数　値　解　析　32－3
（１９９０．３．１７）

多倍長浮動小数点数の除算を高速に行なう方法
— $O(n^2)$のアルゴリズムに関して —

小沢一文、海野啓明
（仙台電波高専）

多倍長浮動小数点数の高速な除算法を提案する。この計算法を用いると乗算と同じ速さで除算が実行できる。ここで提案するのは、$O(n^2)$（nは桁数）のアルゴリズムであり、我々が紙と鉛筆によって行なう方法(divide-and-correct 法)を用いている。 divide-and-correct法では、（単精度数）×（多倍長数）という演算がn回行なわれるが、ここではこの計算過程を省力化することによって高速化を達成している。時間計算量の解析を行なった結果、本アルゴリズムはニュートン法を用いる方法の３倍高速であることが判明した。また誤差解析の結果、本アルゴリズムの相対誤差は*machine* εのオーダであることが判明した。

A Fast $O(n^2)$ Algorithm for the Division of Multiple-Precision Floating Point Numbers

Kazufumi Ozawa Keimei Kaino
Sendai National College of Technology,
Kami-Ayashi, Aoba-Ku, Sendai 989-31, JAPAN.
e-mail:ozawa@sendai-ct.ac.jp

We propose a fast "divide-and-correct" algorithm for the division of multiple-precision floating point numbers(MPF). In this algorithm we reduce the computational effort of the operations of (single precision number)×MPF. The time-complexity analysis of the algorithms shows that our one is as fast as the usual multiplication algorithm of MPF and is 3 times fast as the Newton technique. This algorithm gives the results with the relative accuracy of *machine* ε.

1. はじめに

最近、研究者の間で数式処理言語が注目を集めるようになってきた。それらにはたいてい無限多倍長演算の機能が備わっているが、一般に、除算の計算速度が乗算の計算速度に比べてかなり遅いようである。本講演では、浮動小数点数の$O(n^2)$の除算アルゴリズムを、やはり$O(n^2)$の乗算と同じ程度に高速化する方法を提案する。

2. Divide-and-correct 法による除算

多倍長数の除算法は、divide-and-correct法と、Newton法を用いる方法の2つに分けられる。Divide-and-correct法は、基本的には我々が紙と鉛筆を用いて商を計算する方法と同じである。ここで、この方法の計算過程を示すと次のようになる：

以下の1)〜4)を必要な回数だけ繰り返す。

1)まず除数と被除数の先頭の数桁から商（1桁分）の見積りを計算し、

2)見積られた商の1桁に除数を掛けたものを計算し、

3)それを被除数から引いて剰余を計算する。このとき、それが負になったら商を修正する。

4)剰余を1桁左へシフトしたものを新たに被除数とする。

このdivide-and-correct法では、いかにして修正回数の少ない商の見積りを得るかということが研究の主眼であった（例えば、1), 2), 3)等）。しかし、実際に修正が起こる確率は極めて低いし、また、修正に要する時間はわずかである。上に示した4つのプロセスの中で最も時間を要するのは、2)の部分である。なぜならば、1)では単精度数（あるいは倍精度数）の除算が、3)では多倍長数の加減算が、4)ではメモリの入れ替えが、それぞれ行なわれるのに対して、2)では（単精度数）×（多倍長数）という計算が行われるからである。

本講演で提案するものは、2)のプロセスを省力化することによって計算速度を速め、しかも、省力化しない場合と同程度に誤差を抑えるアルゴリズムである。

2.1. 「正確な除算」について

b進n桁の浮動小数点数$X,\ Y$

$$X = x_1 b^{n-1} + x_2 b^{n-2} + \cdots + x_n, \tag{2.1}$$

$$Y = y_1 b^{n-1} + y_2 b^{n-2} + \cdots + y_n, \tag{2.2}$$

による除算$Z = X/Y$を考える。ここでは指数は考えず仮数のみを考えることにする。またX, Yは正規化されているものとする。すなわち、

$$0 < x_1, \quad 0 < y_1 \tag{2.3}$$

$$0 \leq x_i \leq b-1, \quad 0 \leq y_i \leq b-1, \quad i = 2,3,\cdots,n$$

を仮定する。これより、$Z = X/Y$は

$$b^{-1} < Z < b \tag{2.4}$$

の範囲の数となるから、Zを

$$Z = z_0+z_1b^{-1}+z_2b^{-2}+\cdots \equiv (\ z_0.z_1,z_2\cdots\ )_b \tag{2.5}$$

