計算機アーキテクチャ　128－1
ハイパフォーマンス　70－1
コンピューティング
（１９９８．３．５）

多倍長演算パッケージMPPACKの高速化

平山　弘

神奈川工科大学　機械システム工学科

多倍長演算パッケージ(MPPACK)は、標準的なC++言語で開発された整数、有理数、浮動小数点、複素数からなる可変精度演算パッケージである。これらのパッケージでは、非常に桁数の大きな数の乗算には、高速化のために高速フーリエ変換(FFT)を使っている。

多倍長数は、C++言語のクラスで表現されている。このため、多倍長数は、言語に組み込まれている整数型や浮動小数点数と同じように扱うことができるので、CやC++言語で書かれたプログラムを多倍長数のプログラムに変換するのは、かなり容易にできる。

いくつかの例を挙げて、このパッケージの説明を行う。

Speed-up of Mutiple-precision package MPPACK

Hiroshi Hirayama

Kanagawa Institute of Technology

A multiple-precision arithmetic system(MPPACK) is an arithmetic package of variable precision numbers consist of integer, rational number, floating point number and complex number has been developed using by standard C++ language. Fast Fourie transformation (FFT) algorithms is used to multiply two very high precision numbers in this package for speed-up.

These multiple-precision numbers are represented as classes in C++ language. Therefore it is easy to convert C or C++ source code to multiple-precision ones because these numbers can be treated as the intrinsic numbers such as int, float and double.

Some example programs are shown to illustrate the multiple-precision arithmetic package.

1. はじめに

高精度計算プログラムは、現在まで多くの人によって開発・作成されいる。公開されているプログラム[1][2][4][5][10][11]も多数存在する。しかしながら、これらのプログラムは、実際の計算に有効に使われているとはいえないようである。この理由として考えられるのは、これらのプログラムを利用するには、利用者が通常のプログラムを高精度計算プログラムのサブルーチンの呼び出し列に変換しなければならないことが考えられる。

たとえば、通常のFORTRANプログラムで

```
S=(A+B)*H/2
```

と簡単に書けるプログラムが

```
CALL ADD(A,B,WORK1)
CALL MUL(WORK1,H,WORK2)
CALL DIV(WORK2,2,S)
```

のように変換される。この作業は小さなプログラムでは、あまり問題にはならないが、少し大きなプログラムになると無視できない大変な作業となる。

この問題を解決するために、最近の言語（C++言語[3]、FORTRAN90など）の関数およびオペレータのオーバーローディング機能により、解決できる。Watt等[11]は、プレ・コンパイラを準備して、この問題の解決を図っている。

このような機能を使うことによって、この問題が一応解決すると、次に問題となるのが計算精度の問題である。高精度の計算は、通常の精度の計算と比べると非常に多くの計算時間を必要とするので、できるだけ計算量を少なくし、高速化したいと考えるのは当然である。その中で最も無駄と思えるのが、必要以上の計算精度で計算することである。

この問題に関しては、多くの多倍長演算パッケージでは、1語長と多倍長数との四則演算等は準備し、計算する数値の片方の精度が1語長で表現できる場合には高速化ができるようになっている。

本論文では、計算する数値が両方とも多倍長の数値であるが、その精度が異なるような場合も扱えるような可変精度の多倍長精度のパッケージを開発し、無駄な高精度計算を省き、簡単なプログラムでその効果を調べた。

この演算パッケージには、高速フーリエ変換(FFT)を使った乗算、二乗を組み込んだ。特に乗算では、精度の異なる多倍長の計算をできるように改良した。

また、Intel社のプロセッサ用に一部アセンブラで記述して、FFTでは高速化できない低精度の計算の高速化を行った。

この研究で使用したC++言語は、主にBorland C++ 5.0J+TASM、Visual C++ 5.0である。計算機は、主にパソコンのNEC PC-9821 RvⅡ26（Pentium Ⅱ 266MHz）を使用した。

2. 可変精度の多倍長数

精度を可変にするために、例えば浮動小数点数に関しては、C++言語のクラスを使って、次のような形式にした。

```
class long_float{
    int     alloc_siz  ;   //  配置されたmantの大きさ
    int     act_prec   ;   //  実際に使われているmantの大きさ
    int     sign  ;        //  符号
    int     expp  ;        //  指数部分
    short   *mant ;        //  仮数部分
} ;
```

