Calcul du PGCD en parallèle

Sidi Mohamed SEDJELMACI

Attaché de recherche au LIPN.

Journée en l'honneur du Pr. Christian LAVAULT.

Villtaneuse le 05/07/2011.

Une petite anecdote

Question: Quel est le numéro de téléphone de la salle A105 de l'institut Galilée ?

Réponse: C'est le 01 49 40 **28 62**.

Question: Peut-on se rappeler des 4 derniers chiffres ?

Réponse: Oui : Par exemple considérer la suite

$$u_0 = 2,\ u_1 = 8 \ \text{ et}$$
$$u_{n+2} = |u_{n+1} - u_n|.$$

Ce qui donne bien $u_2 = 6$ et $u_2 = 2$.

Propriétés de la suite $u_{n+2} = |u_{n+1} - u_n|$; $u_0, u_1 \geq 1$.

1. $\forall n \geq 0, \;\; u_n \geq 0.$

2. $\exists\, N \geq 0,$ tel que $u_N = 0$

3. Si $u_{k+1} = 0$, alors $u_k = PGCD(u_0, u_1)$

Exemple: $(\mathbf{2}, \mathbf{8}) \to \{\, \mathbf{6}, \mathbf{2}, \mathbf{4}, \mathbf{2}, \mathbf{2}, \mathbf{0} \,\}$ et $PGCD(6, 2) = 2.$

- Ancêtre de l'algorithme d'Euclide:
  1 itér. d'Euclide = plusieurs itér. $|u_{n+1} - u_n|$.
- Analyse en moyenne délicate (Knuth-Yao, 2000).
- Pire des cas de l'algorithme : $(u_0, u_1) = (2^{n-1}, 1)$ avec
  $3 \times 2^{n-1} - 2$ itérations.

PGCD de deux entiers

Motivations:

1. **Pratique:** Le PGCD est très utilisé en Calcul Formel, cryptographie, arithmétique des ordinateurs, etc...
2. **Théorique:** Complexité parallèle inconnue: *NC* or *P-complet* ?

   Séquentiel: $O(n \log^2 n \log\log n)$, Knuth (70)-Schönhage (71).

   Parallèle: $O_\epsilon(n/\log n)$ temps avec $O(n^{1+\epsilon})$ processeurs, Chor-Goldreich (90), Sorenson (94) and Sedjelmaci (08).

Depuis 1990 : ? Aucune amélioration !

Ce problème résiste encore (P ou NC ?)

Nom	Année	Worst-case
Euclide	~ -300	$O(n^2)$
Lehmer	1938	$O(n^2)$
Stein	1961	$O(n^2)$
Knuth	1970	$O(\log nM(n))$
Schönhage	1971	$O(\log nM(n))$
Brent-Kung	1983	$O(n^2)$
Jebelean-Weber	1993	$O(n^2)$
Sorenson	1994	$O(n^2/\log n)$
Stehlé et al.	2004	$O(\log nM(n))$
Möhler	2008	$O(\log nM(n)$

Table 1: Algorithmes du PGCD en séquentiel.

En pratique, on utilise les algorithmes du PGCD suivants :

- **Knuth-Schönhage**: Entiers très longs
- **Jebelean-Weber**: Entiers de moyenne taille
- **Euclide ou Binaire**: Petits entiers.
  (ou MBE, Sed.-Weber, IPL, 2011)

Auteurs	Temps	Nb. de proc.	Modèle
Brent-Kung, 1983	$O(n)$	$O(n)$	Systolic
Kannan et al., 1987	$O(n\,\frac{\log\log n}{\log n})$	$O(n^{2+\epsilon})$	CRCW
Adleman et al., rand., 1988	$O(\log^2 n)$	$\mathrm{e}^{O(\sqrt{n\log n})}$	CRCW
Chor-Goldreich, 1990	$O(n\,/\log n)$	$O(n^{1+\epsilon})$	CRCW
Sorenson, 1994	$O(n\,/\log n)$	$O(n^{1+\epsilon})$	CRCW
Sedjelmaci, 2008	$O(n\,/\log n)$	$O(n^{1+\epsilon})$	CRCW
Sorenson, rand., 2010	$O(n\,\frac{\log\log n}{\log n})$	$O(n^{6+\epsilon})$	EREW

Table 2: Algorithmes du PGCD en parallèle.

Algorithme PM de Brent-Kung

Il réduit 2 bits à chaque itération en temps constant 0(1).

• Il se base sur la simple propriété: Si u et v sont impairs alors:
Soit $u + v \bmod 4 = 0$, soit $u - v \bmod 4 = 0$ (exclusivement).

Exemples: $(13 + 7) \bmod 4 = 0$ et $(15 - 7) \bmod 4 = 0$.