と表す。ここで商Zの各桁z_iの計算過程を示すと次のようになる：

$$\begin{aligned} z_0 &= \left\lfloor \frac{X}{Y} \right\rfloor, \quad r_0 = X-z_0Y, \\ z_1 &= \left\lfloor \frac{br_0}{Y} \right\rfloor, \quad r_1 = br_0 - z_1Y, \\ z_2 &= \left\lfloor \frac{br_1}{Y} \right\rfloor, \quad r_2 = br_1 - z_2Y, \\ &\cdots\cdots\cdots \end{aligned} \tag{2.6}$$

ここで$br_{-1} = X$と置くことによって、上式は次のように簡単化される：

$$z_i = \left\lfloor \frac{br_{i-1}}{Y} \right\rfloor, \quad r_i = br_{i-1} - z_iY, \tag{2.7}$$

$$i=0,1,2,\cdots$$

浮動小数点演算では、実際に必要なのは上位n桁であるから、式(2.7)の演算を$i=0,1,\cdots,n$について行なうと、

$$\begin{aligned} r_n &= (\cdots((((X - z_0Y)b - z_1Y)b - z_2Y)b \cdots - z_{n-1}Y)b - z_nY \\ &= b^nX - (b^nz_0 + b^{n-1}z_1 + \cdots + z_n)\ Y \end{aligned} \tag{2.8}$$

という関係式が得られる。

式(2.7)に従って商Zの上位桁$z_0,z_1,\cdots,z_n$を計算する方法を、ここでは「正確な除算」と呼ぶことにする。「正確な除算」を行なうとき、剰余r_iの計算は合計で$n+1$回行なわれることになるが、この計算では、$z_i \times Y$という演算、すなわち（単精度数）×（多倍長数）という演算が毎回行なわれている。したが

って、「正確な除算」全体では、nが十分大きいときおおよそn^2回の単精度乗算が行なわれることになる。以下、この演算量を減らすことを考える。

2.2. 「手抜きの除算」について

ここで、まずYの下位のi桁を切り捨てた値Y_iを定義する：

$$Y_i = Y - (b^{i-1}y_{n-i+1} + b^{i-2}y_{n-i+2} + \cdots + y_n). \tag{2.9}$$

この式の右辺を

$$Y_i = Y(1-c_i b^{-n+i}),\quad i=1,2,\ldots,n-1$$

と置けば、c_iは

$$0 \le c_i < b \tag{2.10}$$

を満たすことがわかる。

式(2.7)に示した「正確な除算」を行なう代わりに、次のような「手抜きの除算」を考える。まず、Zの初めのm桁、すなわち、$z_0, z_1, \cdots, z_{m-1}$を求めるときは、「正確な除算」と同じように剰余を1桁シフトした値をYで割り整商を求め、次に、z_mから後はYの代わりに$Y_1, Y_2, \cdots, Y_{n-m+1}$で割っていき、剰余の計算も$Y_i$を用いて行なう。このようにすれば、$Y_i$は$Y$より桁数が少ないので剰余の計算が省力化される。このとき得られた剰余を$\tilde{r}_i$で表し、各桁を$\tilde{z}_i$で表す。以上の過程を式で示すと次のようになる：

$$\begin{aligned} &\tilde{z}_i = z_i, \qquad \tilde{r}_i = r_i, \qquad i=0,1,\cdots,m-1, \\ &\tilde{z}_i = \left\lfloor \frac{b\tilde{r}_{i-1}}{Y_{i-m+1}} \right\rfloor,\quad \tilde{r}_i = b\tilde{r}_{i-1} - \tilde{z}_i Y_{i-m+1},\quad i=m,m+1,\cdots,n. \end{aligned} \tag{2.11}$$