符号は、signに入る。signは、負の数、ゼロ、正の数ならば、それぞれ－１、０、１となる。数値の０は、sign=0で表現と定義される。仮数部分は、10000進数で、以下の様なalloc_siz個の配列で表現される。

mant[0]	mant[1]	mant[2]	mant[3]	...	mant[alloc_siz-1]

また、変数act_precは、この変数に値が代入されたとき、０でない数値の桁数が入る。

例えば、このような形式で表した数値の例を示す。

$F = 123456789012345$

を高精度浮動小数点数の形式で表すとき、

$F = 0.0123\ 4567\ 8901\ 2345 \times 10000^4$

と変形できるので、

act_siz=4, sign=1, expp=4, mant[0]=123, mant[1]=4567, mant[2]=8901, mant[3]=2345

となる。alloc_sizが、４以上であれば、この例のように値が入る。mant[i](i>3)には、何が入っていてもかまわない。0が入っていると解釈される。すなわち、上の数値は

123456789012345.000000000000000000000000...

のように無限に0が続く数値として計算される。

このように定義すると精度の制御が複雑になる可能性があるが、今回のプログラムでは、以前の版[5]と互換性が保たれ、単純である以下の精度制御方法を使った。

一個のシステム変数(prec)で制御する方法である。ある多倍長数を表す変数の仮数部の配列の大きさは、その変数の宣言時のこのシステム変数の大きさになる。また、計算精度は、途中の計算結果を含め、すべて、このシステム変数の精度で行われる。対象となる変数の精度が小さいならば、0を追加して、その精度に変換し、精度が大きいならば、丸め処理によって、その精度に変形して計算結果を得る。最終結果を代入する変数がこのシステム変数より大きいならば、その精度の結果がその変数に入る。この場合、変数act_precはprecになる。逆に代入する変数の精度が計算精度より小さい場合、丸め処理によって、その変数の精度に変換し代入する。

システム変数(prec)の大きさは、関数set_long_float_prec関数を使って制御する。このような制御した場合、システム変数(prec)の初期値になるc

整数型の変数は、計算結果がその精度を超えると自動的に精度が２倍になるようになっているので、基本的には精度制御の必要はない。しかし、精度が一旦変更されると、小さくなることはないので、メモリー効率などを考えて強制的に小さくするような場合、精度制御を行う。

FORTRAN90を利用して同様の試みを行ったが、FORTRAN90では、C++言語のコンストラクターに相当する部分がプログラム可能でないだけでなくデストラクターの機能がないため、使用方法が非常に複雑になる。このため、今回はC++言語のみで作成した。

3. 可変精度プログラムの利用

可変精度のプログラムを使うことによって、計算精度を計算で要求される精度で計算できるだけでなく、必要とするメモリーを最小限に抑えることができる利点がある。これは、非常に大きな桁数で計算する場合には、非常に重要である。

固定精度のプログラムを利用する場合、プログラムはコンパイルする前にその精度を設定する必要があり、精度を変更する場合、プログラムを修正して再コンパイルしなければならない。これは、面倒であるだけでなく間違い易いため、プログラマに大きな負担を掛

けることになる。また、問題のパラメータが変わったり、計算環境が変わった場合にも、精度を変更するために再コンパイルしなければならなくなる。

可変精度にすると効果的に計算できる具体的例をあげる。数値解析でよく使う、Newton法が効果的に実行できる良い例である。一般にNewton法は、$f(x)=0$の零点を求めるのに、漸化式

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3.1}$$

を繰り返し計算して、解を求める。x_nは$f(x)$の零点の近似値であるから、$f(x)$の値は、x_nに比べ非常に小さくなる。このため、$f(x)$の最終計算精度は、(3.1)式の半分の精度で済む。このことを利用すると、いろいろな計算を高速に計算できる。このような精度制御は、可変精度の本プログラムでは、自動的に行う。本パッケージでは、Newton法は、逆数すなわち割り算、平方根の計算に使われている。数値aの逆数の計算は、Newton法

$$x_{n+1} = 2x_n - ax_n^2 \tag{3.2}$$

と記される場合が多いが、このように計算すると、計算精度が半分で済むという効果がでなくので、

$$x_{n+1} = x_n - x_n(ax_n - 1) \tag{3.3}$$

として計算する。同様に平方根の逆数は、

$$x_{n+1} = (3x_n - ax_n^3)/2 \tag{3.4}$$

を

$$x_{n+1} = x_n - x_n(ax_n^2 - 1)/2 \tag{3.5}$$

として、計算できる。このようにして、計算を高速化できる。

400桁の平方根の計算で、(3.4)式で計算する場合と、(3.5)式で可変精度プログラムで計算したときの時間は、それぞれ2.34msec、1.87msecであった。可変精度プログラムを利用することによって、約20%の高速化することができた。多くの例では、この程度速くなるが、中には、ほとんど変わらない場合もあった。

