans cette même base:

$$\begin{aligned} N &= n_0 + n_1 B + \cdots + n_{k-1} B^{k-1} \\ M &= m_0 + m_1 B + \cdots + m_{l-1} B^{l-1} \\ P &= p_0 + p_1 B + \cdots + p_{s-1} B^{s-1} \end{aligned}$$

Définition 3 *La multiplication-accumulation de NM à P est l'opération qui remplace P par P' et qui renvoie r tel que $P + NM = P' + rB^s$.*

L'algorithme naïf de multiplication-accumulation s'écrit:

Algorithme naïf:

```
r:=0;
for j:=0 to l-1 do begin
    q:=0;
    for i:=0 to k-1 do
        q:=MULADD(p[i+j],n[i],m[j],q);
    t:=k+j;
    while (q≠0) and (t<s) do begin
        q:=ADD(p[t],q);
        t:=t+1;
    end;
    r:=r+q;
end;
return(r);
```

Pour comparer les performances des différentes méthodes que nous allons étudier, nous utiliserons comme critère

Définition 4 *Le* coût $C(k, l)$ *d'un algorithme de multiplication est la moyenne des coûts d'exécution de cet algorithme sur l'ensemble des couples de nombres ayant respectivement k et l chiffres. Le coût d'une exécution est la somme des coûts des opérations élémentaires qu'elle nécessite.*

Définition 5 *Le coût d'une opération élémentaire est le temps nécessaire pour mener à bien cette opération.*

Exemple : soit $t_\times$ le coût de MULADD, et t_+ celui de ADD. Le coût de l'algorithme ci-dessus est par conséquent:

$$C_n(k, l) = kl\, t_\times + l\, t_+$$

En effet, la probabilité que q soit à 0 au début de la boucle **while** est en $\log B/B$, et la probabilité pour qu'il reste non nul est divisée par B à chaque passage dans cette boucle. Elle sera donc effectuée en moyenne une et une seule fois, d'où le terme $l\, t_+$.
Remarque : si $k < l$, il vaut mieux échanger N et M, ce qui conduit à un coût plus faible. Lorsque $k = l = N$, il vient le:

Théorème 1

$$C_n(N) = t_\times N^2 + O(N) \tag{1.1}$$

1.2 Les algorithmes d'évaluation-interpolation

C'est une famille de méthodes permettant de réduire la complexité de la multiplication de deux entiers. Comme c'est souvent le cas, il faut d'abord compliquer le problème de départ pour pouvoir mieux le simplifier. Nous allons donc considérer que les entiers que nous avons à multiplier sont les valeurs au point $X = B$ de polynômes $N(X)$ et $M(X)$:

$$\begin{aligned} N(X) &= n_0 + n_1 X + \cdots + n_{k-1} X^{k-1} \\ M(X) &= m_0 + m_1 X + \cdots + m_l X^{l-1} \end{aligned}$$

Soit $P(X)$ le produit de ces polynômes:

$$P(X) = p_0 + p_1 X + \cdots + p_{k+l-2} X^{k+l-2}$$

où $p_j = \sum_{i=0}^{j} n_i m_{j-i}$ en convenant que les coefficients de N et M hors des intervalles $[0 \ldots k-1]$ et $[0 \ldots l-1]$ sont nuls. Notre but est de calculer la valeur de $P(B)$. Pour cela, il suffit de déterminer les $k + l - 1$ coefficients $p_0, \ldots, p_{k+l-2}$. L'addition d'un de ces termes au résultat coûtera $O(\log j)$ puisque $p_j \leq (j + 1)B^2$ donc a au plus $O(\log j)$ chiffres. Nous allons établir un système linéaire de $k + l - 1$ équations en évaluant P en $k + l - 1$ points entiers distincts $a_0, \ldots, a_{k+l-2}$:

$$\left\{ \begin{aligned} p_0 + a_0 p_1 + \cdots + a_0^{k+l-2} p_{k+l-2} &= N(a_0)M(a_0) \\ p_0 + a_1 p_1 + \cdots + a_1^{k+l-2} p_{k+l-2} &= N(a_1)M(a_1) \\ &\vdots \\ p_0 + a_i p_1 + \cdots + a_i^{k+l-2} p_{k+l-2} &= N(a_i)M(a_i) \\ &\vdots \\ p_0 + a_{k+l-2} p_1 + \cdots + a_{k+l-2}^{k+l-2} p_{k+l-2} &= N(a_{k+l-2})M(a_{k+l-2}) \end{aligned} \right.$$

Il faut choisir les points d'évaluation de telle sorte que:

1. le système soit facile à résoudre
2. les entiers $N(i)$ et $M(i)$ ne soient pas trop grands

1.2.1 L'algorithme de Karatsuba (k=l=2)

Dans le plus simple des cas, c'est-à-dire lorsque k=l=2, on obtient:

$$\begin{aligned} N(X) &= n_0 + n_1 X \\ M(X) &= m_0 + m_1 X \\ P(X) &= p_0 + p_1 X + p_2 X^2 \end{aligned}$$

Il nous faut 3 équations en p_0, p_1 et p_2:

$$\begin{aligned} p_0 &= P(0) &&= n_0 m_0 \\ p_0 + p_1 + p_2 &= P(1) &&= (n_0 + n_1)(m_0 + m_1) \\ p_2 &= P(\infty) &&= n_1 m_1 \end{aligned}$$

Les deux premières sont les évaluations de P en 0 et 1; la troisième provient du fait que le coefficient dominant de P est le produit des coefficients dominants de N et M (on peut assimiler cela à une évaluation en ∞). Supposons à présent que les coefficients n_i et m_i ne sont plus des chiffres, mais des nombres de N chiffres. Pour calculer P(0), nous devons effectuer une multiplication de N chiffres par N chiffres, de même pour P(∞). Pour obtenir P(1), cela n'est pas si simple car les sommes $n_0 + n_1$ et $m_0 + m_1$ peuvent excéder B^N:

$$\begin{aligned} n_0 + n_1 &= n'_0 + n'_1 B^N \\ m_0 + m_1 &= m'_0 + m'_1 B^N \end{aligned}$$

où $0 \leq n'_0, m'_0 < B^N$ et $0 \leq n'_1, m'_1 \leq 1$. Procédons alors comme suit:

1. calculer $n'_0 m'_0$
2. si $n'_1 = 1$, ajouter $m'_0 B^N$
3. si $m'_1 = 1$, ajouter $n'_0 B^N$
4. si $n'_1 = 1$ et $m'_1 = 1$, ajouter B^{2N}

Dans le pire des cas, c'est-à-dire lorsque n'_1 et m'_1 valent 1, nous avons deux additions de N chiffres à effectuer, plus une addition d'une puissance de B. Si nous utilisons cette méthode lorsque $N > N_2$, nous obtenons le système récurrent suivant:

$$C_2(N) = \begin{cases} 3C_2(N/2) + \lambda N/2 & \text{si } N > N_2 \\ C_n(N) & \text{sinon} \end{cases} \tag{1.2}$$

où $\lambda N/2$ représente le coût de toutes les opérations en sus des 3 multiplications. Sa valeur exacte dépend de la façon dont est implémenté l'algorithme (nous en verrons un exemple au prochain chapitre). La solution de ce système est:

$$C_2(2^m N_2) = 3^m C_n(N_2) + (3^m - 2^m)\lambda N_2$$

d'où en posant $N = 2^m N_2$:

$$C_2(N) = \frac{C_n(N_2) + \lambda N_2}{{N_2}^{\log_2 3}} N^{\log_2 3} - \lambda N \tag{1.3}$$

Or N_2 est déterminé par le fait que l'algorithme de Karatsuba est préférable si et seulement si $N > N_2$:

$$N < N_2 \Leftrightarrow C_n(N) < 3C_n(N/2) + \lambda N/2$$

ce qui donne en remplaçant C_n par sa valeur donnée par 1.1:

$$N < N_2 \Leftrightarrow N < 2\frac{t_+ + \lambda}{t_\times}$$

d'où le nombre de chiffres à partir duquel il faut utiliser cette méthode de préférence à l'algorithme naïf:

$$N_2 = \lceil 2\frac{t_+ + \lambda}{t_\times} \rceil \tag{1.4}$$

Revenons au calcul de $C_2(N)$:

$$C_n(N_2) + \lambda N_2 = 3(t_+ + \lambda)N_2$$

d'où le résultat final:

Théorème 2

$$C_2(N) = c_2 N^{\log_2 3} + O(N)$$

avec

$$c_2 = 2(t_+ + \lambda)^{2-\log_2 3} {t_\times}^{log_2 3 - 1} \ \ et \ \ \log_2 3 \simeq 1.585$$

Remarque : si l'on introduit dans 1.2 un terme μ représentant les coûts constants (dus au contrôle par exemple), le coefficient de $N^{log_2 3}$ dans 1.3 devient

$$\frac{C_n(N_2) + \lambda N_2 + \mu/2}{N_2^{log_2 3}}$$

et la formule 1.4 devient

$$N_2 = \lceil 2\frac{t_+ + \lambda}{t_\times} + 2\frac{\mu}{t_+ + \lambda} \rceil$$

1.2.2 Le cas général ($k > 2$)

Celui-ci est entièrement décrit par Knuth [2, page 282] et est dû en partie à Toom. Il permet de réduire la multiplication de nombres de kN chiffres à $2k-1$ produits de nombres de $N+t$ chiffres, où t ne dépend que de k. La multiplication de deux nombres de $N+t$ chiffres peut se faire par une multiplication de N par N, plus deux de N par t, plus une de t par t. Les trois dernières ont un coût au plus en $O(N)$, donc on peut les faire rentrer sous le terme λN:

$$C_k(N) = \begin{cases} (2k-1)C_k(N/k) + \lambda N/k & \text{si } N > N_k \\ C_n(N) & \text{sinon} \end{cases}$$

La solution de ce système est, si N_k est le nombre de chiffres critique:

$$C_k(k^m N_k) = (2k-1)^m C_n(N_k) + ((2k-1)^m - k^m)\lambda N_k$$

d'où en posant $N = k^m N_k$ et $\log_k(2k-1) = 1 + \epsilon_k$:

$$C_k(N) = \frac{C_n(N_k) + \lambda N_k}{{N_k}^{1+\epsilon_k}} N^{1+\epsilon_k} - \lambda N$$

Or N_k est déterminé par le fait que l'algorithme de Karatsuba est préférable si et seulement si $N > N_k$:

$$N < N_k \Leftrightarrow C_n(N) < (2k-1)C_n(N/k) + \lambda N/k$$

ce qui donne en remplaçant C_n par sa valeur donnée par 1.1:

$$N < N_k \Leftrightarrow N < k\frac{(k-1)t_+ + \lambda}{(k-1)^2 t_\times}$$

d'où la valeur à partir de laquelle il faut utiliser cette méthode de préference à l'algorithme naïf:

$$N_k = \lceil k\frac{(k-1)t_+ + \lambda}{(k-1)^2 t_\times}\rceil$$

Revenons au calcul de $C_k(N)$:

$$C_n(N_k) + \lambda N_k = \frac{2k-1}{(k-1)^2}((k-1)t_+ + \lambda)N_k$$

d'où le résultat final:

Théorème 3

$$C_k(N) = c_k N^{1+\epsilon_k} + O(N)$$

avec

$$c_k = \frac{k}{(k-1)^2}((k-1)t_+ + \lambda)^{1-\epsilon_k}((k-1)^2 t_\times)^{\epsilon_k}$$

et $1 + \epsilon_k = log_k(2k-1)$.