以上より、式(2.8)と同様の式

$$\begin{aligned} \tilde{r}_n &= b^n X - (b^n z_0 + b^{n-1}z_1 + \cdots + b^{n-m+1}z_{m-1})\ Y \\ &\quad - b^{n-m}\tilde{z}_m Y_1 - b^{n-m-1}\tilde{z}_{m+1}Y_2 - \cdots - \tilde{z}_n Y_{n-m+1} \\ &= b^n X - \left\{ \sum_{i=0}^{m-1} b^{n-i}\tilde{z}_i + \sum_{i=m}^{n} b^{n-i}\tilde{z}_i(1-c_{i-m+1}b^{-n+i-m+1}) \right\} Y \end{aligned} \tag{2.12}$$

が得られる。

ここで、「手抜きの除算」による結果を

$$Z' = (\tilde{z}_0 . \tilde{z}_1 \cdots \tilde{z}_n)_b \tag{2.13}$$

と置き、このZ'と真の商$Z=X/Y$との差、すなわち誤差を解析する。式(2.12), (2.13)より

$$\tilde{r}_n = (b^n Z - b^n Z' + \sum_{i=m}^{n} c_{i-m+1}\tilde{z}_i b^{-m+1}) Y \tag{2.14}$$

であるから、

$$|Z'-Z| \leq b^{-n} (\tilde{r}_n Y^{-1} + \sum_{i=m}^{n} c_{i-m+1}\tilde{z}_i b^{-m+1}) \tag{2.15}$$

となる。ここで、$|\tilde{r}_n| < Y_{n-m+1}$に着目すると

$$0 \leq \frac{\tilde{r}_n}{Y} = \frac{\tilde{r}_n}{Y_{n-m+1}} \times \frac{Y_{n-m+1}}{Y} < 1$$

が得られる。また、

$$0 \leq c_i < b, \quad 0 \leq z_i < b$$

であるから、

$$\sum_{i=m}^{n} c_{i-m+1}\tilde{z}_i < (n-m+1)b^2 \tag{2.16}$$

を得る。したがって、誤差の上限は

$$|Z'-Z| < b^{-n}\left[1+(n-m+1)b^{-m+3}\right] \tag{2.17}$$

となる。ここで、mの値として$0 < m \ll n$かつ

$$(n-m+1)b^{-m+3} \ll 1, \tag{2.18}$$

となるようなものを選べば、次式の誤差評価を得る：

$$|Z'-Z| < b^{-n} (1+o(1)) \tag{2.19}$$

これはZ'はZに比べそのn桁目が高々1単位程度しか違わないことを示している。

以上の方法で除算を行なうと、単精度の乗算は、 $\tilde{z}_i \times Y$ $(i=0,1,\cdots,m-1)$の各々の計算においてn回、 $\tilde{z}_i \times Y_{i-m+1}$ $(i=m,m+1,\cdots,n)$において$n-(i-m+1)$ 回行なわれるから、合計で

$$D_{dc}(n) \equiv mn + \sum_{i=m}^{n}(n+m-i-1) = \frac{1}{2}n(n-1) - \left\{ \frac{1}{2}(m-1)(m-2) - nm \right\} \tag{2.20}$$

$$= n^2/2 + O(n)$$

回行なわれることになり、「手抜き」を行なわなかった場合のおおよそ半分の時間計算量で計算が完了す

ることになる。また、これは乗算の時間計算量とほとんど同じであることを次に示す。

2.3. 乗算との比較

式(2.1), (2.2)で与えられるX, Yの間で乗算を考える。$Z = X \times Y$の値を

$$Z = (z_1\ z_2\ \cdots\ z_{2n}\ .\ 0)_b, \tag{2.21}$$

$$z_k = (\sum_{i=1}^{k-1} x_i y_{k-i} + carry) \mod b$$

と表す。浮動小数点表示では実際に必要になるのは上位n桁だけであるから、 $z_1, z_2, \cdots, z_n$だけを計算すればよいことになる。しかし実際は、丸め誤差の影響を考慮して、 $z_1, z_2, \cdots, z_{n+m}$の上位$n+m$桁を計算したとすると、単精度の乗算は

$$M(n) \equiv \sum_{k=1}^{n+m} (k-1) = \frac{(n+m-1)(n-m)}{2} = \frac{n^2}{2} + O(n) \tag{2.22}$$

