Multiplication rapide par la méthode de Pollard
Application au calcul de constantes mathématiques

Fabrice Bellard

1. Généralités
2. Méthode de Pollard
   - Description et choix des paramètres
   - Etude de la vitesse
3. Extension au calcul de la racine carrée et de l'inverse – Calcul de π
4. Performances réelles

Généralités

(A_i) et (B_i) de 2^{k-1} chiffres en base M. Ajout de 0 jusqu'à une taille de 2^k chiffres. Puis on calcule:

$$\tilde{A} = FFT(A) \text{ et } \tilde{B} = FFT(B)$$

$$\widetilde{C_i} = \widetilde{A_i}.\widetilde{B_i}$$

$$C = FFT^{-1}(\tilde{C})$$

Ce qui donne:

$$C = A * B$$

Par propagation des retenues on obtient le produit $A.B$ en $O(n.\log(n))$ au lieu de $O(n^2)$ (si coûts fixes).

Mais C_i peut atteindre $2^{k-1}.(M-1)^2$
$\rightarrow$ arithmétique étendue.

Méthode de Pollard

FFT dans $\mathbf{Z}/N\mathbf{Z}$: calcul exact. Il faut :

$$N > 2^{k-1}.(M-1)^2$$

On cherche N tel qu'il existe des racines principales de l'unité d'ordre 2^k.
On pose alors :

$$N = p_1 p_2 \ldots p_r$$

avec

$$p_i = a_i.2^{k_i} + 1 \text{ premier}$$

et

$$(p_i) \text{ distincts 2 à 2}$$

On choisit k_i le plus grand possible avec $p_i \leq T$ où $T-1$ est le nombre maximum stockable dans un mot de la machine.

Si $k = min(k_i)$ alors il existe des racines principales d'ordre 2^k pour chacun des p_i, et il en existe aussi pour N (théorème chinois).

On mène r calculs de convolution indépendamment, et on rassemble les résultats à la fin.

Avantage: calcul en simple précision.

Choix des paramètres

- On désire faire des calculs sur plusieurs dizaines de millions de chiffres décimaux.

- $T = 2^{32}$ convient bien sur les machines actuelles, sachant que la multiplication donne un résultat sur un double mot ; on divise généralement un double mot par un mot.

- La conversion en base 10 prend un temps non négligeable, et elle est relativement compliquée si on la fait en $O(n \log(n))$

On choisit donc:

$$T = 2^{32}$$

$$M = 10^9 < T$$

Cela nous impose $k_{max} = 27$ avec les meilleurs nombres premiers:

$$p_1 = 29.2^{27} + 1 \ , \ p_2 = 13.2^{28} + 1 \ , \ p_3 = 3.2^{30} + 1$$

Donc $r = 3$ convient dans tous les cas pratiques, et on a bien:

$$2^{k_{max}-1}(M-1)^2 < N$$

Calcul de la FFT

Si $n' = 2^{k-1}$ et ω est une racine principale d'ordre 2^k de l'unité alors:

$$FFT(a)_{2j} = FFT((a_i + a_{n'+i})_i)$$

$$FFT(a)_{2j+1} = FFT((\omega_i(a_i - a_{n'+i}))_i)$$

D'où l'algorithme de calcul.

– On remarque la nécessité d'une permutation finale des éléments. Suivant la mémoire disponible, on précalcule une table de taille 2^k, ou de $2^{\lfloor k/2 \rfloor}$ caractérisant la permutation.

– On peut aussi précalculer les $(\omega_i)_{i=1\ldots 2^{k-1}-1}$.

– Parallélisation possible si le hardware s'y prête.

– Regrouper plusieurs passes ne diminue pas le nombre d'opérations.

– Pour une convolution :

$$\begin{aligned} N_{mulmod} &= (3k-2).2^{k-1} + 3 \\ N_{addmod} &= 3k.2^k \end{aligned}$$

Calcul de la FFT

Exemple pour $k = 3$.

	Etape 1	Etape 2	Etape 3
a_0	$b_0 = a_0 + a_4$	$c_0 = b_0 + b_2$	$\widetilde{a_0} = c_0 + c_1$
a_1	$b_1 = a_1 + a_5$	$c_1 = b_1 + b_3$	$\widetilde{a_4} = c_0 - c_1$
a_2	$b_2 = a_2 + a_6$	$c_2 = b_0 - b_2$	$\widetilde{a_2} = c_2 + c_3$
a_3	$b_3 = a_3 + a_7$	$c_3 = (b_1 - b_3)\omega^2$	$\widetilde{a_6} = c_2 - c_3$
a_4	$b_4 = a_0 - a_4$	$c_4 = b_4 + b_6$	$\widetilde{a_1} = c_4 + c_5$
a_5	$b_5 = (a_1 - a_5)\omega^1$	$c_5 = b_5 + b_7$	$\widetilde{a_5} = c_4 - c_5$
a_6	$b_6 = (a_2 - a_6)\omega^2$	$c_6 = b_4 - b_6$	$\widetilde{a_3} = c_6 + c_7$
a_7	$b_7 = (a_3 - a_7)\omega^3$	$c_7 = (b_5 - b_7)\omega^2$	$\widetilde{a_7} = c_6 - c_7$

- On peut utiliser le même tableau en entrée et en sortie en codant la permutation comme un ensemble de transpositions

- La transposition peut être effectuée après les produits $\widetilde{A_i}.\widetilde{B_i}$.