Comment se comporte ϵ_k lorsque $k \to \infty$?

$$\begin{aligned} \epsilon_k &= \log_k(2k-1) - 1 \\ &= \frac{\log \frac{2k-1}{k}}{\log k} \\ &< \frac{\log 2}{\log k} \end{aligned}$$

Si l'on désire $\epsilon_k < \epsilon$, il suffit de choisir $k > 2^{1/\epsilon}$.

Théorème 4 *Pour tout $\epsilon > 0$, il existe une constante C_ϵ et un algorithme de multiplication dont le coût est inférieur à $C_\epsilon N^{1+\epsilon}$: par exemple la méthode de Toom avec $k > 2^{1/\epsilon}$.*

Lorsque $\epsilon \to 0$, $k = \lceil 2^{1/\epsilon} \rceil \to +\infty$, d'où:

$$C_\epsilon = c_k \approx t_+ + \frac{\lambda}{k}$$

Que vaut λ ? Le calcul de chaque $A(r)$, $0 \leq r \leq 2k-2$, par la méthode de factorisation suivante:

$$A(r) = a_0 + r(a_1 + r(a_2 + \cdots (a_{k-2} + r a_{k-1}) \ldots))$$

requiert $k-1$ multiplications de r (ayant $\log k$ chiffres) par un entier d'au moins N/k chiffres, en supposant que chaque a_i a au N/k chiffres. Effectué par la méthode naïve, chacun de ces calculs a donc un coût $O(k \log k \frac{N}{k})$, d'où un coût total de $O(k^2 \log k \frac{N}{k})$. Comme on pouvait s'en douter, λ dépend de k, et plus précisément, λ est au moins en $O(k^2 \log k)$ (on a seulement évalué le coût de l'évaluation).

1.2.3 Le cas k=l=3

On considère alors des polynômes de degré deux:

$$\begin{aligned}
N(X) &= n_0 + n_1 X + n_2 X^2 \\
M(X) &= m_0 + m_1 X + m_2 X^2 \\
P(X) &= p_0 + p_1 X + p_2 X^2 + p_3 X^3 + p_4 X^4
\end{aligned}$$

Evaluons P aux points 0, 1, 2, 3 et 4 (un autre choix possible est -2, -1, 0, 1 et 2):

$$\begin{aligned}
P(0) &= p_0 \\
P(1) &= p_0 + p_1 + p_2 + p_3 + p_4 \\
P(2) &= p_0 + 2p_1 + 4p_2 + 8p_3 + 16p_4 \\
P(3) &= p_0 + 3p_1 + 9p_2 + 27p_3 + 81p_4 \\
P(4) &= p_0 + 4p_1 + 16p_2 + 64p_3 + 256p_4
\end{aligned}$$

soit

$$\begin{bmatrix} P(0) \\ P(1) \\ P(2) \\ P(3) \\ P(4) \end{bmatrix} = A \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

avec

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \end{bmatrix}$$

Pour obtenir les p_j, il faut inverser A:

$$A^{-1} = \frac{1}{24} \begin{bmatrix} 24 & 0 & 0 & 0 & 0 \\ -50 & 96 & -72 & 32 & -6 \\ 35 & -104 & 114 & -56 & 11 \\ -10 & 36 & -48 & 28 & -6 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix}$$

Supposons $0 \leq n_i, m_i < B^N$, alors $0 \leq N(j), M(j) < 341B^N$ (341=1+4+16+64+256). Si $B > 341$, la multiplication de deux nombres de $3N$ chiffres se réduit ainsi à 5 multiplications de nombres de $N+1$ chiffres, plus des opérations de coût $O(N)$, comme on l'a vu dans le cas général.

Nous avons détaillé ce cas pour montrer que l'évaluation et l'interpolation ne sont plus de simples additions ou soustractions comme dans le cas $k = 2$, mais nécessitent des multiplications et divisions par des constantes entières. Knuth propose une façon habile d'effectuer ces calculs d'évaluation et d'interpolation dans le cas général. De plus, il propose un algorithme où le choix de k dépend de la taille de N et M, ce qui le conduit à un coût $O(N2^{3.5\sqrt{\log N}})$. Moyennant quelques complications, il déclare pouvoir améliorer celui-ci jusqu'à un coût de l'ordre de:

$$C(N) = O(N2^{\sqrt{2\log N}} \log N)$$

1.3 Algorithmes utilisant la transformée de Fourier discrète

Ces algorithmes ressemblent beaucoup à l'évaluation-interpolation, car ils comportent également cinq étapes:

1. découpage des nombres de départ en entiers plus petits
2. transformation de ces petits entiers
3. multiplication terme à terme
4. transformation des produits
5. construction du produit final

La différence est que le coût des étapes 2 et 4, effectuées à l'aide de la transformée de Fourier rapide (FFT), n'est que $O(N \log N)$. De plus, le nombre de morceaux en lesquels on scinde les entiers initiaux n'est plus fixe, mais varie suivant la taille de ces entiers. Définissons tout d'abord la transformée de Fourier que nous allons utiliser.

1.3.1 La transformée de Fourier

Soit Z un anneau commutatif, et ω un élément de Z. On définit la transformée de Fourier sur un vecteur d'éléments de Z:

Définition 6 *La transformée de Fourier du vecteur $(a_0, \ldots, a_{n-1})$, notée $F(a)$, est le vecteur $(\hat{a}_0, \ldots, \hat{a}_{n-1})$ tel que:*

$$\hat{a}_i = \sum_{j=0}^{n-1} \omega^{ij} a_j \tag{1.5}$$

C'est une application de Z^n dans lui-même. Il nous faut maintenant trouver à quelle condition la transformée inverse existe. Celle-ci s'obtient généralement en appliquant une seconde transformée:

$$\begin{aligned} \hat{\hat{a}}_l &= \sum_{i=0}^{n-1} \omega^{il} \hat{a}_i \\ &= \sum_{i=0}^{n-1} \omega^{il} \sum_{j=0}^{n-1} \omega^{ij} a_j \end{aligned}$$

$$= \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} \omega^{i(l+j)}$$
$$= \sum_{j=0}^{n-1} a_j S_{l+j}$$

où

$$S_m = \sum_{i=0}^{n-1} \omega^{im}$$

Pour obtenir une bijection, il faut que $\hat{\hat{a}}_l$ ne dépende que d'un seul a_j, donc qu'il existe une permutation ϕ de $(0, 1, \ldots, n-1)$ telle que:

i) $S_{l+\phi(l)} \neq 0$

ii) $S_{l+j} = 0$ pour $j \neq \phi(l)$

Ceci nous amène au concept de n-ième racine principale de l'unité:

Définition 7 *Un élément $\omega \neq 1$ de Z est une* n-ième racine principale *de l'unité si:*

$$\begin{array}{ll} i) & \omega^n = 1 \\ ii) & \displaystyle\sum_{i=0}^{n-1} \omega^{im} = 0 \text{ pour } 1 \leq m < n \end{array} \tag{1.6}$$

De tels éléments existent. En effet, si Z=C, $e^{\frac{2i\pi}{n}}$ est une n-ième racine principale de l'unité. La somme S_m devient alors:

$$S_m = \begin{cases} n & \text{si } m \equiv 0 \bmod n \\ 0 & \text{sinon} \end{cases}$$

d'où

$$\begin{cases} \hat{\hat{a}}_0 &= na_0 \\ \hat{\hat{a}}_j &= na_{n-j} \text{ pour } 1 \leq j < n \end{cases}$$

ce qui conduit au:

Théorème 5 *Soit Z un anneau commutatif, n un entier inversible dans Z et $\omega \in Z$ une n-ième racine principale de l'unité. Alors F est une bijection de Z^n dans lui-même, et la bijection inverse est définie par:*

$$F^{-1}(a_0, \ldots, a_{n-1}) = (\tilde{a}_0, \ldots, \tilde{a}_{n-1})$$

avec

$$\begin{cases} \tilde{a}_0 &= n^{-1}\hat{a}_0 \\ \tilde{a}_j &= n^{-1}\hat{a}_{n-j} \text{ pour } 1 \leq j < n \end{cases} \tag{1.7}$$

L'hypothèse que n est inversible signifie que l'élément $\overbrace{1 + \cdots + 1}^{n}$ a un inverse pour la multiplication dans Z.

Puisque nous avons maintenant défini la transformée de Fourier, nous allons étudier ses propriétés, puis voir comment l'appliquer au produit de polynômes.

1.3.2 Propriétés de F

Observons ce qu'il advient lorsque nous multiplions terme à terme les transformées de deux vecteurs. Soit $a = (a_0, \ldots, a_{n-1})$, $b = (b_0, \ldots, b_{n-1})$ puis $c = (c_0, \ldots, c_{n-1})$ avec $c_k = \hat{a}_k \hat{b}_k$. On note $c = \hat{a} \otimes \hat{b}$.

$$\begin{aligned}
\hat{c}_l &= \sum_{k=0}^{n-1} \omega^{kl} c_k \\
&= \sum_{k=0}^{n-1} \omega^{kl} \hat{a}_k \hat{b}_k \\
&= \sum_{k=0}^{n-1} \omega^{kl} (\sum_{i=0}^{n-1} \omega^{ik} a_i)(\sum_{j=0}^{n-1} \omega^{jk} b_j) \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j (\sum_{k=0}^{n-1} \omega^{k(i+j+l)}) \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j S_{i+j+l} \\
&= n \sum_{\substack{i+j+l \equiv 0 \bmod n \\ 0 \leq i,j < n}} a_i b_j
\end{aligned}$$

puis

$$\tilde{c}_l = \sum_{\substack{i+j \equiv l \bmod n \\ 0 \leq i,j < n}} a_i b_j$$

Définition 8 *La* convolution cyclique positive *de deux vecteurs a et b est le vecteur $c = a \oplus b$ tel que:*

$$c_l = \sum_{i=0}^{l} a_i b_{l-i} + \sum_{i=l+1}^{n-1} a_i b_{n+l-i}$$

Théorème 6 $\widetilde{(\hat{a} \otimes \hat{b})} = a \oplus b$

On peut aussi définir la convolution cyclique négative :

