

多倍長浮動小数点演算プログラムの開発

平 山 弘

Development of the Arithmetic Package for the Multiple-precision Floating Point Number

Hiroshi HIRAYAMA

Abstract

The Arithmetic Package for the multiple-precision floating number is developed by FORTRAN 77 programming language. FFT algorithms is used to multiply two big integers in this package. By Using it we can multiply about 2000 figures integers about 8 times faster than usual multiplication methods. We can have a better computing circumstance because this multiple-precision floating number does not cause over-flow and under-flow errors.

1. は じ め に

計算機を使って数値計算を行なっているとしばしばオーバーフローやアンダーフローに出会う。これは言うまでもなく浮動小数点の指数部のビット数が足りないからである。一般にこのようなことが起こったとき、プログラムのミスの場合もあるが、そうでない場合も多い。このようなことが起こった場合、計算順序や計算方法の変更など非常に大変な作業となる。このような異常事態を回避するためにプログラムを大きく修正しなければならない。この修正行が本来の計算したい行数よりも長くなる場合もある。このような場合最も簡単な解決策は、浮動小数点の指数部のビット数を増やすことである。

指数部のビット数が増えれば、いろいろな有効な計算法が大変使い易くなる。このような計算法として代数方程式の解法の一つである Greaffe の方法¹⁾や非常に有効な数値積分法である Double Exponential 変換数値積分公式²⁾などがある。

このような問題点に対しては、松井・伊理³⁾や浜田⁴⁾の浮動小数点のフォーマットの提案がある。

工学上現われる数値計算では最終的に有効数字が5桁程度の結果が出れば十分な精度と言える。計算機内部で15桁程度の精度を持つ倍精度で計算すればこと

足りるように思われる。しかし、15桁の精度は5重根に近い根を持つ5次以上の方程式を解く場合には根の精度は3桁程度に低下するし、微分の公式をそのまま使って数値微分するとその微係数の精度は7.5桁に低下する。このようなことが重なると15桁の精度は十分とは言えなくなる。

このような問題が生じた場合、簡単に高精度で指数部が非常に大きい計算が出来れば大変便利である。このような目的のために、多倍長演算のプログラムを作成した。

高精度多倍長演算のプログラムはいままでに、多くの人によって発表されている^{5,6,7)}し、このような計算が出来る言語 REDUCE⁸⁾、UBASIC⁸⁶⁾および MACSYMA¹⁰⁾などが発表されている。しかし、これらのプログラムは手に入れる出来ないかまたは手に入られるとしてもその中で閉じた形になっており、上のような目的で簡単に使うことが出来ないことが多い。

また、このサブルーチンの開発には上のような目的の他に次のような利用も考えられる。

- (1) 非常に高い精度で計算出来るので、これを利用して数表として使う。
- (2) 計算精度が非常に高い場合、数値計算のアルゴリズムにどのように影響を与えるか調べる。
- (3) 現在の計算機では大型計算機でも最大で約33桁程度である。つぎの段階としてどのような浮動小数点フォーマットが良いか調べる。

A(0)	A(1)	A(2)	A(3)		A(N)
Exponent	Mantissa	Mantissa	Mantissa		Mantissa
	1	2	3		N

Fig. 1 Representation of Multiple-Precision Floating Point

このために、かなり完成度の高いサブルーチン・パッケージを目標に作成した。FORTRAN 77 では文字の取り扱いが出来るようになっていたため、このようなサブルーチンでも完全に文法の範囲内で記述でき、大変移植性の良いプログラムができた。

多倍長精度の数の積の計算には、高速フーリエ変換(FFT)が大変有効である^{11,12,13)}ことが知られている。ここで作成したプログラムでもFFTを使い高速化を計った。

2. 浮動小数点フォーマット

作成した多倍長浮動小数点は図1のように1つの浮動小数点を0から始まる添え字を持つ1個の単精度実数の配列で表現する。図の中の一つの四角の枠は単精度の一個の実数を示す。一個の枠に10進数の5桁を入れるようにした。このような計算では枠には整数型を用いるのが普通であるが、整数型には倍精度がないため、これらの多倍長の数の積を計算する場合のことを考えると、一個の枠に入れられる桁数は3桁程度になってしまい非常に効率が悪くなるため、枠を実数型にした。これはまた実数計算であるFFTを使って計算する場合、都合の良いことでもある。このフォーマットを使えばFFTを使わない筆算と同様な計算法で計算するときは約10万桁程度の数迄計算出来る。FFTを使った場合、後で述べるように誤差を分離する必要があるためその適用範囲がかなり小さくなる。