- La transformation réciproque s'obtient en permutant a_{2^k-i} et a_i pour $i = 1 \ldots 2^{k-1}$ et en multipliant chaque nombre par 2^{-k} avant d'appliquer la FFT.

Reste Chinois

On combine en fait cette étape avec la propagation des retenues pour plus d'efficacité et pour utiliser moins de mémoire.

– Pour calculer $(A_i) \bmod p_j$ et $(B_i) \bmod p_j$ il faut:

$$N_{div} = r.2^k$$

– A la fin, si $C_i = u_{ij} \bmod p_j$ (on omet i par la suite) et si on pose

$$C = v_r.p_{r-1}p_{r-2}\dots p_1 + \dots + v_2.p_1 + v_1$$

avec $v_i \in \{0 \dots (p_i - 1)\}$, alors:

$$\begin{array}{rcl} v_1 & = & u_1 \bmod p_1 \\ v_2 & = & (u_2 - v_1){p_1}^{-1} \bmod p_2 \\ v_3 & = & ((u_3 - v_1){p_1}^{-1} - v_2){p_2}^{-1} \bmod p_3 \\ \dots & & \end{array}$$

– On évalue finalement C_i en faisant les calculs sur r mots.

Retenues – Vitesse théorique

Si (z_i) est la suite des retenues et (w_i) le résultat de la multiplication:

$$\begin{aligned} z_i &= \lfloor (z_{i-1} + C_i)/M \rfloor \\ w_i &= (z_{i-1} + C_i) \bmod M \end{aligned}$$

Dans le cas $r = 3$ on peut supposer $0 \leq z_i \leq M^2 - 1$. En effet:

$$0 \leq \lfloor (z_{i-1} + C_i)/M \rfloor \leq 2^{k-1} M + M \leq M^2 - 1$$

Pour ces 3 dernières étapes on estime:

$$\begin{aligned} N_{mulmod} &= r(r-1)2^k \\ N_{addmod} &= r(r-1)2^k \end{aligned}$$

(Le choix de la base T ou mieux encore p_1 serait plus avantageux ici).

Au total :

$$\begin{aligned} N_{mulmod} &= r(2r + 3k - 4)2^{k-1} + 3r \\ N_{addmod} &= r(k + r - 1)2^k \end{aligned}$$

Par rapport à la méthode normale (r=3):

$$Ratio_{mulmod} = \frac{9k - 6}{2^{k-1}} + \frac{9}{2^{2k-2}}$$

Méthode de Pollard meilleure pour $k = 7$ (600 chiffres).

Inverse – Racine Carrée – π

On suppose ici pour plus de simplicité que le temps nécessaire pour faire une multiplication est linéaire en le nombre de chiffres. Cette approximation est en fait assez précise...

- Inverse: on utilise la suite
$$z_{k+1} = 2z_k - a.{z_k}^2$$
qui converge vers $1/a$ avec $1/2 \leq a < 1$.
Temps équivalent: 3 multiplications.

- Racine carrée: on utilise la suite
$$z_{k+1} = (3z_k - a.{z_k}^3)/2$$
qui converge vers $1/\sqrt{a}$ avec $1/4 \leq a < 1$.
Temps équivalent: 5 multiplications.

- Utilisation de la formule de Salamin reposant sur les propriétés de la suite arithmético-géométrique. La convergence est quadratique. Le coût est en $O(n \log(n))$ dans le cas de notre approximation.

  Temps équivalent: $9(k + 3) + 7$ multiplications pour 9.2^k décimales.

Performances

Tests sur un *486DX2/66* .
Maple a été lancé sur une station *DEC Alpha*.

	Temps (secondes)			
	k=12 (18000)	k=13 (36000)	k=14 (73000)	Maple (73000)
$x * y$	0.7	1.6	3.4	35
$1/x$	2.0	4.3	9.3	60
$\sqrt{2}$	2.1	4.5	9.9	80
$\sqrt{x}$	3.8	8.1	18	80
π	100	225	445	

Le calcul de 2.35 millions de décimales de π par la méthode de Salamin a demandé 4h10 sur un *Pentium 90*.