Définition 9 *La* convolution cyclique négative *de deux vecteurs a et b est le vecteur $d = a \ominus b$ tel que:*

$$d_l = \sum_{i=0}^{l} a_i b_{l-i} - \sum_{i=l+1}^{n-1} a_i b_{n+l-i}$$

Celle-ci se calcule également à l'aide de F. Si a est un vecteur de n éléments de Z, et si $\theta \in Z$ vérifie $\theta^n = -1$, on notera a' le vecteur $(a'_0, \ldots, a'_{n-1})$ tel que:

$$a'_i = \theta^i a_i$$

Appliquons le théorème précédent à a' et b':

$$\widetilde{(\hat{a}' \otimes \hat{b}')} = a' \oplus b'$$

Soit alors $c = a' \oplus b'$ et $d = a \ominus b$:

$$
\begin{aligned}
c_l &= \sum_{i=0}^{l} a'_i b'_{l-i} + \sum_{i=l+1}^{n-1} a'_i b'_{n+l-i} \\
&= \sum_{i=0}^{l} \theta^l a_i b_{l-i} + \sum_{i=l+1}^{n-1} \theta^{n+l} a_i b_{n+l-i} \\
&= \sum_{i=0}^{l} \theta^l a_i b_{l-i} - \sum_{i=l+1}^{n-1} \theta^l a_i b_{n+l-i} \\
&= d'_l
\end{aligned}
$$

Nous avons démontré le

Théorème 7 $(\widetilde{\hat{a}' \otimes \hat{b}'}) = (a \ominus b)'$

1.3.3 Application au produit polynômial

Soient deux polynômes de degré $n-1$, à coefficients dans Z:

$$
\begin{aligned}
A(X) &= a_0 + a_1 X + \cdots + a_{n-1} X^{n-1} \\
B(X) &= b_0 + b_1 X + \cdots + b_{n-1} X^{n-1}
\end{aligned}
$$

Les coefficients c_l de $C(X) = A(X)B(X)$ sont de la forme:

$$
c_l = \sum_{i=0}^{l} a_i b_{l-i} \quad 0 \le l \le 2n-2
$$

en convenant que $a_k = b_k = 0$ pour $n \le k \le 2n-1$. Nous désirons donc obtenir la convolution simple de $a = (a_0, \ldots, a_{n-1})$ et $b = (b_0, \ldots, b_{n-1})$, et non une convolution cyclique. Nous pouvons cependant l'en déduire. En effet, si nous récrivons la formule ci-dessus:

$$
c_l = \sum_{i=0}^{l} a_i b_{l-i} + \sum_{i=l+1}^{2n-1} a_i b_{2n+l-i} \tag{1.8}
$$

La seconde sommation est nulle puisque lorsque $i < n$, $2n + l - i > n$. Nous avons exprimé la convolution normale d'ordre n à l'aide de la convolution cyclique d'ordre $2n$, ce qui nous permet d'écrire:

Théorème 8 *Les coefficients de C sont les $2n-1$ premiers termes du vecteur $c = (\widetilde{\hat{\bar{a}} \otimes \hat{\bar{b}}})$ où $\bar{a} = (a_0, \ldots, a_{n-1}, \underbrace{0, \ldots, 0}_{n})$ et $\bar{b} = (b_0, \ldots, b_{n-1}, \underbrace{0, \ldots, 0}_{n})$.*

Bien entendu, il s'agit dans ce théorème de transformées d'ordre $2n$. D'autre part, le dernier terme est nul puisque 1.8 nous donne 0 pour $l = 2n-1$.

Jusqu'ici, nous n'avons pas choisi l'anneau commutatif Z où nous travaillons. Nous présentons ci-dessous deux possibilités intéressantes, l'une dans un anneau infini, en l'occurrence le corps des complexes, l'autre dans un anneau fini, la classe des restes modulo $2^N + 1$. Mais voyons d'abord comment effectuer rapidement la transformation de Fourier.

1.3.4 Qu'est-ce que la FFT ?

Ce sigle désigne l'expression anglaise *Fast Fourier Transform*, ce qui signifie transformée de Fourier rapide. En effet, d'après 1.5, il apparaît que chaque $\hat{a}_i$ est une somme de n termes. Le calcul de chacun de ces n éléments par cette formule a donc un coût $O(n^2)$. S'il n'y avait pas d'autre moyen d'effectuer une transformée de Fourier, celle-ci aurait beaucoup moins d' intérêt.

Mais il existe pour des valeurs particulières de n, en l'occurrence $n = 2^k$, une autre méthode de coût $O(nk) = O(n \log n)$. Nous allons présenter en premier lieu le cas $n = 2$, puis nous déduirons le cas $n = 2^k$ du cas $n = 2^{k-1}$.

Lorsque n=2, le fait que ω soit une racine principale de l'unité signifie simplement que $\omega^2 = 1$ et $1 + \omega = 0$, donc $\omega = -1$. Par conséquent, il vient:

$$\begin{aligned} \hat{a}_0 &= a_0 + a_1 \\ \hat{a}_1 &= a_0 - a_1 \end{aligned}$$

Si $n = 2^k$, nous définissons $a'_j = a_{2j}$ et $a''_j = a_{2j+1}$ pour $0 \leq j < n/2$. Pour $i < n/2$:

$$\begin{aligned} \hat{a}_i &= \sum_{l=0}^{n-1} \omega^{li} a_l \\ &= \sum_{j=0}^{n/2-1} (\omega^{2ji} a_{2j} + \omega^{(2j+1)i} a_{2j+1}) \\ &= \sum_{j=0}^{n/2-1} (\omega^2)^{ji} a'_j + \omega^i \sum_{j=0}^{n/2-1} (\omega^2)^{ji} a''_j \\ &= \hat{a}'_i + \omega^i \hat{a}''_i \end{aligned} \tag{1.9}$$

La dernière ligne demande quelques explications. Premièrement, il s'agit là de transformées d'ordre $n/2$ sur les vecteurs a' et a''. Ensuite, celles-ci ne sont définies que si ω^2 est une racine $\frac{n}{2}$-ième principale de l'unité. La première condition de 1.6 est immédiatement vérifiée:

$$(\omega^2)^{n/2} = \omega^n = 1$$

Quant à la seconde, si $1 \leq m < n/2$, on a, puisque ω est une racine n-ième de l'unité:

$$S_{2m} = 0$$

Or cette somme vaut

$$\begin{aligned} S_{2m} &= \sum_{i=0}^{n/2-1} \omega^{2im} + \sum_{i=n/2}^{n-1} \omega^{2im} \\ &= \sum_{i=0}^{n/2-1} (\omega^2)^{im} + \omega^{nm} \sum_{i'=0}^{n/2-1} (\omega^2)^{i'm} \\ &= 2 \sum_{i=0}^{n/2-1} (\omega^2)^{im} \end{aligned}$$

S_{2m} étant nul par hypothèse, ω^2 est bien une racine $\frac{n}{2}$-ième principale de l'unité si 2 est inversible dans Z.

La formule 1.9 nous donne $\hat{a}_i$ uniquement pour $i < n/2$. Si $i = n/2 + i'$, avec $0 \le i' < n/2$, alors:

$$\begin{aligned}
\hat{a}_i &= \sum_{l=0}^{n-1} \omega^{l(n/2+i')} a_l \\
&= \sum_{j=0}^{n/2-1} (\omega^{2j(n/2+i')} a_{2j} + \omega^{(2j+1)(n/2+i')} a_{2j+1}) \\
&= \sum_{j=0}^{n/2-1} (\omega^2)^{ji'} a'_j - \omega^{i'} \sum_{j=0}^{n/2-1} (\omega^2)^{ji'} a''_j \\
&= \hat{a}'_{i'} - \omega^{i'} \hat{a}''_{i'}
\end{aligned}$$

Les formules permettant de calculer la FFT d'ordre n à partir de celle d'ordre $n/2$ sont donc:

$$\left.\begin{aligned}
\hat{a}_i &= \hat{a}_{2i} + \omega^i \hat{a}_{2i+1} \\
\hat{a}_{n/2+i} &= \hat{a}_{2i} - \omega^i \hat{a}_{2i+1}
\end{aligned}\right\} 0 \le i < n/2 \tag{1.10}$$

Le coût de la FFT vérifie par conséquent:

$$C_{fft}(n) = 2C_{fft}(n/2) + \lambda n$$

en considérant que les opérations sur Z (addition, soustraction, produit) ont un coût constant. On obtient bien le résultat annoncé:

Théorème 9 *La transformée de Fourier effectuée à l'aide de la FFT a un coût $O(n \log n)$.*

1.3.5 Méthode de Strassen

Choisissons donc Z=C, le corps des complexes. Le fait de travailler dans un anneau infini nous oblige à n'avoir qu'une représentation approchée des éléments de Z. Tous les calculs seront donc faits avec une précision fixe, que l'on déterminera plus loin.

La méthode de Strassen est une application directe du théorème 8. Pour multiplier

$$a_0 + a_1 B + \cdots + a_{n-1} B^{n-1}$$

par

$$b_0 + b_1 B + \cdots + b_{n-1} B^{n-1}$$

nous formons $\bar{a} = (a_0, \ldots, a_{n-1}, \overbrace{0, \ldots, 0}^{n})$ et $\bar{b} = (b_0, \ldots, b_{n-1}, \overbrace{0, \ldots, 0}^{n})$, puis $\tilde{c} = \widetilde{(\hat{\bar{a}} \otimes \hat{\bar{b}})}$. Les $\tilde{c}_l$ sont les produits de convolution cherchés, donc des entiers. Comme nous ne pouvons avoir une arithmétique exacte dans C, la suite de calculs que nous allons effectuer ne nous donnera qu'un résultat approché, par exemple $\tilde{c}'_l = \tilde{c}_l + \delta$ avec $\delta \in C$. Pour pouvoir retrouver l'entier $\tilde{c}_l$ à partir de $\tilde{c}'_l$, il faut et il suffit que la partie réelle de δ soit strictement inférieure à 1/2 en valeur absolue.

Nous allons utiliser une arithmétique complexe de précision ϵ, c'est-à-dire qu'à chaque opération, il faudra prendre en compte une erreur relative de troncature égale à ϵ.

La taille de la FFT est $2n$, donc nous choisissons $\omega = e^{\frac{2i\pi}{2^k}}$ où $2^k = 2n$. La première étape est le calcul des puissances de ω, c'est-à-dire ω^j pour $0 \le j < 2^k$. Nous les calculons avec une erreur d'au plus ϵ. Nous supposons de plus que ces valeurs approchées sont de module inférieur ou égal à 1.

