Division nach Beame, Cook und Hoover

Andreas Abel

19. November 1996, 5. März 1997

1 Division mittels Inversenbildung in NC^1

Annahme: Es steht ein NC^1-Schaltkreis zur Verfügung, der zu einem gegebenen $y \in \mathbf{N}\backslash\{0\}$ das Inverse auf n Stellen berechnet (inverse(y, n)), also

$$0 \leq \frac{1}{y} - \text{inverse}(y, n) \leq \frac{1}{y \cdot 2^n}$$

Dann berechnet folgender NC^1-Algorithmus bei Eingabe von natürlichen Zahlen $x \geq 0$, $y > 0$ den Quotienten $q = \lfloor \frac{x}{y} \rfloor$:

AC^0	$n := \lvert x \rvert$
NC^1	$z := \text{inverse}(y, n)$
NC^1	$r := x - y \cdot \lfloor x \cdot z \rfloor$ („Rest“)
AC^0	if $r \geq y$ then $q := \lfloor x \cdot z \rfloor + 1$
	else $q := \lfloor x \cdot z \rfloor$

Beweis: Für den Rest $R := x - q \cdot y$ der Division muss gelten:

$$0 \leq x - q \cdot y < y$$

Ist nun $0 \leq r < 2y$ in obigem Algorithmus gegeben, so ist der Algorithmus korrekt, denn:

1. Fall: $0 \leq r < y \Rightarrow R = x - q \cdot y = x - y \cdot \lfloor x \cdot z \rfloor = r$
2. Fall: $y \leq r < 2y \Rightarrow R = x - q \cdot y = x - y \cdot (\lfloor x \cdot z \rfloor + 1) =$
$= x - y \cdot \lfloor x \cdot z \rfloor - y = r - y \Rightarrow 0 \leq R < y,$

also in beiden Fällen $0 \leq R < y$. Wir zeigen sogar:

$$0 \leq r < y + 1$$

Beweis hierfür: Per Definition galt

$$0 \leq \frac{1}{y} - \underbrace{\text{inverse}(y,n)}_{:=z} \leq \frac{1}{y \cdot 2^n} \quad \Rightarrow$$

$$0 \leq \frac{x}{y} - x \cdot z \leq \frac{1}{y} \cdot \frac{x}{2^n} < \frac{1}{y},$$

da $n = |x|$, d.h. $2^{n-1} \leq x < 2^n \quad \Rightarrow \frac{x}{2^n} < 1$. Weiter

$$\begin{aligned}
&0 \leq x - y \cdot x \cdot z < 1 \\
&0 \leq x - y(\lfloor x \cdot z \rfloor + x \cdot z - \lfloor x \cdot z \rfloor) < 1 \\
&y(x \cdot z - \lfloor x \cdot z \rfloor) \leq x - y \cdot \lfloor x \cdot z \rfloor < 1 + y(x \cdot z - \lfloor x \cdot z \rfloor),
\end{aligned}$$

und wegen $0 \leq x \cdot z - \lfloor x \cdot z \rfloor < 1$ (nach Definition von $\lfloor \cdot \rfloor$)

$$0 \leq x - y \cdot \lfloor x \cdot z \rfloor < 1 + y.$$

Also $0 \leq r < y + 1$.

2 Inversenbildung durch geometrische Reihe in NC^1

Annahme: Es stehen NC^1-Schaltkreise zur Verfügung, die ε^i ($0 \leq \varepsilon < 1$, $0 \leq i < n$) berechnen.
Dann ist folgender Algorithmus zur Berechnung von $z :=$ inverse(y, n) in NC^1 und korrekt (sei $y > 0$):

AC^0	$k := \lvert y \rvert$
AC^0	$\varepsilon := 1 - y \cdot 2^{-k}$
NC^1	berechne ε^i für $i = 0, 1, ..., n-1$ parallel
NC^1	$z := 2^{-k} \sum_{i=0}^{n-1} \varepsilon^i$

