Représentation des objets mathématiques et algorithmes

Préparation agrégation, option C

Révision du 27/01/2011

Les objets de base à représenter sur machine sont d'abord les scalaires. On distingue les objets sur lesquels le microprocesseur opère directement dont la taille mémoire est fixée, ce sont les entiers courts et les flottants, et les objets dont la taille n'est pas fixée et qui sont gérés par des programmes, dont les entiers longs, les flottants multiprécision, les polynômes à une ou plusieurs variables. Les opérations arithmétiques (addition, multiplication, division) sur les premiers sont rapides et en temps constant, ce qui n'est pas le cas des autres, le temps dépendant de la taille des objets, on utilise des algorithmes plus ou moins sophistiqués mathématiquement parlant pour réaliser les opérations arithmétiques de base le plus rapidement possible, dont voici quelques exemples.

1 Les entiers

On distingue les entiers machines (codés sur 32 ou 64 bits, signés ou non) et les entiers en précision arbitraire (dont les opérations arithmétique de base ne sont pas effectuées par le microprocesseur mais doivent être programmées). Pour représenter un entier en précision arbitraire, on utilise par exemple un mot de 32 bits pour représenter le signe et la taille de l'entier, puis l'écriture en base 2^{32} de la valeurs absolue de l'entier. La plupart des implémentations utilisent des algorithmes efficaces pour :

- la multiplication : Karatsuba, Toom-3, FFT, dont certains sont esquissés plus loin pour les polynômes (l'adaptation de ces algorithmes aux entiers nécessitant de tenir compte des retenues)
- la division : la méthode "de base" consiste à poser la division et nécessite un temps proportionnel à la taille du diviseur par la taille du quotient. Il existe aussi des améliorations.
  On dit qu'un nombre écrit en base β^N est normalisé si son écriture commence par un digit $B \geq \beta^N/2$. Si on divise un nombre A de taille $2N$ en base β par un nombre B normalisé de taille N, avec N pair, l'algorithme "diviser pour régner" consiste à faire une division de base dont les "chiffres" sont des entiers longs de taille $N/2$ en base β (on divise alors un nombre de 4 chiffres écrit en base $\beta^{N/2}$ par un nombre de 2 chiffres), les multiplications que l'on effectuera entre les "chiffres" du quotient et du diviseur peuvent alors utiliser un algorithme de

multiplication rapide. Pour diviser un nombre a de 4 chiffres par un nombre b de 2 chiffres, on calcule le premier chiffre du quotient en utilisant les 2 premiers chiffres de a et le premier chiffre de b (en appelant récursivement l'algorithme de division rapide), on obtient un candidat quotient q que l'on décale et multiplie par b et retranche de a, on ajuste alors q en fonction du signe du reste (on diminue q si le reste est négatif). Si b est normalisé, donc au moins égal à la moitié du plus petit nombre écrit sur 3 "chiffres" (par exemple en base 10, si b est supérieur à 50), on montre qu'on doit diminuer q de 0, 1 ou 2. On recommence pour calculer le deuxième chiffre du quotient. Par exemple pour calculer le quotient de 4513 par 68, on commence par diviser 45 par 6, il y va 7 fois, 7 fois 68 font 476, donc le reste est 451-476=-25, donc on diminue le premier chiffre du quotient de 7 à 6, on ajoute 68 au reste qui vaut 43 il reste à diviser 433 par 68, 43 divisé par 6 il y va 7 fois, puis 433 moins 7 fois 68 font -43, on diminue donc le deuxième chiffre du quotient (le quotient final est 66), et on ajoute 68 au reste qui devient 25. Les opérations longues sont les 2 multiplications de 1 chiffre par 2 chiffres.
Si le diviseur n'est pas normalisé, on multiplie les 2 nombres à diviser par une puissance de 2 pour qu'il soit normalisé.
Il existe aussi une méthode théorique de complexité meilleure appelée meéthode de Newton (cf. par exemple Gerhard et Von zur Gathen).

– l'extraction de racine carrée : on peut distinguer deux algorithmes, savoir si le nombre est un carré parfait, extraire la partie entière de la racine carrée. Pour le premier, la documentation de GMP indique
'mpz_perfect_square_p' is able to quickly exclude most non-squares by checking whether the input is a quadratic residue modulo some small integers.
The first test is modulo 256 which means simply examining the least significant byte. Only 44 different values occur as the low byte of a square, so 82.8% of non-squares can be immediately excluded. Similar tests modulo primes from 3 to 29 exclude 99.5% of those remaining, or if a limb is 64 bits then primes up to 53 are used, excluding 99.99%. A single Nx1 remainder using 'PP' from 'gmp-impl.h' quickly gives all these remainders.
Pour le deuxième, il existe aussi des algorithmes récursifs utilisant la possibilité de multiplier rapidement deux entiers longs de même taille.