計算の種類によっては10進数でない方が便利ながあるため、一個の枠に入れる数値は2進数の14桁のように簡単に換えられるようプログラムを作成してある。しかし、以下の議論ではすべて10進数5桁が入って入っているものとして述べる。

数の符号は最初の仮数部A(1)の符号で指定するようにしてある。最初の仮数部以外は常に正の数になるようにした。指数部A(0)も単精度の実数で表現した。このため指数部は最大で約8桁となる。

多倍長浮動小数点は基本的には100000進数となる。これは正規化の計算で一個の枠に入っている数を割り算と余りを求める計算で分離する必要がなくなるためである。このため、計算精度は落ちるが、その分計算が高速になっている。計算の精度が必要ならば、もう一つ仮数部の枠を増やせば精度は確保できる。

仮数部は常に絶対値が0以上1未満になるように正規化される。すなわち、A(0)とA(1)の間に小数点があるとしている。正規化とはA(1)が0でないようにしたものである。0は配列の中をすべて0にして表現する。

以下で例を示す。

例1

A(0)=23.0 A(1)=12345.0 A(2)=34561.0 A(I)=0.0 Iは3以上の数)

このとき、この配列で表現されている数Fは

$$F = 0.1234534561 \times (100000)^{23} \\ = 0.1234534561 \times 10^{115}$$

例2

A(0)=-10.0 A(1)=-123.0 A(2)=123.0 A(I)=0.0 (Iは3以上の数)

このとき、この配列で表現されている数Fは

$$F = -0.0012300123 \times (100000)^{-10} \\ = -0.0012300123 \times 10^{-50} \\ = -1.2300123 \times 10^{-53}$$

例3

$$F = 3.141592653$$

を配列の中の表現に変換すること考える。このとき、Fは

$$F = 0.000031415926530 \times (10000)^1$$

となるので

A(0)=1.0 A(1)=3.0 A(2)=14159.0 A(3)=26530.0となる。

3. 計 算 方 法

多倍長の数の計算は加算, 乗算については筆算とほぼ同様な方法で行なっている。引算は引数の符号を変えて加算する方法で行なっている。除算は除数の逆数を計算し, 被除数と乗算を行なうことによって実行している。逆数の計算はニュートン法によって求めている。すなわち, a の逆数は反復公式

$$X_{n+1} = X_n \cdot (2 - a \cdot X_n)$$

を何回か繰り返すことによって, その逆数を計算する。最初の出発値は倍精度の実数の計算に変換して近似値を求め, それを出発値にしている。この反復公式は1回の反復でその精度が2倍になるので, 桁数の多い場合非常に有効である。また, この計算は出発値がどんな値でも a の逆数に近づくので, 途中の反復の計算の精度は最終的に得たい精度で計算する必要はない。反復途中の精度はその反復で得られる精度があれば十分である。

高精度の計算ではニュートン法は圧倒的に有利な計算法であるが, そんなに高精度でない場合には, テイラー展開法が有利である。これは高速計算機の除算回路にも利用されている方法¹⁴⁾でもある。

$$\frac{1}{1+X} = 1 - X + X^2 - X^3 + \dots$$

これによって, 計算を行なうのである。右辺はさらに因数分解することが出来るから, 2回の乗算と2回の加減算で右辺を計算できる。ニュートン法は1回の反復に2回の乗算と1回の加減算が必要である。上の式と同じ程度精度を得るには2回の反復が必要になるので, 4回の乗算と2回の加減算が必要となる。

このため, このルーチンを除算ルーチンに組み込むことを考えたが, その差はほとんどなかったため, ニュートン法だけを使うことにした。

同様に平方根の計算もニュートン法によって計算するようにした。この場合, 平方根を直接計算しないで, 平方根の逆数をニュートン法で計算する。

これは, 平方根の逆数のニュートン法の反復公式には, 除算が含まれないからである。すなわち, a の平方根の逆数はつぎの反復公式で計算出来る。

$$X_{n+1} = 0.5 \cdot X_n \cdot (3 - a \cdot X_n^2)$$

収束した結果に a を掛ければ a の平方根が得られる。この計算も途中の反復では計算結果の精度で計算すれ

ば計算量を減らすことができる。

4. 高速フーリエ変換の利用

多倍長数の計算は高速フーリエ変換 (FFT) を利用すれば, 理論的には高速に計算できることが知られている。文献 [11] [15] に述べられているように, フェルマーの小定理をベースにした, ガロア体を使ったFFTを使うことも考えられるが, この計算には適当なガロア体を選ばなければならない等の問題があるように思われたので通常のFFTを使うことにした。ガロア体を使う方法が駄目かどうかはもうすこし検討が必要である。