Beweis: Zu zeigen ist $0 \leq \frac{1}{y} - \text{inverse}(y, n) \leq \frac{1}{y \cdot 2^n}$. Gesetzt ist $k = |y|$, d.h. $2^{k-1} \leq y < 2^k \iff 2^{-1} \leq y \cdot 2^{-k} < 2^0 \iff \frac{1}{2} \geq \underbrace{1 - y \cdot 2^{-k}}_{:=\varepsilon} > 0$. Also können wir die geometrische Reihe wie folgt anwenden:

$$\begin{aligned}
\frac{1}{y} &= \frac{2^{-k}}{y \cdot 2^{-k}} = \frac{2^{-k}}{1 - (1 - y \cdot 2^{-k})} = 2^{-k} \frac{1}{1-\varepsilon} = \\
&= 2^{-k} \sum_{i=0}^{\infty} \varepsilon^i = 2^{-k} (\sum_{i=0}^{n-1} \varepsilon^i + \varepsilon^n \sum_{i=0}^{\infty} \varepsilon^i) \\
&= 2^{-k} (\sum_{i=0}^{n-1} \varepsilon^i + \frac{\varepsilon^n}{1-\varepsilon})
\end{aligned}$$

Mit $\text{inverse}(y,n) = 2^{-k}\sum_{i=0}^{n-1}\varepsilon^i$ erhalten wir

$$d := \frac{1}{y} - \text{inverse}(y,n) = \frac{2^{-k}}{1-\varepsilon}\varepsilon^n = \frac{1}{y}\varepsilon^n,$$

denn $1 - y\cdot 2^{-k} = \varepsilon \iff 1-\varepsilon = y\cdot 2^{-k} \iff 2^k(1-\varepsilon) = y \iff \frac{2^{-k}}{1-\varepsilon} = \frac{1}{y}$, und wegen $\varepsilon \leq \frac{1}{2}$ gilt ($d \geq 0$ klar)

$$0 \leq d \leq \frac{1}{y\cdot 2^n}.$$

Die Berechnung einer Potenz ε^n lässt sich auf die Berechnung des Produkts $\prod_{i=1}^n x_i$ von n natürlichen Zahlen $x_1, ..., x_n$ zurückführen. Der naheliegendste Algorithmus dafür ist aber in NC^2. Idee für einen NC^1-Algorithmus:

$$\log(\prod_{i=1}^n x_i) = \sum_{i=1}^n (\log x_i)$$

Zurückführung auf eine Summe von Logarithmen. Da wir nur Ganzzahl-Arithmetik zur Verfügung haben, benötigen wir den *diskreten Logarithmus*.

3 Zahlentheoretische Grundlagen

Die multiplikativen Gruppen

$$\mathbf{Z}_p^* = \{\overline{1}, \overline{2}, ..., \overline{p-1}\} \qquad (p \text{ prim})$$

sind **zyklisch**, d.h. es existiert ein Element $g \,\epsilon\, \mathbf{Z}_p^*$, genannt **Primitivwurzel**, mit

$$\mathbf{Z}_p^* = \{g^0, g^1, ..., g^{p-2}\}.$$

Also existiert zu jedem $x \,\epsilon\, \mathbf{Z}_p^*$ ein $a \,\epsilon\, \{0, 1, ..., p-2\}$ mit

$$g^a \equiv x \bmod p.$$

Dieses heißt **Index** oder **diskreter Logarithmus** von x zur Basis g und wird mit $\text{ind}_g(x)$ bezeichnet.
Mit diesen zahlentheoretischen Hilfsmitteln können wir in der Tat iterierte Multiplikation auf iterierte Addition zurückführen; denn ist $P > \prod_{i=1}^n x_i$ prim und g eine Primitivwurzel von $\mathbf{Z}_p^*$, so gilt

$$\prod_{i=1}^n x_i \equiv \prod_{i=1}^n g^{\text{ind}_g(x_i)} \equiv g^{\sum_{i=1}^n \text{ind}_g(x_i)} \bmod P.$$

Es stellt sich aber heraus, dass es nur für kleine m NC^1-Algorithmen zur Berechnung von $x \bmod m$ oder des diskreten Logarithmus in $\mathbf{Z}_m^*$ gibt. Abhilfe schafft hier ein weiterer Satz der Zahlentheorie, der *Chinesische Restsatz*. Doch zuerst der

Algorithmus zur Berechnung von x mod m für kleine m

EINGABE:	$x = \sum_{i=0}^{n-1} x_i 2^i$, $m \leq n$
AUSGABE:	$x \bmod m$
	$a_{im} := 2^i \bmod m$
NC^1	$y := \sum_{i=0}^{n-1} x_i a_{im}$ (Es gilt $y < n \cdot m$)
NC^1	Berechne parallel $y - i \cdot m$ für $i = 0, ..., n-1$
AC^0	Wähle das kleinste $y - i \cdot m$ aus, das ≥ 0 ist, und gebe es aus.