La FFT se décompose en k étapes du type

$$\left\{ \begin{array}{lcl} a_{i,l+1} & \longleftarrow & a_{i,l} + \omega^l a_{j,l} \\ a_{j,l+1} & \longleftarrow & a_{i,l} - \omega^l a_{j,l} \end{array} \right. \tag{1.11}$$

où i (respectivement j) parcourt n valeurs parmi $0, 1, \ldots, 2n-1$, et $a_{i,0} = a_i$. Il nous faut donc déterminer l'erreur δ_a commise sur l'opération approchée $a' \longleftarrow b' + \omega' c'$ par rapport à l'opération exacte $a \longleftarrow b + \omega c$, avec $|b| < m$ et $|c| < m$:

$$\begin{array}{rcl} \delta_a = |a' - a| & = & |b' - b| + |(\omega c)' - \omega c| + 2\epsilon m \ (*) \\ & = & |b' - b| + |\omega' c' - \omega c| + \epsilon m + 2\epsilon m \ (**) \\ & \leq & \delta_b + |\omega'(c' - c) + (\omega' - \omega)c| + 3\epsilon m \\ & \leq & \delta_b + |\omega'(c' - c)| + |(\omega' - \omega)c| + 3\epsilon m \\ & \leq & \delta_b + \delta_c + \epsilon m + 3\epsilon m \\ & \leq & \delta_b + \delta_c + 4\epsilon m \end{array} \tag{1.12}$$

A la ligne (*), $2\epsilon m$ est l'erreur de troncature sur la somme $b' + (\omega c)'$, de module inférieur à $2m$. En (**), ϵm est l'erreur de troncature sur le produit $\omega' c'$, de module borné par m.

Notons δ_l l'erreur maximale sur les $a_{i,l}$ ($\delta_0 = 0$ puisqu'on part d'entiers), et $m_l = sup_i a_{i,l}$ ($m_0 = B$). Il découle de 1.11 et de 1.12 que:

$$\delta_{l+1} \leq 2\delta_l + 4\epsilon m_l \tag{1.13}$$

Il vient également:

$$m_{l+1} \leq 2m_l$$

dont on déduit immédiatement

$$m_l \leq 2^l B$$

que l'on remplace dans 1.13:

$$\delta_{l+1} \leq 2\delta_l + 4\epsilon 2^l B$$

Cette relation se résout en

$$\delta_l \leq 2^l \delta_0 + l 2^{l+1} \epsilon B$$

Par deux FFT de taille $2n$, nous avons obtenu $\hat{a}_0, \ldots, \hat{a}_{2n-1}$ et $\hat{b}_0, \ldots, \hat{b}_{2n-1}$, chacun de ces nombres complexes ayant un module inférieur à m_k et une erreur majorée par δ_k. Puis nous effectuons le produit terme à terme $c_j = \hat{a}_j \hat{b}_j$. Les c_j sont de module borné par $m'_0 = {m_k}^2$, avec une précision de $\delta'_0 = 2m_k\delta_k$. Nous effectuons alors une troisième FFT sur $c_0, \ldots, c_{2n-1}$. Le même calcul que ci-dessus (sauf que $\delta'_0 \neq 0$) nous donne le module maximum des $\hat{c}_j$:

$$m'_k = 2^k m'_0 = 2^k (2^k B)^2 = 2^{3k} B^2$$

et une majoration de l'erreur:

$$\begin{array}{rcl} \delta'_k & \leq & 2^k \delta'_0 + k 2^{k+1} \epsilon m'_0 \\ & \leq & 2^k (2m_k \delta_k) + k 2^{k+1} \epsilon {m_k}^2 \\ & \leq & 6k 2^{3k} \epsilon B^2 \end{array}$$

Pour obtenir les $\tilde{c}_j$, il faut encore diviser les $\hat{c}_j$ par $2n = 2^k$, ce qui nous donne une erreur finale majorée par

$$6k2^{2k}\epsilon B^2$$

Celle-ci devant être inférieure à $\frac{1}{2}$, la condition sur ϵ est:

$$\epsilon < \frac{1}{12k2^{2k}B^2}$$

La FFT nous permet par conséquent de remplacer une multiplication de nombres de taille $N = n \log B$ par $2n$ multiplications de nombres de taille

$$-\log \epsilon = 2 \log B + 2k + \log 12k = 2N/n + 2 \log 2n + \log 12 \log 2n$$

Si chacune d'entre elles est effectuée par la méthode naïve, le coût de ces produits est:

$$O(n(N/n + \log n)^2)$$

Comme les nombres traités ont une longueur $\log 12k2^{2k}B^2 = O(k + \log B) = O(\log n + N/n)$, la FFT a un coût:

$$O(n \log n(\log n + N/n)^2)$$

puisqu'elle requiert $n \log n$ étapes du type $a \longleftarrow b + \omega c$, de coût unitaire $(\log n + N/n)^2$ si l'on effectue aussi le produit ωc par la méthode naïve. C'est donc la FFT qui fixe le coût total de l'algorithme, soit

$$C(N, n) = n \log n(N/n + \log n)^2$$

Pour obtenir un coût optimal, il faut équilibrer les termes N/n et $\log n$ en prenant $n = \frac{N}{\log N}$ et il vient:

$$C(N) = N(\log N)^2$$

Théorème 10 *Le coût de la multiplication en utilisant une fois la méthode de Strassen est*

$$C(N) = N(\log N)^2$$

On peut aussi utiliser cette même méthode pour les produits terme à terme $c_l = \hat{a}_l \hat{b}_l$, ainsi que pour ceux du genre $a \longleftarrow b + \omega c$ de la FFT et l' on arrive alors à

$$C(N) = N \log N(\log \log N)^2$$

Plus généralement, en utilisant récursivement cet algorithme, la borne la plus basse à laquelle on puisse arriver est

$$C(N) = N \log N \log \log N \log \log \log N \ldots$$

Théorème 11 *Le coût de la méthode de Strassen utilisée récursivement est*

$$C_{st}(N) = N \log N \log \log N \log \log \log N \ldots$$

1.3.6 Méthode de Schönhage

Nous avons aperçu les difficultés que l'on rencontre lorsque l'on travaille dans un anneau infini. Comme nous ne pouvons y utiliser qu'une arithmétique finie, il faut à tout moment se soucier de la précision des calculs que l'on effectue. Schönhage a résolu tous ces problèmes en se plaçant dans un anneau fini.

Définition 10 *On note Z_p l'anneau des restes modulo p. Un élément de Z_p est appelé un résidu.*

Définition 11 *On note F_N l'entier $2^N + 1$.*

L'anneau de travail sera Z_{F_N}, avec $N = 2^k M$. La taille de la FFT sera $n = 2^k$, et il nous faut donc trouver une n-ième racine principale de l'unité dans Z_{F_N}.

Choix d'une racine principale de l'unité

Il y avait quand même un avantage à se trouver dans un corps: le choix d'une telle racine était vite fait. Nous devons chercher ω tel que:

- $1 < \omega \leq 2^N$
- $\omega^n \equiv 1 \bmod F_N$
- $\sum_{i=0}^{n-1} \omega^{im} \equiv 0 \bmod F_N$ pour $1 \leq m < n$

La seconde condition peut être aisément réalisée en prenant $\omega = 2^{2M}$ car

$$\omega^n = 2^{2nM} = 2^{2N} \equiv 1 \bmod F_N$$

La somme de la troisième condition peut s'écrire aussi:

$$\begin{aligned} S_m &= (1+\omega^m)(1+\omega^{2m})\ldots(1+\omega^{2^{k-1}m}) \\ &= (1+2^{2Mm})(1+2^{4Mm})\ldots(1+2^{Nm}) \end{aligned}$$

Décomposons m sous la forme du produit d'un nombre impair par une puissance de 2, soit $m = (2\mu+1)2^\lambda$ avec $0 \leq lambda < k$. Le terme d'indice $k - \lambda$ de S_m est alors nul:

$$1 + 2^{2^{k-\lambda}Mm} = 1 + 2^{2^{k-\lambda}M(2\mu+1)2^\lambda} = 1 + 2^{(2\mu+1)2^k M} = 1 + 2^{(2\mu+1)N} \equiv 0 \bmod F_N$$

Multiplication dans Z_{F_N}

Dans cette section, & désignera notre base de travail, c'est-à-dire 2^M. Soient A et B deux éléments de Z_{F_N}:

$$\begin{aligned} A &= a_0 + a_1\& + \cdots + a_{n-1}\&^{n-1} \\ B &= b_0 + b_1\& + \cdots + b_{n-1}\&^{n-1} \end{aligned}$$

où $0 \leq a_i, b_i < \&$ pour $0 \leq i < n-1$ et $0 \leq a_{n-1}, b_{n-1} \leq \&$. Calculons AB modulo F_N:

$$\begin{aligned} AB &= (\sum_{i=0}^{n-1} a_i \&^i)(\sum_{j=0}^{n-1} b_j \&^j) \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j \&^{i+j} \end{aligned}$$

or $\&^n = 2^{nM} \equiv -1 \bmod F_N$, d'où

$$AB = \sum_{\mu=0}^{n-1} \Big(\sum_{\substack{0 \le i,j < n \\ i+j=\mu}} a_i b_j - \sum_{\substack{0 \le i,j < n \\ i+j=n+\mu}} a_i b_j \Big) \&^\mu = \sum_{\mu=0}^{n-1} c_\mu \&^\mu \tag{1.14}$$

où l'on définit $c = a \ominus b$. Nous allons maintenant appliquer le théorème 7:

$$(\widetilde{\hat{a}' \otimes \hat{b}'}) = (a \ominus b)'$$

Pour pouvoir définir a', il faut trouver $\theta \in Z_{F_N}$ tel que $\theta^n = -1$. La valeur $\theta = 2^M$ convient. L'algorithme de Schönhage s'écrit alors:

1. Calculer a' et b'
2. Effectuer les FFT donnant $\hat{a}'$ et $\hat{b}'$
3. Multiplier terme à terme pour obtenir $\hat{c}'$
4. Effectuer une FFT inverse donnant c'
5. Calculer c à partir de c'
6. Ajouter les c_μ suivant la formule 1.14

Il nous reste à choisir l'anneau dans lequel nous effectuerons la FFT et les produits terme à terme. Nous prendrons un $Z_{F_{N'}}$, pour pouvoir utiliser récursivement la méthode. Après l'étape 5, nous aurons donc calculé $d_\mu = c_\mu \bmod F_{N'}$. Or ce qui nous intéresse est la valeur exacte des c_μ. Pour pouvoir y accéder, il suffit que l'intervalle de variation de ceux-ci soit inférieur à $2^{N'}$. Quelles valeurs peut donc prendre c_μ ?

$$c_\mu = a_0 b_\mu + \cdots + a_\mu b_0 + a_{\mu+1} b_{n-1} + \cdots + a_{n-1} b_{\mu+1}$$

Il y a $\mu + 1$ termes positifs et $n - (\mu + 1)$ termes négatifs, par conséquent

$$-(n - (\mu + 1))\&^2 \le c_\mu \le (\mu + 1)\&^2 \tag{1.15}$$

Les c_μ varient dans des intervalles différents, mais de longueur commune $n\&^2$. On doit vérifier $n\&^2 \le 2^{N'}$, soit $N' \ge 2M + k$. Une fois les d_μ calculés, on retrouve les c_μ par:

$$c_\mu = \begin{cases} d_\mu & \text{si } d_\mu \le (\mu + 1)\&^2 \\ d_\mu - F_{N'} & \text{sinon} \end{cases}$$

Application au produit entier

Soient A et B deux entiers de p et q bits respectivement. Choisissons $N \ge \sup(p, q)$. A et B sont alors égaux à leurs résidus modulo F_N. La multiplication de A par B dans Z_{F_N} nous donnera la valeur du produit AB modulo F_N. Comme ce produit a $p + q$ bits, il suffit de prendre $N \ge p + q$ pour que ce produit modulaire soit le produit entier de A par B. Cela conduit aux schémas de calcul ci-dessous:

$$\boxed{\quad p \quad} \times_{ent} \boxed{\quad q \quad} \Longrightarrow \boxed{\qquad p+q \qquad} \times_{mod} \boxed{\qquad p+q \qquad}$$

$$\boxed{\qquad 2^k M \qquad} \times_{mod} \boxed{\qquad 2^k M \qquad} \Longrightarrow 2^k \text{ fois } \boxed{\; 2M+k \;} \times_{mod} \boxed{\; 2M+k \;}$$

Chapitre 2

Implémentation de la méthode de Karatsuba

Nous arrivons à un endroit où il faut préciser quelles sont les instructions élémentaires dont on se dote. Les résultats qui suivent dépendent donc du choix (ou de la restriction) de ces instructions.