• La retenue (pour + ou −) se fait progressivement uniquement par tranche de 2 bits (pipelining).

Coût paralèle: $O(n)$ en temps avec $O(n)$ processeurs (calcul systolique).

Algorithme KMR: Kannan-Miller-Rudolph

Soient u, v 2 entiers de n bits. On calcule en parallèle:
$r_k = ku \bmod v$, pour $0 \leq k \leq n$.

Principe des trous des pigeons: Il existe $i \neq j$ tels r_i et r_j ont les même $\log n$ premiers bits. Exemple $(u, v) = (143, 221)$, $n = 8$: on a:

$$
\begin{array}{rl}
& (0 \times 143) \bmod 221 = 0 = 000...0_2. \\
\rightarrow & (1 \times 143) \bmod 221 = 143 = 100...1_2. \\
& (2 \times 143) \bmod 221 = 65 = 010...1_2. \\
& (3 \times 143) \bmod 221 = 208 = 110...0_2. \\
\rightarrow & (4 \times 143) \bmod 221 = 130 = 100...0_2. \\
& (5 \times 143) \bmod 221 = 52 = 001...0_2. \\
& (6 \times 143) \bmod 221 = 195 = 110...1_2. \\
& (7 \times 143) \bmod 221 = 117 = 011...1_2. \\
& (8 \times 143) \bmod 221 = 39 = 001...1_2.
\end{array}
$$

• On remarque que
$(4 \times 143) \bmod 221 - (1 \times 143) \bmod 221 = 13 = v - (3u) \bmod v$.

• En faisant la différence $|r_i - r_j|$ on réduit à chaque itération de $O(\log n)$ bits. D'où $O(n/\log n)$ itérations. Chaque itération coûte $O(\log\log n)$ en temps avec $O(n^2 \log^2 n)$ processeurs.

• Coût paralèle total : $O(\frac{n \log\log n}{\log n})$ en temps avec $O(n^2 \log^2 n)$ processeurs sur une CRCW PRAM.

• L'algorithme KMR est le premier algorithme sous-linéaire.

Algorithme de Chor et Goldreich:

Remarque: Dans l'algorithme PM, seuls les deux derniers bits servent à déterminer la prochaine opération: $+$ ou $-$..

Idée: Une *phase* : Regroupe (*pack*) $O(\log n)$ itérations à la fois en considérant uniquement que les $O(\log n)$ derniers bits des opérandes.

• Il y a $O(n/\log n)$ phases.

• Chaque phase peut se faire en $O(1)$ en temps avec $O(n^{1+\epsilon})$ processeurs.

• Complexité paralèle totale : $O(n/\log n)$ en temps avec $O(n^{1+\epsilon})$ processeurs sur une CRCW PRAM.

Meilleure performance parallèle actuelle.

Question: Comment multiplier un entier de n bits par un entier de $\log n$ bits en temps constant ?

- Représenter un entier $A = \ldots a_4\, a_3\, a_2\, a_1$ de n bits par les 2 entiers suivants en base $\log n$:

$$A_P = \ldots a_4\, 0\, a_2\, 0 \text{ et } A_I = \ldots 0\, a_3\, 0\, a_1,$$

- Les a_i et les "0" sont des blocks de $(\log n)$ bits qui ne s'interceptent pas dans A_P et A_I et $A = A_P + A_I$.

Exemple: Calculer 7×1234. En base 10, on découpant par block d'un digit: $A_I = 1030$ et $A_P = 0204$. Puis

$$7 \times 1234 = 7 \times (1030 + 0204) = 7210 + 1428 = 8638.$$

- *Il n'y a pas de propagation de retenue dans les produits!*
- L'addition se fait en $O(1)$ en parallèle (Chandra et al.)

L'algorithme Par-ILE: (ISSAC'01 & JDA'08).

Input: $u \geq v$ et $k = 2^m = O(n)$ t.q. $p > 2m + 3$.
Output: $R_{ILE}(u, v)$.

Step 1:

$\lambda = 2m + n - p + 2$, $u_1 = \lfloor u/2^{p-\lambda} \rfloor$, $v_1 = \lfloor v/2^{p-\lambda} \rfloor$, and
$u = u_1 2^{p-\lambda} + u_2$, $v = v_1 2^{p-\lambda} + v_2$.

Step 2: **For** $i = 1, 2, .., 2^m$ **Do in parallel**

$q_i := \lfloor iu_1/v_1 \rfloor$; $r_i := iu_1 - q_i v_1$;
if $r_i < v_1/k$ **then** $r := r_i$; $a := i$; $b := q_i$;
if $v_1 - r_i < v_1/k$ **then** $r := v_1 - r_i$; $a := i$; $b := q_i + 1$;

End Do

Step 3: Compute in parallel
$R_{ILE} = |au - bv| = |r 2^{p-\lambda} + au_2 - bv_2|$;

Return $R_{ILE} < 2v/k$.