Dabei werden die a_{im} in der Konstruktionsphase für alle $i = 0, ..., m-1$ und $m = 2, ..., n$ berechnet und fest in den Schaltkreis eingebaut, alle anderen Schritte sind in NC^1 .

4 Chinesischer Restsatz

Seien $p_1, ..., p_m$ paarweise teilerfremde natürliche Zahlen und $P := \prod_{i=1}^m p_i$. Dann gibt es genau ein $x \,\epsilon\, \mathbf{Z}_P$ mit

$$\begin{aligned} x_1 &\equiv x \bmod p_1 \\ x_2 &\equiv x \bmod p_2 \\ &\vdots \\ x_m &\equiv x \bmod p_m \end{aligned}$$

für $x_1 \,\epsilon\, \mathbf{Z}_{p_1}, ..., x_m \,\epsilon\, \mathbf{Z}_{p_m}$. Sei $v_i \,\epsilon\, \mathbf{Z}_{p_i}$ das Inverse zu $\frac{P}{p_i}$ modulo p_i für alle $i = 1, ..., m$, d.h.

$$v_i \cdot \frac{P}{p_i} \equiv 1 \bmod p_i.$$

Dann gilt

$$x \equiv \sum_{i=1}^{m} x_i \cdot v_i \cdot \frac{P}{p_i} \bmod P.$$

Beweis hierfür:

$$x \equiv x_i \cdot \underbrace{v_i \cdot \frac{P}{p_i}}_{\equiv 1 \bmod p_i} + \sum_{j \neq i} \underbrace{x_j \cdot v_j \cdot \frac{P}{p_j}}_{\equiv 0 \bmod p_i, \, \text{da}\, p_j | P} \bmod p_i$$

Annahme: Wir haben NC^1-Algorithmen, die $P := \prod_{i=1}^m p_i$ und $\frac{P}{p_i}$ für $i = 1, ..., m$ berechnen.
Dann ist folgender Algorithmus, der den Chinesischen Restsatz anwendet, d.h. zu gegebenen paarweise teilerfremden

$$1 < p_1 < p_2 < ... < p_m \leq m^2$$

und

$$x_i = (x \bmod p_i) \qquad (i = 1, ..m)$$

x berechnet, korrekt und in NC^1:

NC^1	(1) Berechne $P := \prod_{i=1}^m p_i$.
NC^1	(2) Berechne $\frac{P}{p_i}$ für $i = 1, 2, ..., m$ parallel.
NC^1	(3) Löse die Gleichungen $v_i \cdot \frac{P}{p_i} \equiv 1 \bmod p_i$ für $i = 1, ..., m$ parallel. (Z.B. durch Ausprobieren)
NC^1	(4) Berechne $v_i \cdot \frac{P}{p_i}$ für $i = 1, ..., m$ parallel.
NC^1	(5) Berechne $x_i \cdot v_i \cdot \frac{P}{p_i}$ für $i = 1, ..., m$ parallel.
NC^1	(6) Berechne $y := \sum_{i=1}^m x_i \cdot v_i \cdot \frac{P}{p_i}$

Das gesuchte Ergebnis ist nun

$$y \bmod P = y - t \cdot P, \quad t = \left\lfloor \frac{y}{P} \right\rfloor.$$

Im folgenden leiten wir uns die Berechnung von t her: Sei k die kleinste Zweierpotenz $\geq m$, $k = 2^{\lceil \log_2 m \rceil}$.

$$\begin{aligned}
\frac{y}{P} &= \sum_{i=1}^m \frac{x_i v_i}{p_i} = \frac{1}{k} \sum_{i=1}^m \frac{k x_i v_i}{p_i} \geq \frac{1}{k} \sum_{i=1}^m \left\lfloor \frac{k x_i v_i}{p_i} \right\rfloor =: s \qquad \Rightarrow \\
0 &\leq y - s \cdot P = \frac{1}{k} \sum_{i=1}^m \underbrace{\left(\frac{k x_i v_i}{p_i} - \left\lfloor \frac{k x_i v_i}{p_i} \right\rfloor \right)}_{<1} P < \\
&< \frac{1}{k} \sum_{i=1}^m P = \underbrace{\frac{m}{k}}_{\leq 1} P \leq P
\end{aligned}$$