Notre philosophie a été de définir un ensemble de fonctions de base, écrites en assembleur, qui effectuent des opérations simples. Les fonctions de cet ensemble sont appelées *fonctions LLM4*. Nous avons préféré une représentation vectorielle des entiers à la représentation arborescente qui avait été implémentée par Jean Vuillemin.

Le vecteur de chiffres $[n_{nd}, \ldots, n_{nd+nl-1}]$ représente l'entier $N = n_{nd} + \cdots + n_{nd+nl-1}B^{nl-1}$ noté $val < n[nd], \ldots, n[nd+nl-1] >$, où $B = 2^{32}$ (nous avons aussi une implémentation avec $B = 2^{16}$). Ce vecteur est repéré par le nom n du tableau auquel il appartient, l'indice nd de début dans ce tableau et sa longueur nl. Le i-ème chiffre du tableau n est noté $n[i]$.

Les fonctions LLM4 qui vont nous être utiles sont:

N→N(m,md,ml,n,nd) Remplace n[nd+i] par m[md+i] pour $0 \leq i < ml$. Si $t_{N \to N}$ est le coût de recopie d'un chiffre, le coût de cette fonction vaut $ml\, t_{N \to N}$.

razN(n,nd,nl) Met à 0 $n[nd] \ldots n[nd+nl-1]$. Son coût est $nl\, t_{razN}$.

N+b(n,nd,nl,r) Propage la retenue r valant 0 ou 1 sur les nl chiffres $n[nd] \ldots n[nd+nl-1]$. Supposons qu'à l'appel, r vaut 1 avec une probabilité $\frac{1}{2}$. La probabilité p_i que la retenue se propage jusqu'au i-ème chiffre vérifie $p_{i+1} = p_i/B$ puisqu'elle ne se propage plus loin que lorsque le i-ème chiffre vaut $B-1$, donc une fois sur B. Il en résulte que la probabilité pour que la retenue s'arrête au i-ème chiffre pour $1 \leq i < nl$ est $\frac{B-1}{2B^i}$. La longueur moyenne de propagation de la retenue, qui est aussi le nombre moyen d'additions effectuées, vaut donc pour $nl \geq 1$:

$$\begin{aligned}
\bar{l} &= \frac{B-1}{2B}1 + \frac{B-1}{2B^2}2 + \cdots + \frac{B-1}{2B^{nl-1}}(nl-1) + \frac{B}{2B^{nl}}nl \\
&< \sum_{i=1}^{+\infty} \frac{B-1}{2B^i}i \\
&= \frac{B-1}{2B}\sum_{i=1}^{+\infty}\frac{i}{B^{i-1}} \\
&= \frac{B-1}{2B}(\sum_{i=1}^{+\infty}\frac{1}{B^{i-1}})^2
\end{aligned}$$

$$= \frac{1}{2}\frac{B}{B-1}$$

La première ligne donne aussi $\bar{l} \geq \frac{1}{2}$. Le coût de la propagation est donc $\frac{1}{2}t_+$ avec une approximation de l'ordre de $1/B$. Quand $nl = 0$, cette fonction ne fait rien et renvoie r, donc son coût est nul.

N+N+b(n,nd,nl,m,md,ml,r) Ajoute à $n[nd]\ldots n[nd+nl-1]$ la valeur de $m[md]\ldots m[md+ml-1]$ (cela suppose $nl \geq ml$), puis propage la retenue sur les $nl-ml$ chiffres restant. Enfin, renvoie la retenue sortante. Son coût est donc $ml\, t_+$ si $nl = ml$, $(ml+1/2)\, t_+$ sinon.

N−N+b−1(n,nd,nl,m,md,ml,r) Retranche la valeur de l'entier $m[md]\ldots m[md+ml-1]$ à $n[nd]\ldots n[nd+nl-1]$ (cela suppose $nl \geq ml$), puis propage l'emprunt sur les $nl-ml$ chiffres restant (un emprunt vaut 0 s'il faut retrancher 1 au chiffre suivant, et 1 sinon). Enfin, renvoie l'emprunt sortant. Son coût est donc $ml\, t_-$ si $nl = ml$, $(ml+1/2)\, t_-$ sinon, pour la même raison que l'addition.

N+N×C(p,pd,pl,n,nd,nl,C,cd) Ajoute à $p[pd]\ldots p[pd+nl]$ la valeur de $n[nd]\ldots n[nd+nl-1]$ multiplié par $C[cd]$, (cela suppose $pl \geq nl+1$), puis propage la retenue sur les $pl-nl-1$ chiffres restant. Renvoie la retenue finale. Comme nous l'avons déjà montré au chapitre précédent, son coût est $nl\, t_\times + t_+$.

2.1 Description de l'implémentation

Décrivons maintenant l'implémentation de l'algorithme de Karatsuba que nous avons imaginée pour accumuler dans $P = val < p[pd], \ldots, p[pd+2l-1] >$ la valeur du produit NM, avec $N = val < n[nd], \ldots, n[nd+l-1] >$ et $M = val < m[md], \ldots, m[md+l-1] >$. N et M ont donc le même nombre de chiffres l. Nous avons aussi besoin d'un tableau auxiliaire de calcul t, de $2l$ chiffres.

Fonction Karatsuba(p,pd,n,nd,m,md,l,t,td)

- si $l < N_2$, utiliser l'algorithme naïf
- si l est impair, exécuter

  Karatsuba(p,pd+2,n,nd+1,m,md+1,l-1,t,td);

  N+N×C(p,pd,2l,n,nd,l,m,md);

  N+N×C(p,pd+1,2l-1,m,md+1,l-1,n,nd);

- sinon, $l = 2l'$. Posons alors $\& = B^{l'}$, $N = N_0 + \&N_1$, $M = M_0 + \&M_1$, et $P = P_0 + \&P_1 + \&^2 P_2 + \&^3 P_3$.

  La disposition initiale est alors:

p				n		m		t			
P_0	P_1	P_2	P_3	N_0	N_1	M_0	M_1	?	?	?	?

  N→N(n,nd+l',l',t,td);

  N→N(m,md+l',l',t,td+l');

p				n		m		t			
P_0	P_1	P_2	P_3	N_0	N_1	M_0	M_1	N_1	M_1	?	?

cn := N+N+b(t,td,l',n,nd,l',0);

cm := N+N+b(t,td+l',l',m,md,l',0);

p				n		m		t			
P_0	P_1	P_2	P_3	N_0	N_1	M_0	M_1	S_n	S_m	?	?

$N_0 + N_1 = S_n + c_n\&$ et $M_0 + M_1 = S_m + c_m\&$

if cn=1 then r3 := cm + N+N+b(p,pd+2l',l',t,td+l',l',0);

if cm=1 then r3 := r3 + N+N+b(p,pd+2l',l',t,td,l',0);

p				n		m		t			
P_0	P_1	P_4	P_3	N_0	N_1	M_0	M_1	S_n	S_m	?	?

$P_4 + r_3\& = P_2 + c_n S_m + c_m S_n + c_n c_m \&$

r3 := r3 + Karatsuba(p,pd+l',t,td,t,td+l',l',t,td+2l');

p			n		m		t			
P_0	P_5	P_3	N_0	N_1	M_0	M_1	S_n	S_m	?	?

$P_5 + r_3\&^2 = P_1 + \&P_2 + (N_0 + N_1)(M_0 + M_1)$

razN(t,td,l);

Karatsuba(t,td,n,nd,m,md,l',t,td+2l');

p			n		m		t		
P_0	P_5	P_3	N_0	N_1	M_0	M_1	N_0M_0	?	?

r3 := r3 + N−N+b−1(p,pd+l',l,t,td,l,1) - 1;

r3 := r3 + N+N+b(p,pd,l+l',t,td,l,0);

p		n		m		t		
P_6	P_3	N_0	N_1	M_0	M_1	N_0M_0	?	?

$P_6 + r_3\&^3 = P_0 + P_1\& + P_2\&^2 + N_0M_0 + (N_0M_1 + N_1M_0 + N_1M_1)\&$

razN(t,td,l);

Karatsuba(t,td,n,nd+l',m,md+l',l',t,td+2l');

p		n		m		t		
P_6	P_3	N_0	N_1	M_0	M_1	N_1M_1	?	?

r3 := r3 + N−N+b−1(p,pd+l',l,t,td,l,1) - 1;

r4 := N+N+b(p,pd+l,l,t,td,l,0);

p	n		m		t		
P_7	N_0	N_1	M_0	M_1	N_1M_1	?	?

$P_7 + r_3\&^3 + r_4\&^4 = P + N_0M_0 + (N_0M_1 + N_1M_0)\& + N_1M_1\&^2$

```
while r3 > 0 do begin
        r4 := r4 + N+b(p,pd+l+l',l',1);
        r3 := r3 - 1;
end;
```

p	n		m		t		
P_8	N_0	N_1	M_0	M_1	N_1M_1	?	?

$P_8 + r_4\&^4 = P + NM$

renvoyer r_4 qui vaut 0 ou 1.

2.2 Evaluation du coût théorique

Nous allons calculer en fonction de $t_{N\to N}$, t_{razN}, t_+, t_-, $t_\times$ la valeur de la constante λ que nous avions introduite au chapitre précédent. Cela nous permettra de calculer la valeur théorique de N_2.