4. 高速フーリエ変換を使った高速化

高速フーリエ変換(FFT)を使うと高精度の数値を高速に乗算が行うことができる。MPPACKの乗算、二乗計算に、このアルゴリズムを使っている。計算する数値の精度をそれぞれn、mとすると、低精度では、通常の乗算法（計算量$O(nm)$）で計算し、ある程度以上の精度では、計算量$O(\frac{n+m}{2}\log(\frac{n+m}{2}))$のFFTを使うアルゴリズムを使う乗算法で計算するように作成した。

表1 乗算の実行時間（単位秒）

10進の桁数	FFT	通常
10,000	0.047	0.141
20,000	0.094	0.516
40,000	0.187	2.125
100,000	0.422	13.453
200,000	1.109	53.859
400,000	2.391	215.984
1,000,000	4.937	1830.375

同じ桁の数値の掛け算を行うとき、Pentium Ⅱ 266MHzの計算機では、10000進数で約380桁（10進数で約1500桁）以上のとき、FFTを使った計算法を使い、それ以下では通常の乗算法を使っている。二乗の計算では、通常の計算法が掛け算の約半分の計算時間計算できるので、その値は、10000

進数で約410桁（10進数で約1600桁）であった。計算精度を上げていくと、途中再び通常の計算法が一旦速くなるが、計算時間にあまり変化しないので、その部分でも、FFTを使った計算法を適用している。通常の計算法とFFTを利用した計算法の境界の桁数は、高速計算機ほど、大きくなる傾向がある。最初にこのプログラム作成した計算機(Intel社i286+287)では、乗算の限界が10進数で約700桁、二乗の限界が約800桁であった。

Intel社のi386を使い、C++言語で作成した場合、精度nの数値の乗算を計算量$O(n^{\log_2 3})$で計算できるKaratsuba[7]のアルゴリズムが有効な精度も存在するが、現在の高速な計算機では、通常の乗算法がキャッシュ等の効果で非常に速くなり、Karatsubaのアルゴリズムが有効な精度領域が存在しなくなった。このため、本パッケージでは、このアルゴリズムは多倍長数の乗算法としては組み込んでいない。複素数の乗算法としては、単純な使い方であるが利用している。

表1に、FFTを利用した場合と通常の計算法で計算した場合の計算時間を示す。使用したデータは、$\frac{1}{3}$と$\frac{1}{7}$である。10000桁以下の場合には、計算時間は、計算機のタイマー精度以下なので省略した。100万桁の場合、300倍以上高速であることがわかる。

5. アセンブラを使ったプログラム

多倍長計算にアセンブラを使えば、効果的に計算できることはよく知られている。アセンブラを使った場合の最大の問題は、プログラムがコンピュータの機種に依存することである。この節以前までのプログラムは、コンピュータに依存しない形で作ることができたが、アセンブラを使った場合、機械に依存することになる。

今回は、機種として、Intel社のPentiumⅡを利用し、C++言語とインライン・アセンブラ機能を使って作成した。このプログラムは、代表的な二つC++コンパイラ（Borland C++ 5.0+TASMとVisal C++）でコンパイル可能である。PentiumⅡの特有の命令は使用していないので、Intel社のi386以上の機種であれば、どの機種でも動作する。

ここで使用した多倍長の数値は、基数として10000進数ではなく、2^{32}進数を使うことにした。構造は、2節で示した構造とほぼ同じで、仮数部分を32ビットの配列で表現するように変更した。アセンブラでは、高級言語では使えない32ビット整数同士の積を64ビットで得る計算ができたり、桁上げをハードウエアで実行できるので、このような構造体で実現可能となる。入出力関連のプログラムは、2^{32}進数と10000進数の相互変換関数を準備することによって、これまで、開発してきたプログラムを流用した。