回行われることになる。これより、浮動小数点数の除算は「手抜き」を行なうことによって、乗算とほとんど同じくらいの手間で実行できることになる。

3. Ｎｅｗｔｏｎ法による除算

多倍長数Yの逆数Y^{-1}は、方程式

$$f(u) = 1/u - Y = 0 \tag{3.1}$$

の根であるから、上記方程式にNewton法

$$u_{i+1} = u_i + \Delta(u_i),\ \ i=0,1,\cdots, \tag{3.2}$$

$$\Delta(u_i) = u_i - Y u_i^2$$

を適用すれば、除算を行なわないで商$Z = X/Y$が求まることになる。以下ではこの方法の時間計算量を解析し、前記の方法と比較する。

いま u_i がY^{-1}のn_i $(n_i = 2^i n_0)$桁正しい近似であるとすると、二次収束性が保証されるためには、増分Δ_iは$2n_i$ $(=2^{i+1}n_0)$ で計算すれば十分である。そこで増分の計算を$2n_i$桁で計算すると、n桁の近似を得るためには、

$$\sum_{i=0}^{p-1} \frac{1}{2} K (2n_i)^2 = \frac{1}{2} K \frac{4(n^2 - n_0^2)}{3} \approx \frac{2}{3} K n^2, \tag{3.3}$$

$$p = \log_2 (n/n_0)$$

だけの計算量を要することになる。ただし、Kは反復一回当たりの乗算の回数とする。ここでKの見積りをK=1.5とし（二乗演算1回を乗算0.5回と数える）、反復終了後にY^{-1}の近似にXを掛けるのに要する計算量 $n^2/2$ を式(3.3)に加えれば、この方法で除算を行なのに要する時間計算量は、

$$D_N(n) \equiv \frac{3}{2}n^2 + O(n) \tag{3.4}$$

となる。これはdivide-and-correct法を「手抜きで」計算したときの三倍の計算量である。

4. 数値例

最後に、2つの計算法、すなわちdivide-and-correct法の「手抜き」による計算法と、 Newton法による除算法のcpu-timeとを乗算のcpu-timeと比較する。用いたソフトウェアは、著者らがＦＯＲＴＲＡＮで自作したものであり、 10進浮動小数演算を採用している。桁数nは8桁から32,768桁まで8桁ずつ増減できるようになっている。ここでは、被除数X、除数Yの値として乱数データを用い、桁数nを何通りか変えて計算時間を測定した。なお、mの値はどれも24としている。使用した計算機は、東北大学大型計算機センタのACOS 2020である。

＜参考文献＞

1) Knuth, D. E.: *The Art of Computer Programming, seminumerical algorithms, 2nd, ed.,* (1981).

2)戸田英雄、小野令美：高精度計算用の除算のアルゴリズムに関して、電総研彙報 **42** (1978), pp.66-71.

3) Semba, I.: An Algorithm for Division of Large Integers, JIP **9** (1986), pp.145-147.

表1.各種演算の実行時間と時間比

(1)$n=1024$

演算	時間(msec)	時間比	相対誤差
乗算	14	1	
除算(d-c)	16	1.14	5.44e-1024
除算(Newton)	81	5.79	7.61e-1025

(2)$n=2048$

演算	時間(msec)	時間比	相対誤差
乗算	56	1	
除算(d-c)	58	1.04	5.97e-2048
除算(Newton)	210	3.75	-1.77e-2048

(3)$n=4096$

演算	時間(msec)	時間比	相対誤差
乗算	223	1	
除算(d-c)	224	1.01	2.94e-4096
除算(Newton)	706	3.18	-3.25e-4096

(4)$n=8192$

演算	時間(msec)	時間比	相対誤差
乗算	890	1	
除算(d-c)	887	0.997	3.97e-8192
除算(Newton)	2669	3.00	-2.23e-8192

(5)$n=16384$

演算	時間(msec)	時間比	相対誤差
乗算	3604	1	
除算(d-c)	3544	0.980	1.77e-16384
除算(Newton)	10593	2.94	-1.33e-16384

(6)$n=32768$

演算	時間(msec)	時間比	相対誤差
乗算	14413	1	
除算(d-c)	14164	0.983	-7.10e-32769
除算(Newton)	42205	2.93	8.40e-32769