Lorsque $l = 2l'$, nous effectuons:

- 2 transferts de l' chiffres
- 2 additions sur l' chiffres
- 0, 1 ou 2 additions sur l' chiffres suivant la valeur de c_n et c_m: comptabilisons une addition sur l' chiffres en moyenne
- 3 multiplications de l' chiffres par l' chiffres
- 2 mises-à-zéro sur l chiffres
- 2 soustractions sur l chiffres
- 2 additions sur l chiffres

La valeur de λ est par conséquent:

$$\lambda = 7t_+ + 4t_- + 4t_{razN} + 2t_{N\to N}$$

et d'après 1.4

$$\begin{aligned} N_2 &= \lceil 2\frac{\lambda + t_+}{t_\times} \rceil \\ &= \lceil 4\frac{4t_+ + 2t_- + 2t_{razN} + t_{N\to N}}{t_\times} \rceil \end{aligned} \tag{2.1}$$

2.3 Exemples

Les tableaux ci-dessous indiquent pour chaque instruction LLM4 utilisée, le nombre de fois où l'instruction machine correspondante a été utilisée pour multiplier deux nombres de N chiffres. Par exemple, le 4460 dans la ligne N+N+b pour $N = 160$ et $N_2 = 10$ indique que 4460 additions d'un chiffre ont été réalisées.

	$N_2 = 10$	$N_2 = 20$	$N_2 = 40$
N→N	100	40	0
razN	200	80	0
N+N+b	320	120	0
N−N+b−1	200	80	0
N+N×C	900	1200	1600

Table 2.1: 40 chiffres × 40 chiffres

	$N_2 = 10$	$N_2 = 20$	$N_2 = 80$
N→N	380	200	0
razN	760	400	0
N+N+b	1280	640	0
N−N+b−1	760	400	0
N+N×C	2700	3600	6400

Table 2.2: 80 chiffres × 80 chiffres

	$N_2 = 10$	$N_2 = 20$	$N_2 = 160$
N→N	1300	76	0
razN	2600	1520	0
N+N+b	4460	2520	0
N−N+b−1	2600	1520	0
N+N×C	8100	10800	25600

Table 2.3: 160 chiffres × 160 chiffres

Chapitre 3

Implémentation de la méthode de Schönhage

3.1 Gestion de la mémoire

La nature semble avoir des lois qu'on ne peut transgresser. Quand on essaie d'obtenir plus d'un côté, on perd de l'autre. Ainsi, il semble que plus un algorithme de multiplication est rapide, plus sa mise en oeuvre requiert d'espace mémoire. Dans le cas d'un circuit, la mémoire est en fait de la surface. F. Preparata et J. Vuillemin ont proposé des circuits optimaux pour ce compromis espace-temps dans [1]. La multiplication-accumulation naïve se fait en place (cf algorithme naïf). Il nous a fallu une mémoire tampon de taille égale au produit des opérandes pour notre implémentation de la méthode de Karatsuba (cf section 2.1). Pour implémenter celle de Schönhage, nous aurons besoin d'encore plus de mémoire auxiliaire, comme nous allons le voir tout de suite.

Si $N = 2^k M$, avec $k \geq 1$, la multiplication de deux entiers de $N/2$ bits se fait par une multiplication modulaire sur N bits. Celle-ci se ramène à 2^k multiplications modulaires sur $2M + k$ bits. Supposons que chacune de ces 'petites' multiplications se fasse en place, c'est-à-dire que le produit sera stocké à la place d'un des opérandes. Chacune nécessite donc une place mémoire de $2(2M+k)$ bits, et la place totale nécessaire est (cf la remarque ci-dessous):

$$2^k(2(2M + k)) = 4N + k2^{k+1}$$

Dans la pratique, M est choisi voisin de $\sqrt{N}$, donc $k \sim \frac{1}{2}\log_2 N$. Le terme $k2^{k+1}$, qui vaut alors $\sqrt{N}\log_2 N$, est par conséquent négligeable devant le terme $4N$. Pour multiplier deux entiers de $N/2$ bits, il nous faut une mémoire auxiliaire d'au moins $4N$ bits. Par suite, pour multiplier deux entiers de N bits, il nous faut $8N$ bits en plus, soit quatre fois ce qu'il nous fallait pour la méthode de Karatsuba !

Remarque : on pourrait penser à calculer les $\hat{a}'_\mu$ et $\hat{b}'_\mu$ l'un après l'autre, mais cela ne pourrait pas se faire par la FFT (qui les calcule tous en même temps). Le coût de ces calculs serait en $O(N^2)$, et non plus en $O(N \log N)$. Au stade $O(N^2)$, autant utiliser l'algorithme naïf !

3.2 Description de l'implémentation

Celle-ci n'a été faite que dans le cas où les opérandes sont de même taille nl. Les paramètres sont donc n, nd, nl pour le premier entier, m, md, nl pour le second et p, pd, pl pour l'emplacement où doit être accumulé le produit , avec $pl \geq 2nl$ (cf le chapitre précédent pour la signification de ces variables). Désignons par L le nombre de bits que contient un chiffre ($L = 32$ ou $L = 16$ dans notre cas). Voici la suite d'opérations que nous effectuons:

Algorithme Schönhage (p,pd,pl,n,nd,nl,m,md)

- trouver $N = nL$ tel que $n \geq 2nl$ et $N = 2^k M$ avec k convenable (cf la section 3.2.5)
- déterminer $N' = n'L$ vérifiant $N' \geq 2M + k + 3$ (l'apparition de la constante 3 est expliquée à la fin de la section 3.2.2)
- réserver deux tampons de travail de $2^k(n' + 1)$ chiffres, chacun étant considéré comme un vecteur de 2^k résidus modulo $2^{N'} + 1$
- répartir les bits de $< n, nd, nl >$ dans le premier tampon, M bits dans chaque résidu ,idem pour $< m, md, nl >$ (étape de décomposition)
- multiplier le i-ème résidu de chaque tampon par θ, où $\theta = 2^M$ (étape de préparation)
- effectuer une FFT sur chaque vecteur de résidus, avec $\omega = \theta^2$
- effectuer les 2^k produits terme à terme modulo $2^{N'} + 1$, en stockant le résultat à la place de l'un des deux résidus
- effectuer une FFT sur le vecteur des produits modulaires
- réajuster les résidus calculés selon la formule 1.7 pour obtenir la FFT inverse (étape de réajustement)
- ajouter les 2^k produits de convolution cyclique négative à $< p, pd, pl >$, chacun à l'emplacement donné par 1.14 (étape de repli)
- renvoyer la retenue sortante

3.2.1 Implémentation de la FFT

Celle-ci prend comme paramètres:

1. K, le nombre d'éléments du vecteur à transformer
2. le nom n du tableau dans lequel sont situés ces résidus
3. le nombre de chiffres nl sur lequel est stocké chaque résidu
4. une liste lnd de K indices, le i-ème étant l'indice de début du i-ème élément du vecteur ($0 \leq i < K$)
5. l'entier w tel que $\omega = 2^w$
6. l'indice td de début de zone disponible du tampon

Algorithme FFT (K,n,nl,lnd,w,td)

- si $K = 2$, alors $\omega \equiv -1 \bmod F_{N'}$. Soit A_0 (resp. A_1) l'emplacement du tableau pointé par l'élément d'ordre 0 (resp. 1) de *lnd*, et T les *nl* premiers chiffres disponibles du tampon (les majuscules désignent des zones mémoire, les minuscules des valeurs). Il faut passer de la disposition:

A_0	A_1	T
a_0	a_1	?

à

A_0	A_1	T
$a_0 + a_1$	$a_0 - a_1$	XXX

XXX : valeur quelconque
? : valeur indéterminée

  - recopier A_0 dans T:

A_0	A_1	T
a_0	a_1	a_0

  - ajouter A_1 à A_0:

A_0	A_1	T
$a_0 + a_1$	a_1	a_0

  - remplacer A_1 par son opposé:

A_0	A_1	T
$a_0 + a_1$	$-a_1$	a_0

  - ajouter T à A_1:

A_0	A_1	T
$a_0 + a_1$	$a_0 - a_1$	a_0

- sinon:
  - calculer *lndp* (resp. *lndi*), la liste des éléments d'ordre pair (resp. impair) de *lnd*
  - FFT (K/2,n,nl,lndp,2w,td) qui renvoie la liste $lp = (nd_0, nd_2, \ldots, nd_{K-2})$ des indices de début des $K/2$ résidus transformés d'indice pair
  - FFT (K/2,n,nl,lndi,2w,td) qui renvoie la liste $li = (nd_1, nd_3, \ldots, nd_{K-1})$ des indices de début des $K/2$ résidus transformés d'indice impair
  - pour $0 \leq i < K/2$, soit A_j l'emplacement pointé par n_j, A'_i (resp. $A'_{K/2+i}$) l'emplacement où l'on va stocker $\hat{a}_i$ (resp. $\hat{a}_{K/2+i}$). D'après 1.10, il faut passer de la disposition

A_{2i}	A_{2i+1}	A'_i	$A'_{K/2+i}$
a_{2i}	a_{2i+1}	?	?

à

A_{2i}	A_{2i+1}	A'_i	$A'_{K/2+i}$
XXX	XXX	$a_{2i} + \omega^i a_{2i+1}$	$a_{2i} - \omega^i a_{2i+1}$

mais on écraserait alors les résidus en A'_i et $A'_{K/2+i}$. La solution est d'effectuer ces transformations en place dans A_{2i} et A_{2i+1}:

    * recopier A_{2i} dans T:

A_{2i}	A_{2i+1}	T
a_{2i}	a_{2i+1}	a_{2i}

    * multiplier A_{2i+1} par ω^i:

A_{2i}	A_{2i+1}	T
a_{2i}	$\omega^i a_{2i+1}$	a_{2i}

    * ajouter A_{2i+1} à A_{2i}:

A_{2i}	A_{2i+1}	T
$a_{2i} + \omega^i a_{2i+1}$	$\omega^i a_{2i+1}$	a_{2i}

    * remplacer A_{2i+1} par son opposé:

A_{2i}	A_{2i+1}	T
$a_{2i} + \omega^i a_{2i+1}$	$-\omega^i a_{2i+1}$	a_{2i}

    * ajouter T à A_{2i+1}:

A_{2i}	A_{2i+1}	T
$a_{2i} + \omega^i a_{2i+1}$	$a_{2i} - \omega^i a_{2i+1}$	a_{2i}

  - on a ainsi bouleversé l'ordre des transformés: $\hat{a}_i$ se trouve en A_{2i} et $\hat{a}_{K/2+i}$ se trouve en A_{2i+1}. La liste ordonnée des emplacements où se situent les résidus transformés est donc $(A_0, A_2, \ldots, A_{K-2}, A_1, A_3, \ldots, A_{K-1})$. La liste des indices qu'il faut retourner est par conséquent la simple concaténation de lp et li.