このプログラムを利用すると、1000桁程度の計算では、5倍程度の速度向上が得られるが、100桁程度では、2.5倍程度の速度向上が得られるだけである。これは、実際の計算より関数呼び出しのオーバーヘッドが相対的に大きくなったためだと考えられる。これを速くするには、関数全体をアセンブラで書く必要がある。

さらに、計算精度を上げると、FFTを使う乗算法が効率的な領域になる。FFTを使う乗算法は、浮動小数点数を使うアルゴリズムが主体となるため、2^{32}進数を使う利点が小さくなる。

2^{32}進数で計算すると、その計算結果を10進数に変換する問題が生じる。10進数で9600桁程度の数値ならば、約1.4秒変換でき、それほど時間がかからないが、10進数で48万桁に相当する2^{32}進数を10000進数に変換するには、単純なアルゴリズムで計算すると、1時間以上も計算時間がかかる。改良を加えても約2分19秒の時間に改良することが出来たが、この変換の時間は、これと同じ精度の数値の平方根の計算時間(48万桁で6.6秒)の10倍を

超える時間になる。円周率の計算などのように、最終的に10進数で計算したすべての桁を出力するような計算では、2^{32}進数で計算する利点は、あまりないように思える。10進数ですべての桁数を出力しないような計算では、2^{32}進数で計算する利点は、メモリーの利用効率、計算の途中で生じる低精度の計算が高速できるなどの利点がある。

問題にもよるが、ある程度大きなプログラムを作成し、使用したみると、アセンブラ化によって、およそ3倍程度高速化することができたと考えている。

アセンブラを使ったとき、FFTが有効な領域も変わってくる。FFTの乗算が高速になるのは、2^{32}進数で800桁(10進数で約7600桁)である。二乗の計算では、通常の計算法が掛け算の約半分の計算時間計算できるので、2^{32}進数で約1400桁（10進数で約13000桁）であった。

6. 終わりに

多倍長数の演算プログラムMPPACKの3つ改良法について述べた。可変精度化、FFTを使用した高精度数の高速演算、2進数を利用した部分的アセンブラ化による高速化である。

可変精度の導入では、メモリーの利用効率、無駄な高精度計算を省くことができる。さらに、プログラムの作成が容易になる利点がある。

FFTを使用した計算法は、数千桁以上の非常に高精度な計算を行うのに役立つ。

アセンブラ化は、通常の計算としては高精度であるが、多倍長精度の計算としては、低精度の計算の部分で高速化を行うことができた。

参考文献

[1] Brent, R. P., A Fortran Multiple-Precision Arithmetic Package, ACM Trans. Math. Software,4(1978),57-70

[2] Brent, R. P., ALGORITHM 524 MP, A Fortran Multiple-Precision Arithmetic Package[A1], ACM Trans. Math. Software,4(1978),71-81

[3] Ellis, M. A. and Stroustrup, B., The Annotated C++ Reference Manual, AT&T Bell Laboratories, Murray Hill, New Jersey(1990)

[4] 平山　弘，多倍長計算プログラムパッケージMPPACK 利用の手引き、東京大学大型計算機センター(1992)

[5] 平山　弘，C++言語による高精度計算パッケージの開発,日本応用数理学会，Vol. 5, No.3, pp. 123-134, 1995.9

[6] 伊藤　正俊，丸め誤差の評価できる多重精度演算ツール，名城大学理工学部研究報告，30(1990)，246-251

[7] Karatsuba, A. and Ofman, Y., Multiplication of mutidigit numbers on automata, Doklady Akad. Nauk SSSR, Vol.145, pp.293-294(1962)

[8] 木田祐司，UBASIC86多倍長計算用BASICユーザーズマニュアル，日本評論社(1992)

[9] 大中幸三郎，安井裕，誤差評価可能な多重精度演算，情報処理，15(1974)，110-117

[10] Smith, D.M., A FORTRAN Package For Floating-Point Multiple-Precision Arithmetic, ACM Trans. Math. Soft.,17(1991)273-283

[11] Watt, W.T., Lozier, P. W. and Orser, P. J., A Portable Extended Precision Arithmetic Package and Library with Fortran Precompiler, ACM Trans. Math. Soft.,2(1976)209-231