Exemple: Soient $u = 1\,759\,291$ et $v = 1\,349\,639$. Leurs représentations binaires sont respectivement:

$$\mathbf{11010110}\ 1100000111011_2 = 1\,759\,291$$
$$\mathbf{10100100}\ 1100000000111_2 = 1\,349\,639$$

On a $n = p = 21$. Pour $m = 3$, on obtain $\lambda = 2m + 2 = 8$, $u_1 = 214$ et $v_1 = 164$ (les bits représentant u_1 et v_1 sont en gras). En utilisant EEA à u_1 et v_1, on trouve les entiers q, r, b et a $(r = au + bv)$ suivants:

q	r	a	b
	214	1	0
	164	0	1
1	50	1	-1
3	14	-3	4
3	8	10	-13

Dans notre exemple, on obtient $a = -3$, $b = 4$,
$r = 14 < v_1/k = 164/8 = 20.50$ et

$$R_{ILE} = |-3u + 4v| = 120\,683 < v/8 = 168\,704.88$$

Propriétés de Par-ILE:

- Complexité parallèle: $O(n/\log n)_\epsilon$ en temps avec $O(n^{1+\epsilon})$ processeurs sur une CRCW PRAM (ISSAC'01).
- C'est le seul algorithme qui calcule en parallèle les coéfficients de Bézout avec cette performance (JDA'08).

Depuis 1990, aucune amélioration de la complexité en parallèle !

Questions:

- Est-ce que l'approche par réductions a atteint ses limites ?
- Existe-il d'autres approches ?

Quelques essais:

1) Le PGCD comme un S.L.P. (Straight Line Program) (ANTS'08).
SLP $\sim$ Algorithme sans test et sans branchements conditionels.

2) Une approche par intégrale pour calculer le PGCD (LAGOS'11).

$$\rightarrow \quad \gcd(a,b) = \int_0^1 H(a,b,s)\,ds \quad \text{(avec noyaux).}$$

Un PGCD par Straight Line Program (ANTS'08)

Définition (Miller, Ramachandran et Kaltofen):

Un straight-line program (ou SLP) sur un semi anneau commutatif $R = (R, +, \times, 0, 1)$ est une suite d'affectations de type $a \leftarrow b + c$ ou $a \leftarrow b \times c$, où b et c sont soit des éléments de R soit des variables définies précédemment.

Théorème (Valiant, Skyum, Berkowitz, Rackoff):

Tout programme séquentiel calculant un pôlynome de degré $< d$ en C étapes peut être convertis en un programme parallèle en temps $O(\ (\log d)\ (\log C + \log d)\ )$ avec $O(\ (Cd)^{\beta}\ )$ processeurs, pour un certain $\beta \geq 1$ approprié.

Applications:

- Le calcul du déterminant d'une matrice est dans NC (algorithme de Gauss).
- Le PGCD de deux polynômes est dans NC (calcul de sous-résultant).

Question: Existe t-il un SLP pour le PGCD de deux entiers ?

Input: $x,\ y > 0$ impairs ;

Output: $\mathrm{PGCD}(x, y)$;

$$\begin{pmatrix} u \\ v \end{pmatrix} \leftarrow \begin{pmatrix} x \\ y \end{pmatrix} ;$$

While ($u \neq v$)

$$\begin{pmatrix} u \\ v \end{pmatrix} \leftarrow \begin{pmatrix} v \\ (u+v)/2^t \end{pmatrix}; \ \text{ t.q.: } (u+v)/2^t \text{ est impair.}$$

EndWhile

Return u.

Exemple: Pour $(x, y) = (35, 19)$, on obtient:

$$\begin{pmatrix} 35 \\ 19 \end{pmatrix} \to \begin{pmatrix} 19 \\ 27 \end{pmatrix} \to \begin{pmatrix} 27 \\ 23 \end{pmatrix} \to \begin{pmatrix} 23 \\ 25 \end{pmatrix} \to \begin{pmatrix} 25 \\ 3 \end{pmatrix}$$

$$\to \begin{pmatrix} 3 \\ 7 \end{pmatrix} \to \begin{pmatrix} 7 \\ 5 \end{pmatrix} \to \begin{pmatrix} 5 \\ 3 \end{pmatrix} \to \begin{pmatrix} 3 \\ 1 \end{pmatrix} \to \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Théorème: Soient $u, v \geq 1$ deux entiers impairs de n bits, $n \geq 1$, tels que $|u - v| = r2^t > 1$, avec $r \geq 1$ impair, et $t \geq 1$. Soit (u_k, v_k) la suite des termes consécutifs obtenus dans l'algorithme du PGCD, avec $(u_0, v_0) = (u, v)$. Alors

- *i*) $\max\{u_{t+1}, v_{t+1}\} < (3/4)\ \max\{u, v\}$.
- *ii*) l'algorithme se termine après au plus $n^2/\log_2(4/3)$ itérations et retourne $\mathrm{PGCD}(u, v)$.
- *iii*) On peut remplacer la boucle **While** $u \neq v$ par la boucle **For** $i = 1$ **to** $3n^2$ dans l'algorithme.