3.2.2 Représentation des résidus

Comment représenter en mémoire un résidu A modulo $F_{N'}$? Ce sera par un entier $A' \equiv A \bmod F_{N'}$. N' étant de la forme $n'L$, il nous faudra au moins $n'+1$ chiffres. Nous avons plusieurs possibilités pour la normalisation de A':

- $0 \leq A' \leq 2^{N'}$: le chiffre de poids fort vaut 1 si $A' = 2^{N'}$, 0 sinon
- $0 \leq A' \leq 2^{N'+1} - 1$: le chiffre de poids fort vaut 0 ou 1
- $0 \leq A' \leq 2^{N'+L} - 1$: le chiffre de poids fort varie entre 0 et $B - 1$

Nous avons rejeté tout de suite la troisième solution car elle conduit à une retenue lors d'une addition, qui complique la procédure de renormalisation. La première solution a l'avantage de l'unicité de la représentation, mais la normalisation, après une addition par exemple, n'est pas immédiate. Nous avons donc choisi la seconde solution, imitant en cela A. Schönhage. Nous dirons que A' est une représentation semi- normalisée de A (la représentation normalisée étant celle donnée par la première solution). Ceci modifie l'intervalle de variation des c_μ, car la valeur maximale des a_i et b_j n'est plus 2^M mais $2^{M+1} - 1$:

$$-(n - (\mu + 1))2^{2M+2} \leq c_m u \leq (\mu + 1)2^{2M+2}$$

De plus, si l'on veut éviter que le calcul de c_μ à partir de d_μ conduise à des comparaisons par rapport à des nombres autres que des puissances de 2, il suffit de remarquer que:

$$-n2^{2M+2} \leq c_m u \leq n2^{2M+2}$$

La longueur de l'intervalle est devenue $n2^{2M+3} = 2^{2M+k+3}$, d'où le choix de $N' \geq 2M + k + 3$.

3.2.3 Fonctions modulaires

Nous avons besoin de cinq fonctions sur des résidus. Dans l'évaluation de leurs coûts respectifs, nous négligerons les termes constants.

N+N_ mod(n,nd,nl,m,md) Le paramètre nl est le nombre de chiffres sur lesquels chaque résidu est stocké, donc vaut $n' + 1$. Cette fonction ajoute le résidu $< m, md, nl >$ à $< n, nd, nl >$ et rend un résultat semi-normalisé. Pour cela, on effectue tout d'abord une addition entière N+N+b(n,nd,nl,m,md,ml,0). Puis, soit C le chiffre de poids fort obtenu:

- si $C \leq 1$, le résultat est déjà semi-normalisé
- sinon, on retranche $C - 1$ aux deux chiffres extrêmes. Cela revient à retrancher $(C - 1)(1 + 2^{N'})$, donc ne modifie pas la valeur du résultat modulo $F_{N'}$. De plus, on est assuré d'aboutir à une forme semi-normalisée. Le coût de cette fonction est par conséquent $nl\, t_+$.

−N_ mod(m,md,ml) change $M =< m, md, ml >$ en son opposé modulo $F_{N'}$ et rend un résultat semi-normalisé. Soit $M_0 =< m, md, ml - 1 >$ et C le chiffre de poids fort: $M = M_0 + C2^{N'}$ et $-M = -M_0 - C2^{N'} \equiv -M_0 + C \bmod F_{N'}$. On complémente tout d'abord les $ml - 1$ premiers chiffres, ce qui conduit à $2^{N'} - M_0 - 1$. Il suffit ensuite d'ajouter $C + 2$. Le résultat ne peut excéder $2^{N'} + C + 1$, donc est semi-normalisé. Le coût est $ml\, t_{not}$, où t_{not} est le temps d'un NOT logique sur un chiffre.

N×2ˆd_ mod(m,md,ml,d,td) remplace $M =< m, md, ml >$ par $2^d M$ modulo $F_{N'}$, semi-normalisé. On doit avoir $0 \leq d < N'$, et td pointe sur une zone mémoire auxiliaire. Pour effectuer cette opération, nous écrivons tout d'abord $d = Ld_1 + d_0$ ($0 \leq d_1 < n'$ et $0 \leq d_0 < L$), puis:

- multiplier chaque chiffre par 2^{d_0} en reportant les d_0 chiffres sortant dans le suivant (si M est semi- normalisé, il ne sort que des bits nuls du chiffre de poids fort)
- recopier les $d_1 + 1$ chiffres de poids fort dans le tampon
- décaler les $nl - d_1 - 1$ chiffres de poids faible de d_1 chiffres vers les poids forts
- mettre à zéro les d_1 chiffres de poids faible
- mettre à 1 les deux chiffres extrèmes (qui sont virtuellement à zéro) pour éviter un emprunt sortant dans la soustraction qui suit
- venir retrancher les $d_1 + 1$ chiffres sauvegardés dans le tampon aux poids faibles

Le coût est $n' t_{sh} + n' t_{N\to N} + d_1 t_{razN} + d_1 t_-$, et comme d_1 varie entre 0 et $n' - 1$, le coût moyen s'écrit $n'(t_{sh} + t_{N\to N}) + \frac{n'}{2}(t_{razN} + t_-)$

N/2ˆd_ mod(m,md,ml,d,td) remplace $M =< m, md, ml >$ par $M/2^d$ modulo $F_{N'}$, semi-normalisé. On opère de façon analogue à la multiplication par 2^d, sauf que d est décomposé en $d_1 L - d_0$ avec $0 \leq d_1 \leq N'$ et $0 \leq d_0 < L$. Le coût moyen est aussi $n'(t_{sh} + t_{N\to N}) + \frac{n'}{2}(t_{razN} + t_-)$

N×N_ mod(p,pd,m,md,ml,td) remplace $P =< p, pd, ml >$ par PM modulo $F_{N'}$, semi-normalisé. On opère ainsi:

- mettre à zéro une zone de $2\,ml$ chiffres dans le tampon
- y accumuler le produit entier PM. Si P et M étaient semi-normalisés, le chiffre d'indice $2\,ml - 1$ est forcément nul et le précédent vaut au plus 3.
- retrancher les chiffres d'indice $ml-1, \ldots, 2\,ml-3$ à ceux d'indice $0, \ldots, ml-2$, et ajouter le chiffre d'indice $2\,ml - 1$, en prenant les précautions d'usage de semi-normalisation.
- transférer le résultat dans $P =< p, pd, ml >$

Le coût de cette opération est par suite $2n' t_{razN} + C_\times(n') + n' t_- + n' t_{N\to N}$.

3.2.4 Evaluation des coûts

<u>Coût de la FFT</u>

Soit $C_{fft}(K, n')$ le coût de la FFT d'un vecteur de K éléments, chacun étant un résidu stocké sur n' chiffres. Pour $K = 2$, il vient immédiatement:

$$C_{fft}(2, n') = 2C_+(n') + C_{not}(n') + C_{N\to N}(n')$$

et pour $K > 2$:

$$C_{fft}(K, n') = 2C_{fft}(K/2, n') + \frac{K}{2}(2C_+(n') + C_{sh}(n') + C_{not}(n') + C_{N\to N}(n'))$$

d'où on déduit

$$C_{fft}(2^k, n') = k2^{k-1}(2C_+(n') + C_{not}(n') + C_{N\to N}(n')) + (k-1)2^{k-1}C_{sh}(n')$$

En remplaçant C_+, C_{not}, $C_{N\to N}$ et C_{sh} par leurs valeurs calculées ci-dessus, il vient:

$$C_{fft}(2^k, n') = 2^{k-2}n'(4kt_+ + (k-1)t_- + (k-1)t_{razN} + 2kt_{not} + 2(k-1)t_{sh} + (4k-2)t_{N\to N})$$

Coût de l'algorithme Schönhage

Il découle de la description de l'algorithme que:

$$C_{schon}(n) = 2C_{decomp}(n) + 2C_{prep}(n) + 3C_{fft}(2^k, n') + 2^k C_{\times mod}(n') + C_{reaj}(n) + C_{repli}(n)$$

Remarquons tout d'abord que $C_{reaj}(n) = C_{prep}(n)$; le réajustement consiste à diviser chaque résidu par une puissance de 2, la préparation à les multiplier par une puissance de 2, et ces instructions ont même coût.

$$C_{decomp}(n) = 2^k(\frac{n'}{2}t_{N\to N} + \frac{n'}{2}t_{razN})$$

En effet, $N' \geq 2M + k + 3 \simeq 2M$. L'étape de décomposition consiste à mettre dans chaque résidu M bits des N de départ, puis à compléter par des zéros. Ces M bits remplissent donc environ la moitié du résidu.

$$C_{prep}(n) = 2^k C_{sh}(n') = 2^{k-1}n'(2t_{sh} + 2t_{N\to N} + t_{razN} + t_-)$$

$$C_{\times mod}(n') = 2n't_{razN} + n't_- + n't_{N\to N} + C_\times(n')$$

$$C_{repli}(n) = 2^k(n't_+)$$

Après calcul, on obtient:

$$\begin{aligned} C_{schon}(n) &= 2^{k-2}n'((12k+4)t_+ + (12k+14)t_{N\to N} + (3k+15)t_{razN} + (6k+6)t_{sh} + (3k+7)t_- + 6kt_{not}) \\ &+ 2^k C_\times(n') \end{aligned}$$

Il faut maintenant nous débarrasser de k ou de n' en écrivant que $N' \simeq 2M = N/2^{k-1}$ d'où $n' \simeq n/2^{k-1}$:

$$\begin{aligned} C_{schon}(n) &= \frac{n}{2}((12k+4)t_+ + (12k+14)t_{N\to N} + (3k+15)t_{razN} + (6k+6)t_{sh} + (3k+7)t_- + 6kt_{not}) \\ &+ 2^k C_\times(n/2^{k-1}) \end{aligned} \tag{3.1}$$

Quelles sont les valeurs intéressantes de k ? Pour simplifier le problème de départ, il faut que $n/2^{k-1}$ soit inférieur à n, donc que k soit supérieur ou égal à 2. Mais la valeur $k = 2$ ne convient pas non plus car on a alors $2^k = 4$ produits de taille $n/2$ à effectuer. Il vaut mieux dans ce cas utiliser directement la multiplication naïve. On a les possibilités suivantes pour réduire la multiplication modulaire de deux résidus de n chiffres (le coût indiqué correspond à la méthode naïve pour les 'petits' produits):

8 produits entiers de n/4 chiffres, coût total :	$n^2/2$
16 produits entiers de n/8 chiffres, coût total :	$n^2/4$
32 produits entiers de n/16 chiffres, coût total :	$n^2/8$
$\vdots$	$\vdots$

Lorsque k augmente, le coût des produits terme à terme diminue, mais la complexité de la FFT augmente en nk, comme le montre le premier terme de 3.1.

3.2.5 Quand et comment utiliser Schönhage ?

Nous allons d'abord calculer le point d'équilibre avec la multiplication naïve pour le produit modulaire, puis nous en déduirons celui correspondant au produit entier.