2 Les flottants

On va d'abord décrire les flottants double précision (double). La représentation d'un double en mémoire se compose de 3 parties : le bit de signe $s = \pm 1$ sur 1 bit, la mantisse $M \in [0, 2^{52}[$ sur 52 bits, et l'exposant $e \in [0, 2^{11}[$ sur 11 bits. Pour les nombres "normaux", l'exposant est en fait compris entre 1 et $2^{11} - 2$, le nombre représenté est le rationnel

$$(1 + \frac{M}{2^{52}})2^{e+1-2^{10}}$$

L'exposant 0 est réservé aux nombres dénormalisés de la forme $M/2^{52}2^{2-2^{10}}$, qui permettent de représenter des nombres plus petits que $2^{2-2^{10}}$ sous forme dénormalisée, pour éviter que l'on puisse avoir simultanément $x \neq y$ et $x - y = 0.0$), l'exposant 2^{11}

sert à coder les dépassements de capacité (+ et - l'infini ainsi que NaN, not a number, obtenu par exemple par division de 0 par 0) Exemples :

- -2
  Signe négatif. Il faut diviser sa valeur absolue 2 par 2^1 pour être entre 1 et 2 dont $e + 1 - 2^{10} = 1$, l'exposant est $e = 2^{10}$. On a alors $r = 1$, $r - 1 = 0$. Représentation
  `1 10000000000 00000000...0000`
- 1.5=3/2
  Signe positif, compris entre 1 et 2 dont l'exposant vérifie $e + 1 - 2^{10} = 0$ soit $e = 2^{10} - 1 = 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$. On a $r - 1 = 1/2 = 2^{-1}$. D'où la représentation
  `0 01111111111 10000000...0000`

Lorsque le développement en base 2 est infini, on adopte une règle d'arrondi du dernier bit au nombre supérieur si le bit suivant est un 1. L'arrondi au plus proche entraine une erreur relative de représentation de 2^{-53}.

Les opérations sur les flottants sont très rapides, car implémentées directement par les micro-processeurs mais ne sont pas exactes. Ces erreurs se propagent selon la règle approximative : l'erreur absolue d'une somme/différence est inférieure ou égale à la somme des erreurs absolues sur les arguments, l'erreur relative sur un produit ou un quotient est inférieure ou égale à la somme des erreurs relatives sur les arguments (en toute rigueur, il faut y ajouter l'erreur relative d'arrondi et par exemple le produit des erreurs relatives pour le produit).

On peut diminuer les erreurs d'arrondis en utilisant des flottants multiprécision, dont le principe est identique aux flottants machines, mais dont la mantisse est codée sur un nombre paramétrable de bits. Les opérations sont alors plus lentes, car elles nécessitent de faire des opérations sur des entiers (correspondant à la mantisse) de taille arbitraire. Certaines implémentations permettent de spécifier l'arrondi à effectuer pour chaque opération, ce qui permet d'implémenter une arithmétique des intervalles et donc d'obtenir un encadrement rigoureux du résultat obtenu. Mais on ne peut pas empêcher les problèmes des opérations en flottant comme par exemple :

- la non associativité de l'addition (par exemple $(2^{-53} + 1) - 1 \neq 2^{-53}$ dans les flottants)
- le fait que par exemple $0.3 - 3 \times 0.1$ n'est pas nul si les flottants sont représentés en base 2 (certains microprocesseurs peuvent travailler en base 10, on parle de BCD, certains logiciels comme par exemple maple, utilisent par défaut des flottants codés en base 10).