FFTで計算が出来るのは, 積分で定義されているフーリエ変換ではない。フーリエ変換の近似である有限フーリエ変換である。多倍長の計算で使う性質は2つの関数のフーリエ変換の積は, 二つの関数の畳み込みのフーリエ変換であるという性質である。多倍長の数値の積は畳み込みの計算に相当するので, 二つの数のフーリエ変換を求めその積を計算する。それをフーリエ逆変換で畳み込みを求めるのである。これは, 積分を使ったフーリエ変換についてであるが有限フーリエ変換でもこの性質は厳密に成り立つ。最終の結果には誤差が入るが, 結果は整数になるはずなので, 丸め操作によって厳密な計算が出来る。誤差が0.5以上になるような場合にはこの計算方法は使えない。

この方法による計算手順は

- (1) 被乗数である N 個の5桁の数値にさらに N 個のゼロをつけ加え $2N$ 個のデータに変換する。
- (2) 乗数に対しても (1) と同様な操作を行なう。
- (3) 上の2つのデータを $2N$ 個のデータとして有限フーリエ変換を行なう。
- (4) フーリエ係数どうしを掛け算する。
- (5) (4) の結果を逆フーリエ変換する。
- (6) 得られた結果を $2N$ で割る。
- (7) 誤差がなければ整数になるはずなので, 最も近い整数に丸める。

これによって厳密な計算可能になる。

FFTにもいろいろあるのでどのようなフーリエ変換法が良いか調べなければならない。一般的に言われるように, 次のようなルーチンを使うのが良い。

- (1) 実フーリエ変換
- (2) 基数と8, 4, 2を使うルーチン

Table 1. Timing of Multiplication of Two N Figures Integers by FFT and CONVOLUTION Methods

N	FFT (sec)	CONVOLUTION (sec)	FFT/CONV.
40	0.13	0.02	6.5
80	0.28	0.08	3.5
160	0.59	0.33	1.79
320	1.27	1.30	0.977
640	2.73	5.15	0.530
1280	5.85	20.28	0.288
2580	12.50	80.49	0.155

データが実数ならば実フーリエ変換を使うのが良い。実フーリエ変換は複素数フーリエ変換の半分の時間で計算できる。基数を2から4に変えると計算量は約半分になる。ただし、この場合はデータ数は4のべき乗でなければならない。基数が8になると、4と比較しほんの少しであるが改善される。さらに、細かい改良が出来る。三角関数の値を初めから計算しておくなどの様なことである。

今回作成した多倍長演算ルーチンには、(1)の条件は満たすが、(2)の条件は満たさない。基数は2だけを使うルーチンを使ったためである。このプログラムは、参考文献[16]を使って作成した。このルーチンを使って整数の積を求める時間を測定した。使ったコンピュータはPC-9801VX41でインテル 80286+80287が10 MHzで動作している。この結果を表1に示す。

この結果を見ると320桁あたりでだいたい同じ時間になり、2560桁の積では数倍の性能を発揮することがわかる。ここで作ったプログラムでは400桁を越えると自動的にFFTを使うサブルーチンを呼び出す様になってある。これは、FFTを使わないルーチンほどの様な桁数にも対応できるが、FFTを使う場合、2のべき乗の5倍桁数の場合しか使えないので、FFTをどこから使うべきかを正確に判断するのはかなり難しいためこのような概略の数値をとった。

5. 数値変換サブルーチン

このような多倍長浮動小数点演算ルーチンを使う上でその内部フォーマットに変換したり、逆に内部フォーマットから通常の文字列に変換することは、非

常に面倒な作業になる。このような作業を行なうために、パッケージでは二つのサブルーチンを準備した。

(1) 内部フォーマットへの変換ルーチン

この目的のために作られたルーチンがFXVALルーチンである。これは、

```
CALL FXVAL( AC, A, N )
```

という形で呼ばれる。ACは通常使う数値の文字列、Aはそれを格納する多倍長浮動小数点である。

例えば

```
CALL FXVAL('1.345656788645343E
+437', A, N)
```

と呼び出せば、配列Aに文字列で指定してある数値が内部フォーマットに変換されて入ることになる。ここで、NはAの添え字の上限の値である。

文字列として使える数値の表現は、ほぼ制限はない。

```
1.34567
1234
1.234E+20
1.234D+30
1.563Q-40
```

などが許される。Eの代わりにD, Qを使うことをも許している。

(2) 出力のための変換ルーチン

内部フォーマットを通常の形に変換するサブルーチンである。このために、準備されたサブルーチンはFXSTRである。FXSTRは

```
CALL FXSTR( A,N, FMT, AC )
```