Also gilt sicher: $0 \leq y - \lfloor s \rfloor P < 2P$. Damit erhalten wir t wie folgt:

1. Fall: $y - \lfloor s \rfloor P < P \Rightarrow t = \lfloor s \rfloor$
2. Fall: $y - \lfloor s \rfloor P \geq P \Rightarrow t = \lfloor s \rfloor + 1$

Fortsetzung des Algorithmus:

NC^1	(7)	Berechne $k \cdot v_i \cdot x_i$ für $i = 1, ..., m$ parallel.
$\leq NC^1$	(8)	Berechne $\left\lfloor \frac{k \cdot v_i \cdot x_i}{p_i} \right\rfloor$ für $i = 1, ..., m$ parallel. Da $p_i = O(\log m)$ und $k \cdot v_i \cdot x_i = O(\log m)$ genügt ein einfacher Divisionsalgorithmus mit maximal linearer Komplexität („any reasonable division circuit“).
NC^1	(9)	Berechne $\sum_{i=1}^{m} \left\lfloor \frac{k \cdot v_i \cdot x_i}{p_i} \right\rfloor$.
NC^0	(10)	Berechne $\lfloor s \rfloor = \left\lfloor \frac{\sum_{i=1}^{m} \left\lfloor \frac{k \cdot v_i \cdot x_i}{p_i} \right\rfloor}{m} \right\rfloor$ (shift right).
NC^1	(11)	if $y - \lfloor s \rfloor P < P$ then $x := y - \lfloor s \rfloor P$
NC^1		else $x := y - (\lfloor s \rfloor + 1)\, P$

5 Berechnung des iterierten Produkts

EINGABE: n n-bit-Zahlen $x_1, ..., x_n$
AUSGABE: Produkt $\prod_{i=1}^{n} x_i$

Konstruktionsphase:

- Ermittle m paarweise verschiedene Primzahlen $1 < p_1 < p_2 < ... < p_m \leq n^2$ mit
$$P := \prod_{i=1}^{m} p_i \geq 2^{n^2} > \prod_{i=1}^{n} x_n$$
(immer möglich für $n > 5$).

- Ermittle einen Generator (Primitivwurzel) g_i für $\mathbf{Z}_{p_i}^*$ und eine Tabelle der diskreten Logarithmen
$$g_i^0, g_i^1, ..., g_i^{p_i - 2}$$
für $i = 1, ..., m$.

- Berechne $\frac{P}{p_i} = \prod_{j \neq i} p_j$ und v_i, das Inverse zu $\frac{P}{p_i}$ modulo p_i.

Alle diese Werte werden fest in den Schaltkreis integriert.

Eigentlicher Schaltkreis (Berechnungen zur Laufzeit):

NC^1 Berechne $x_i \bmod p_j$ für $i = 1, ..., n, j = 1, ..., m$ parallel.
AC^0 Lese die diskreten Logarithmen aus
$l_{ij} := \mathrm{ind}_{g_j}(x_i \bmod p_j)$
NC^1 Berechne die Summen $S_j := \sum_{i=1}^{n} l_{ij}$.
AC^0 Lese die Anti-Logarithmen aus $y_j := g_j^{S_j}$.
NC^1 Berechne $y := \prod_{i=1}^{n} x_i$ durch Anwendung des
Chinesischen Restsatzes auf $y_1, ..., y_m$.

Damit hätten wir den NC^1-Schaltkreis für Ganzzahl-Division vollständig realisiert. Da die Konstanten P und $v_i \cdot \frac{P}{p_i}$ vorausberechnet werden müssen, ist der Schaltkreis nicht log space-uniform, nur P-uniform, d.h. kann auf einer Touringmaschine mit polynomialem Platz auf dem Arbeitsband konstruiert werden. Es wurden jedoch auch log space-uniforme Schaltkreise gefunden, nur sind diese nicht in NC^1, sondern haben die Tiefe $O(\log n \log \log n)$.

Literatur

[1] Paul W. Beame, Stephen A. Cook, H. James Hoover, *Log Depth Circuits for Division and Related Problems*, Siam J. Comput. Vol. 15, No. 4, November 1986

[2] R. M. Karp, V. Ramachandran, ???

[3] Peter Clote, Manuskript für zukünftiges Buch über Komplexitätstheorie

[4] Otto Forster, *Algorithmische Zahlentheorie*, vieweg 1996