On peut d'ailleurs utiliser ces arrondis pour déterminer la base de représentation utilisée par le programme suivant (écrit en syntaxe maple)

```
A:=1.0; B:=1.0;
while ((A+1.0)-A)-1.0=0.0 do A:=2*A; od;
while ((A+B)-A)-B <> 0.0 do B:=B+1.0; od;
B;
```

3 Les polynômes à une variable.

On peut représenter les polynômes à une variable par une liste de coefficients, classés par ordre croissant ou décroissant de degré. Cette représentation est appelée dense. La représentation creuse sert pour les polynômes ayant peu de coefficients non nuls (par exemple $x^{100} - 1$). On suppose ici que les opérations de base sur les coefficients sont en temps constant (par exemple les coefficients sont dans un corps fini ou dans les flottants). L'opération d'addition de deux polynômes est proportionnelle en temps au nombre de coefficients représentés. Pour la multiplication de deux polynômes creux, le temps de calcul est proportionnel au produit du nombre de coefficients représentés. Pour les polynomes denses, il existe plusieurs méthodes permettant d'améliorer les performances.

3.1 L'algorithme de Karatsuba

Si P et Q sont de degrés $< 2n$, on peut écrire

$$P = A + Bx^n, Q = C + Dx^n$$

avec A, B, C, D de degré $< n$. Pour calculer PQ, on effectue

$$\begin{aligned} PQ &= (A + Bx^n)(C + Dx^n) = AC + BDx^{2n} + (AD + BC)x^n \\ &= AC + BDx^{2n} + [(A + B)(C + D) - AC - BD)]x^n \end{aligned}$$

Au lieu de faire 4 produits de polynômes de degré $< n$, on effectue seulement 3 produits de polynômes de degré $< n$, il faut ensuite faire des opérations de soustraction, mais elles sont moins couteuses que les opérations de multiplication, et il faut "répartir" selon les puissances de x. On montre que cet algorithme permet de multiplier deux polynômes de degré n en un temps d'ordre $n^{\ln(3)/\ln(2)}$ au lieu de n^2.

Il existe des variantes de cet algorithme permettant de multiplier en un temps asymptotique d'ordre égal à n^r pour $r > 1$ arbitraire (mais la constante devant n^r est d'autant plus grande que r est petit).

3.2 La FFT

Soient P et Q deux polynômes dont le degré du produit PQ est inférieur strict à N (par exemple P et Q de degrés strictement inférieur à $N/2$). Pour le calculer il suffit de connaitre sa valeur en N points distincts ξ_k, ce qui peut se faire en faisant le produit terme à terme $P(\xi_k)Q(\xi_k)$. Pour que cette méthode soit intéressante, il faut pouvoir passer rapidement de la représentation polynomiale habituelle P à la représentation de la liste des $P(\xi_k)$ et inversement. Ceci est possible lorsque N est une puissance de 2, et qu'il existe des $\xi_k = \omega^k$ racines N-ième de l'unité (par exemple si les coefficients sont complexes flottants mais aussi dans certains corps finis), il s'agit de la FFT (Fast Fourier Transform) qui nécessite alors $N \ln(N)$ opérations.

Soit donc P un polynôme de degré strictement inférieur à N :

$$P(X) = \sum_{j=0}^{N-1} p_j X^j$$

que l'on souhaite évaluer aux N points 1, ω, ..., ω^{N-1}. Si $N = 2M$, on découpe P en 2 parties de même longueur par division euclidienne par X^M :

$$P(X) = X^M Q(X) + R(X)$$

on a alors

$$P(\omega^{2k}) = (Q+R)((\omega^2)^k), \quad P(\omega^{2k+1}) = (-Q+R)_\omega((\omega^2)^k) \quad S_\omega(X) = \sum s_k \omega^k X^k$$

On est donc ramené à deux additions de 2 polynômes de degré M, une multiplication coefficient par puissances ($4M$ opérations), et au calcul des transformées de Fourier discrète de $Q + R$ et $R - Q$. Si $N = 2^n$, ces transformées peuvent encore se calculer par division successives par 2, on vérifie que cela nécessite $O(n2^n)$ opérations, donc $N \ln(N)$ opérations.

4 Les polynômes à plusieurs variables

Le problème principal des polynômes à plusieurs variables est d'avoir beaucoup de coefficients même pour des degré totaux petits, ainsi le nombre de monomes possible en degré total d et nombre de variables n est `comb(d+n,n)`, en degrés partiels il est majoré par $\prod_{i=1}^{n}(d_i + 1)$ où d_i désigne le degré par rapport à la i-ième variable. Il est donc souvent impraticable de travailler avec des polynomes denses. On utilise alors des représentations creuses, par exemple une table de coefficients indiciées par des monomes (des n-uplet d'entiers représentant les degrés partiels par rapport aux variables du polynôme).