という形で呼び出される。Aに入っている数値をFMTのフォーマットに従って変換し、文字列ACに出力されるサブルーチンである。FMTで指定できるものはほぼFORTRANのFORMAT文で指定出来るものと同じである。すなわち、

```
F60.50
E150.130E5
10PD180.130E5
```

などである。一般に

```
aPEb. cEd
```

これらを使った計算例を以下に示す。サラミンとブレントが発見したガウス・ルジャンドルの公式に基づく新しい π の計算法である。A と B の相加平均、相乗平均が非常に急速に収束することが分かる。A と B の出力フォーマットは 1PE68.60 である。最後の行はこの結果から得られた円周率である。

PI=3.14159265358979323846264338327950288419
71693993751058209749445

7. お わ り に

最近の半導体の技術革新でコンピュータは非常に多くの記憶装置を使って計算出来るようになってきている。記憶装置の容量の増加は今まで不可能だった三次元の物体の数値解析を可能にするなどデータを多数使う分野で大変有効に使われている。

このように記憶容量が十分にあるならば、ぜひ非常に広い指数部を持ち、非常に高い精度をもつ浮動小数点をサポートしてほしいものである。このような数値をサポートするならば、誤差を小さくするためのいろいろな技法が不要になる。この結果プログラムは短くなり、処理速度が速くなる可能性がある。

プログラムは精度やオーバーフローやアンダーフローから解放されるため、プログラムが途中で止まることが少なくなり、非常に使いやすくなるものになると考えられる。

計算機の容量はこの10年間で100倍程度増加しているが、浮動小数点のフォーマットは全く変化していない。

この多倍長の浮動小数点の演算ルーチンを作成した目的の1つに「使い易いコンピュータは高い精度で計算出来なければ成らない。」と主張したいからでもある。

参 考 文 献

- 1) A.A. Grau: "Algorithm 256, Modified Greaffe Method", Collected Algorithms from CACM.
- 2) H.Takahasi and M.mori: "Double Exponential Formulas for Numerical Integration", Pub. Reserch Institute for Math. Science, Kyoto University 9 721 (1974)
- 3) 松井正一, 伊理正夫: あふれのない浮動小数点演

- 算方式, 情報処理学会論文誌 Vol. 21, No. 4
- 4) 浜田穂積: 二重指数分割に基づくデータ長独立実数値表現法 II, 情報処理学会論文誌 Vol. 24, No. 2
- 5) W.T. Watt, P.W. Lozier and P.J. Orser: "A Portable Extended Precision Arithmetic Package and Library with Fortran Precompiler" ACM Trans. Math. Software, Vol. 2 (1976) pp. 209-231
- 6) 小野令美, 戸田英夫: 多倍長計算プログラムとその応用例, 電総研集報
- 7) 奥村晴彦: コンピュータアルゴリズム事典, 技術評論社 (1987)
- 8) A.C. Hearn: "REDUCE USER'S MANUAL Ver. 3.2", The Rand Corporation Santa Monia, CA 90406
- 9) 木田祐司: UBASIC86 解説書 第2.1版, 金沢大学 (1986)
- 10) R.J. Fateman: "The MACSYMA 'Big-Floating-Point' Arithmetic System", in Proceeding of ACM-SYMSAC '76, Yorktown Heights, NY, Aug. 1976.
- 11) A.V. エイホ, J.E. ホップクロフト, J.D. ウルマン (野崎, 野下訳): アルゴリズムの設計と解析, サイエンス社 (1977)
- 12) 日本物理学会編: スーパーコンピュータ (8章), 培風館 (1985)
- 13) 金田康正: 「ナノピコ教室」が生んだ円周率世界記録, bit, Vol. 20, No. 10, 共立出版 (1988)
- 14) 山田 博: コンピューター・アーキテクチャ, 産業図書 (1976)
- 15) 高橋磐朗, 室谷義昭: 数値計算とその応用, コロナ社 (1979)
- 16) W.H. Press, B.P. Flannery, S.A. Teukolsky, W. T. Vetterling: "Numerical Recipes", CAMBRIDGE UNIV. PRESS (1986)