Remplaçons $C_\times$ par C_n dans 3.1, ce qui donne:

$$\begin{aligned} C_{schon}(n) &= \frac{n}{2}((12k+5)t_+ + (12k+14)t_{N\to N} + (3k+15)t_{razN} + (6k+6)t_{sh} + (3k+7)t_- + 6kt_{not}) \\ &+ \frac{n^2}{2^{k-2}}t_\times \end{aligned}$$

à comparer avec le coût du même produit modulaire par la méthode naïve, soit

$$\begin{aligned} C_{\times mod}(n) &= 2nt_{razN} + nt_- + nt_{N\to N} + C_n(n) \\ &= 2nt_{razN} + nt_- + nt_{N\to N} + n^2 t_\times + nt_+ \end{aligned}$$

L'égalité a lieu lorsque:

$$\frac{n}{2}((12k+3)t_+ + (12k+12)t_{N\to N} + (3k+11)t_{razN} + (6k+6)t_{sh} + (3k+5)t_- + 6kt_{not}) = n^2(1-\frac{1}{2^{k-2}})t_\times$$

d'où

$$n = \frac{(12k+3)t_+ + (12k+12)t_{N\to N} + (3k+11)t_{razN} + (6k+6)t_{sh} + (3k+5)t_- + 6kt_{not}}{2(1-\frac{1}{2^{k-2}})t_\times}$$

Assimilons tous les temps du numérateur à t_+:

$$n = \frac{42k+37}{2(1-\frac{1}{2^{k-2}})}\frac{t_+}{t_\times}$$

ce qui donne pour $k = 3$:

$$n = 163\frac{t_+}{t_\times}$$

et pour $k = 4$:

$$n = \frac{820}{3}\frac{t_+}{t_\times}$$

La constante grandit encore lorsque k augmente, donc le point d'équilibre théorique cherché est:

$$N_{s\,mod} = \frac{39t_+ + 48t_{N\to N} + 20t_{razN} + 24t_{sh} + 14t_- + 18t_{not}}{t_\times}$$

Pour la multiplication entière, le point d'équilibre est plus élevé puisqu'il faut maintenant que Schönhage($2n$) soit aussi rapide que Naïf(n) ! Pour que cela soit possible, il faut déjà que k soit supérieur ou égal à 5.

Après calcul, il vient:

$$N_{s\,ent} = \frac{128t_+ + 148t_{N\to N} + 60t_{razN} + 72t_{sh} + 44t_- + 60t_{not}}{t_\times} \tag{3.2}$$

et si l'on identifie les temps du numérateur avec t_+:

$$N_{s\,ent} = 512\frac{t_+}{t_\times}$$

Chapitre 4

Résultats sur VAX 785, MC 68020 et VAX 8600

L'écriture en langage machine des instructions LLM4 a été faite par Bernard Serpette, pour deux types de machine : un MC 68020 et un VAX (785). Son travail n'a pas été facile : pour le VAX, il a fallu construire un produit 32 bits non signé à partir d'une routine de multiplication signée. Mais les gains réalisés justifient la tâche effectuée. J'ai alors compilé le code source VAX sur un 8600, ce qui permettra de comparer les résultats sur trois machines au lieu de deux.
Remarque importante : la convention que j'ai adoptée pour mes 'benchmark' est la suivante : j'exécute au minimum 5 fois la fonction à mesurer, et je retiens le plus petit des temps indiqués par la fonction Unix (ou Le_Lisp) 'time'. Voici les temps des instructions élémentaires (en microsecondes), qui vont nous permettre de calculer les points théoriques d'équilibre.

	VAX 785	MC 68020	VAX 8600
t_{razN}	1.6	0.8	0.7
$t_{N \to N}$	2.0	1.0	1.0
t_{not}	2.1	0.9	0.9
t_{+}	4.0	2.3	1.9
t_{-}	4.4	2.4	2.2
t_{sh}	4.6	2.2	1.9
$t_{\times}$	12.3	5.4	4.0

4.1 Avant

Où se plaçait Le_Lisp par rapport aux autres logiciels travaillant en précision arbitraire ? Nous allons le savoir grâce au *benchmark 1*. Tous les temps sont désormais en secondes.

Pour obtenir des grands nombres rapidement, il suffit d'élever au carré des entiers déjà relativement grands. Partons de 3 : nous calculons $3^2 = 9$, puis $9^2 = 3^4 = 81$, et ainsi de suite. Le tableau 4.1 reproduit les temps de calcul et d'affichage du carré de 3^{2^n}, où n varie entre 7 et 15, sur un VAX 785. Le temps de la colonne *print* est la somme du temps éventuel de conversion de binaire en décimal et du temps d'impression sur l'écran. Il est ainsi évident que **Maple** et **Bc** utilisent une représentation décimale.

	Franz Lisp		**Maple 4.1**		**Le_Lisp 1/11/86**		**Bc**		
n	calcul	print	calcul	print	calcul	print	calcul	print	$\log_{10} 3^{2^n}$
7					0.00	0.02	0.0	0.0	62
8					0.02	0.10	0.1	0.0	123
9	0.00	0.03	0.03	0.0	0.09	0.38	0.6	0.1	245
10	0.03	0.12	0.11	0.01	0.38	1.54	2.2	0.2	489
11	0.15	0.45	0.52	0.07	1.52	5.60	8.8	0.4	978
12	0.58	1.73	1.91	0.12	6.0	27.0	36.3	0.8	1955
13	2.30	6.73	8.33	0.30					3909
14	9.42	26.8	34.27	0.64					7818
15	40.6	104.8	144.5	1.34					15635

Table 4.1: Benchmark 1 sur VAX 785 *avant*

4.2 Pendant

4.2.1 Karatsuba

Voici les valeurs mesurées des points d'équilibre, ainsi que les valeurs théoriques calculées par 2.1 avec les temps du tableau ci-dessus.

	VAX 785	MC 68020	VAX 8600
$N_{2\,theo}$	10	13	15
$N_{2\,mes}$	25	35	45

La mesure du point d'équilibre consiste à comparer les temps de multiplication lorsque l'on fait varier la taille critique N_2 à partir de laquelle on utilise Karatsuba. Le meilleur choix correspond au minimum de la courbe, mais celle-ci est perturbée par les imprécisions de la fonction 'time', comme on peut le remarquer sur la figure ci-dessous. En abscisse se trouve N_2, et en ordonnée le temps de calcul du carré d'un nombre de 1000 chiffres sur VAX 8600.

4.2.2 Schönhage

La formule 3.2 nous permet de calculer les points d'équilibre théoriques, pour le produit entier, entre la méthode naïve et celle de Schönhage:

	VAX 785	MC 68020	VAX 8600
N_{stheo}	127	150	181
N_{smes}	270		

Il ne manque plus qu'un tableau comparatif des trois méthodes (en fait, il y en a un nombre illimité, puisque pour Schönhage, k peut prendre toutes les valeurs de 5 à $+\infty$). A partir de $N = 700$ chiffres, on a utilisé *complice++*, car *complice* ne suffisait plus.

N	naïf	Karatsuba	Schönhage		
			$k = 5$	$k = 6$	$k = 7$
100	0.11	0.08	0.29	0.47	
200	0.47	0.26	0.56	0.73	1.10
300	1.06	0.50	0.92	0.94	1.36
400	1.96	0.79	1.46	1.25	1.68
500	2.92	1.10	2.06	1.52	1.68
600	4.25	1.50	2.81	2.01	2.13
700	6.64	2.02	3.83	2.72	2.54
800	7.79	2.37	4.57	3.21	2.96
884	8.98	2.83	5.42	3.42	2.96
984	11.1	3.32	6.59	4.09	
1100	14.4	3.79			
2000	46.9	9.96			
4095	197	33.3			

Pour $k = 7$, il n'est pas possible d'effectuer un produit avec $N > 884$ à cause d'une limitation de taille des vecteurs Le_Lisp. Même chose pour $k = 6$ et $N > 984$, ainsi que pour $k = 5$ et $N > 980$. Pour Karatsuba, on ne peut avoir $N > 4095$. Le temps 197 pour $N = 4095$ est une estimation.

4.3 Après

Le tableau 4.1 nous a montré que le programme de référence, celui qu'il faut battre, est Franz Lisp. Le tableau ci-dessous permet de mesurer les gains réalisés:

1. par le passage de 16 bits avec *ratio* à 32 bits avec *llm4as*
2. par l'utilisation de la méthode de Karatsuba

	Franz Lisp		**Le_Lisp 1/11/86**		**Le_Lisp (llm4as)**		**Le_Lisp (llm4as+kara)**		
n	calcul	print	calcul	print	calcul	print	calcul	print	$\log_{10} 3^{2^n}$
7			0.00	0.02	0.00	0.02	0.00	0.02	62
8			0.02	0.10	0.00	0.04	0.00	0.04	123
9	0.00	0.03	0.09	0.38	0.01	0.09	0.01	0.09	245
10	0.03	0.12	0.38	1.54	0.03	0.25	0.02	0.25	489
11	0.15	0.45	1.52	5.60	0.13	0.67	0.09	0.67	978
12	0.58	1.73	6.0	27.0	0.50	1.96	0.27	1.96	1955
13	2.30	6.73			1.94	6.79	0.83	6.79	3909
14	9.42	26.8			7.59	25.07	2.52	25.07	7818
15	40.6	104.8			36.29	98.08	7.66	98.08	15635

Il ressort de ce tableau que:

- on a gagné un facteur 12 sur le calcul en passant en 32 bits et en écrivant les instructions de bas niveau directement en langage machine
- on a gagné sur le *print* un facteur variant de 2 pour des nombres d'une centaine de chiffres décimaux à plus de 13 pour 2000 chiffres !
- Karatsuba permet de gagner un facteur 2 vers 2000 chiffres décimaux et près de 5 pour 15000 chiffres.

Bibliographie

[1] F.P. Preparata et J. Vuillemin *Area-time optimal VLSI networks for computing integer multiplication and discrete Fourier transform*

[2] D.E. Knuth *The Art of Computer Programming, Vol. 2 : Seminumerical Algorithms;* seconde édition, Addison-Wesley, 1981.

[3] Gilles BRASSARD, Sophie MONET et Daniel ZUFFELLATO *Algorithmes pour l'arithmétique des très grands entiers* Technique et Science Informatiques vol.5 numéro 2 (1986) p.89-102

[4] J.M. POLLARD *The Fast Fourier Transform in a Finite Field* Mathematics of Computation 25, p.365-374

[5] A. SCHÖNHAGE *Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients* Lecture Notes 144, p.3-15

[6] A. SCHÖNHAGE *Tapes versus Pointers, a study in implementing fast algorithms* Bulletin de l'EATCS, numéro 30, octobre 1986, p.23-32

[7] A. SCHÖNHAGE *Schnelle Multiplikation großer Zahlen* Computing 7 (1971), p. 281-292