Pour faire le produit de deux polynômes, il faut donc faire le produit terme à terme et additionner les monomes obtenus de même degré. Pour accélerer le temps de calcul d'un produit de polynômes, on peut utiliser des astuces :

- Si les polynômes sont denses par rapport à chaque variable :
  On peut se ramener à une seule variable, par exemple pour deux polynômes $P(x, y), Q(x, y)$ en deux variables x et y dont les degrés partiels sont n_1, m_1 par rapport à x et n_2, m_2 par rapport à y, on pose $x = y^{m_1+m_2+1}$, et on multiplie $P'(y) = P(y^{m_1+m_2+1}, y)$ par $Q'(y) = Q(y^{m_1+m_2+1}, y)$ puis on détermine $PQ(x, y)$ "en écrivant le produit $P'Q'$ en base $y^{m_1+m_2+1}$". On peut alors utiliser l'une des méthodes de multiplication rapide décrites précédemment. Il y a toutefois un inconvenient majeur à utiliser cette méthode, c'est que P' et Q' ont beaucoup de coefficients nuls, ils sont creux si on essaye de généraliser à une dimension plus grande que 2.
  Il est plus intéressant dès les petites dimensions d'interpoler le produit des deux polynômes. On donne à l'une des variables $d_i + 1$ valeurs (d_i désignant le degré partiel par rapport à cette variable), on fait (récursivement) le produit des polynômes à une variable de moins et on applique l'algorithme des différences divisées.
- Si on multiplie terme à terme :
  Il faut un conteneur pour les monomes du polynôme produit dans lequel on peut rapidement déterminer si un monome identique existe déjà (dans ce cas

on ajoute le produit des coefficients des termes que l'on multiplie au coefficient du monome existant) ou n'existe pas (dans ce cas on ajoute ce monome, coefficient au conteneur). On peut utiliser des arbres binaires (tables dont l'indice est le monome et la valeur est le coefficient) ou des tables de hachage, qui permettent d'accéder plus rapidement à un élément de la table du polynôme produit (si la fonction de hachage est adaptée).
– Dans les deux cas, en majorant chaque degré partiel du produit, on peut utiliser un entier au lieu d'un n-uplet d'entiers pour représenter un monome, ceci facilite les tests de comparaison entre monomes (l'ordre lexicographique devenant l'ordre habituel sur les entiers). Pour cela, on généralise l'écriture en base b d'un entier, en calculant les restes successifs par les $d_i + 1$ (d_i désignant le degré partiel du produit par rapport à la i-ème variable).

5 Les corps finis

Ils sont isomorphes à $G = GF(p, n)$ où p est leur caractéristique et p^n leur cardinal, $GF(p, n)$ étant le quotient de $\mathbb{Z}/pZ[X]$ par un polynôme minimal irréductible de degré n. On peut donc représenter un élément de G par un polynôme et effecter les opérations arithmétiques sur G en faisant les opérations sur les polynômes suivi par une division euclidienne par le polynôme minimal. Si le polynôme minimal est primitif, alors G est composé de 0 et des quotients des puissances de x pour $i \in [0, p^n - 2]$, sinon on cherche un élément primitif a. On peut alors représenter les éléments de G par un entier, qu'on choisit égal à -1 pour 0 et compris entre 0 et $p^n - 2$ pour les puissances correspondantes de a. La multiplication et l'inversion dans cette représentation est immédiate, on construit ensuite une table d'addition et d'opposés, ce qui permet d'effectuer les opérations de base sur le corps de manière optimale en temps.

6 Idées de développement

– Justifier la complexité de l'algorithme de Karatsuba (ou/et des variantes).
– Illustrer la multiplication de polynômes à coefficients réels par FFT.
– Chercher un corps fini $\mathbb{Z}/q\mathbb{Z}$ permettant de faire la multiplication par FFT de 2 polynômes de degré $m = 2^n$ à coefficients entiers de valeur absolue plus petite que C. Indication : on cherchera q premier sous la forme $q = t2^m + 1, \quad t \in \mathbb{N}$ (existence garantie par le théoréme de Dirichlet), tel que $q/2$ soit plus grand que le plus grand coefficient du produit, et on cherchera ω une racine 2^n-ième primitive de 1 dans $\mathbb{Z}/q\mathbb{Z}$ ($\omega = \beta^t$).
– Flottants et erreurs d'arrondi.
– Opérations sur les entiers multiprécision.
– Discuter la multiplication des polynômes à plusieurs variables, selon qu'ils sont denses ou creux.
– Opérations sur les corps finis. Comment trouver un polynôme irréductible, voire primitif.