Input: $x,\ y > 0$ impairs ;
Output: $\mathrm{PGCD}(x, y)$;

$$\begin{pmatrix} u \\ v \end{pmatrix} \leftarrow \begin{pmatrix} x \\ y \end{pmatrix};$$

For $i = 1$ **to** $3n^2$ **do**

$$\begin{pmatrix} u \\ v \end{pmatrix} \leftarrow \begin{pmatrix} v \\ (u+v)/2^t \end{pmatrix}; \ \text{t.q.: } (u+v)/2^t \text{ est impair.}$$

EndWhile
Return u.

L'algorithme Add-GCD.

Input: $A = (a_n, a_{n-1}, \ldots, a_1)_2$ un entier positif ;

Output: $A' = (a_n, a_{n-1}, \cdots, a_1)$, t.q.:

$A' = A/2^t$ est un entier impair.

$a_{n+1} = 0$;

for $k = 1$ **to** $n - 1$ **do**

$c = (1 + a_1) \pmod 2$

for $i = 1$ **to** n **do**

$a_i = c \,.\, a_{i+1} + (1 + c) \,.\, a_i \pmod 2$;

endfor

endfor

Return $A' = (a_n, a_{n-1}, \cdots, a_1)$.

L'algorithme MakeOdd en $O(n^2)$.

- L'algorithme ADD-GCD, avec la version MakeOdd, est un *SLP* définie sur le corps GF(2): C'est le *seul* SLP qui calcule le PGCD de deux entiers.

- Nombre d'étapes: $O(n^4)$.

- Les bits d'entrée sont $u_n, u_{n-1}, \ldots, u_1$, et $v_n, v_{n-1}, \ldots, v_1$ de u et v.

- Les bits de sortie sont $d_n, d_{n-1}, \ldots, d_1$ de $d =$PGCD(u, v). Chaque d_i est un polynôme multivarié par rapport aux variables d'entrée u_i et v_i. C'est-à-dire:

$$d_i = d_i(u_n, u_{n-1}, \ldots, u_1\,; v_n, v_{n-1}, \ldots, v_1).$$

Malheureusement le degré de chaque polynôme d_i, généré par ce SLP, croît exponentiellement, et le théorème de contraction de Valiant ne peut pas s'appliquer dans ce cas.

Calcul du PGCD par intégrale (LAGOS 2011)

L'idée:

PGCD

= Nb. de solutions d'une équation Diophantienne

= Coéfficient d'un produit de deux séries génératrices.

= Une intégrale de Cauchy pour calculer ce coéfficient:

$$\gcd(a,b) = \frac{1}{\pi}\int_0^{\pi} \cos[\,(b-a)x\,]\;\frac{\sin^2(abx)}{\sin(ax)\,\sin(bx)}\;dx.$$

Un PGCD pour des nombres réels ?

La formule précédente peut s'étendre à certains réels:

EXEMPLES:

$$\gcd(1/2, 1/2) = \frac{1}{\pi}\int_0^{\pi} \frac{\sin^2(x/4)}{\sin^2(x/2)}\ dx = \frac{4}{\pi}\int_0^{\pi/4} \frac{\sin^2(t)}{\sin^2(2t)}\ dt = \frac{1}{\pi}.$$

$$\gcd(1, 1/2) = \frac{1}{\pi}\int_0^{\pi} \cos(x/2)\ \frac{\sin(x/2)}{\sin(x)}\ dx = \frac{1}{\pi}\int_0^{\pi} \frac{1}{2}\ dx = \frac{1}{2}.$$

$\longrightarrow$ Quels sens donner à ces formules ?

- P. Erdös a dit, en parlant de la suite de Collatz ($3n+1$):

  Les mathématiques ne sont prêtes pour aborder ce genre de problème !

- Cette réflexion pourrait s'appliquer au calcul du PGCD en parallèle: Peut-être faudrait-il développer d'autres outils ?

- Importance de cette question:

$$(\, P = NC\,?) \;\longleftrightarrow (\, P = NP\,?)$$