計算数理工学論文集 Vol. 2 (2002 年 7 月), 論文 No. 02_070312 JASCOME

64bit 計算機への多倍長計算環境の実装

Implementation of Multiple-Precision Arithmetic on 64bit Computing

藤原　宏志 [1)]，磯　祐介 [2)]

Hiroshi FUJIWARA, Yuusuke ISO

1) 京都大学大学院 情報学研究科　(〒 606-8501　京都市左京区吉田本町,　E-mail: fujiwara@acs.i.kyoto-u.ac.jp)
2) 京都大学大学院 情報学研究科　(〒 606-8501　京都市左京区吉田本町,　E-mail: iso@acs.i.kyoto-u.ac.jp)

The aim of the research is to construct a multiple-precision arithmetic environment which is designed for 64-bit computing environments. We choose the Alpha architecture as a typical RISC 64-bit architecture, and implement a fast multiple-precision arithmetic environment. We write the basic routines in the Alpha assembler language and realize fast computing. Our package can be used in C, C++ and Fortran. We also provide the polymorphic interface to our environment in the programming language C++, and we realize easy access from end users.

Key Words: 多倍長計算，浮動小数点数，丸め誤差，大規模数値計算，64bit コンピューティング

1. 序

本研究は，多倍長数値計算が単に研究室での数値実験で使われるのみならず，計算力学の現場で利用されるための高速化を目指した計算環境の整備を行うものである．

工学や物理などに現れる物理現象の多くは，偏微分方程式や積分方程式などを用いて記述される．通常これらの問題に対して厳密解を記述することは困難であり，数値計算によって近似的に数値解を構築する手法がとられることが多い．現在広く使われている計算機上では，数値計算に際して，実数は有限桁の浮動小数点方式で表現されて扱われる．今日では，IEEE754[(4)] に従い，10 進法で約 15 桁の精度を有する倍精度・浮動小数点方式に基くものが一般に用いられるが，そこでは真値と計算機上で近似されて用いられる数値との間に丸め誤差とよばれる誤差が混入する．通常の数値計算では，丸め誤差の増大が問題とならないような安定なスキームを採用して数値計算を行い，理論的な数値解析の研究によってそのスキームの妥当性が議論されている．

しかし逆問題とよばれる問題は，多くの場合は Hadamard の意味で非適切であり，通常の直接的離散化で得られる数値計算スキームは不安定となる．このような問題は数値的に不安定 (ill-conditioned) と呼ばれ，計算過程で丸め誤差が増大して数値計算が破綻してしまい，信頼できる数値計算は不可能であるとされてきた．しかし多倍長計算によって，有効桁数を十分確保し，丸め誤差を任意に小さくすることによってその増大が計算結果として必要な桁に達しなければ，仮想的に丸め誤差のない数値計算が実行でき，数値的に不安定な問題の数値計算が可能となると考えられる．

このような背景にあって，我々は数値的に不安定な問題，特に偏微分方程式などで記述される非適切問題の数値計算を実現するために，大規模な数値計算を扱い得る高速な多倍長計算環境の整備が必要と考えた．これまでにも，FORTRAN や C 言語からの利用を想定した多倍長計算環境が構築され，利用されてきた．しかし，これらはいずれも 32bit 計算環境で動作することを想定して設計されており，演算速度の高速化を考えると今後普及する 64bit 計算環境のハードウエア性能を十分に活かすことができないという欠点がある．また数論などの分野で広く使われている UBASIC では，扱えるメモリの量に制限があり，科学技術計算に現れる大規模な行列などを扱う数値計算には向いていないという欠点がある．このような事情から，新たな設計が必要と考え，高速の多倍長計算環境の構築を行った．

本研究では，64bit 計算環境に適した多倍長計算環境の設計を行い，64bit 計算環境の一つである Alpha アーキテクチャ [(2, 3)] を対象に実装を行い，数値的に不安定な問題の数値計算を行ってその有効性を示した．本論文では，§2 で 32bit 計算環境との比較において 64bit 計算向けの多倍長計算環境の設計を述べる．§3,4,5 において多倍長数のデータ構造と演算のアルゴリズムを論じる．最後に，§8 において従来より構築されてきた多倍長計算環境と比較して，本設計の環境の高速性を示す．

2. 多倍長計算環境の設計

計算機上で数値を処理するに際しては，計算機の内部では大きく分けて整数型と浮動小数点型の二つの型のいずれかで数値は取り扱われている．整数型の特徴は演算が厳密 (exact) である (丸めが発生しない) ことであり，その演算は CPU 内部の ALU (Arithmetic Logical Unit) で行われる．一方，浮動小数点型は FPU (Floating Point Unit) において処理され，乗除算を高速に行なえるという特徴を持つ．

多倍長計算環境を構築するにあたり，現在普及している 32bit 計算機の演算精度と速度を検討すると，浮動小数点数とその演算を中心に用いるのが有利である．現実にも，この設計方針に従った多倍長計算環境が構築され，利用されている [8]．今日の計算機の多くは IEEE754 [4] に準拠する浮動小数点演算の機能を有するため，この設計方針は移植性が高いという利点も兼ね備えている．しかしこの設計では，メモリの利用に関して無駄が生じる．浮動小数点数として現在広く利用されている倍精度数は全体で 64bit の大きさを持ち，仮数部が 52bit を占める．乗算における桁の溢れを考慮すると，多倍長数を記憶する際にはその半分である 26bit しか利用することができず，メモリ全体の 26/64 ∼ 40% しか有効に利用されておらず，約 6 割のメモリは無駄になっていることになる．これは，大規模な数値計算を行うに際し，非常に不利であると考えられる．

一方，今後普及することが考えられる 64bit 計算環境では，整数型が浮動小数点型よりも精度がよいため，多倍長演算環境を設計するにあたって多倍長数のデータ構造と演算アルゴリズムに整数型を利用するほうが有利である．また整数型では，メモリ中の全ビットに有効な情報を保持させて利用することが可能であり，より少ないメモリで多倍長数を表現することができる．そのため，大規模な数値計算が可能になると同時にメモリ・アクセスの回数を減らすことができ，演算の高速化にも貢献すると考えられる．

3. 多倍長数のデータ構造

前節で示した設計方針に基づき，本研究では，多倍長数型のデータ構造を次のように定めた．まず，扱う実数を $(-1)^s \times 2^e \times 1.F$ の形に正規化する．ここで，s は 1bit, e には 63bit, $F = f_1 f_2 \cdots f_n$ には $64 \times n$ bit を割り合てる．ここで，n は，コンパイル時にユーザの要求した桁数に応じて決定される整数値である．この正規化された数を，図 1 の形式で，64bit 符号無し整数の配列としてメモリ中に保持する．従って，図 1 に示された多倍長数は，

$$(-1)^s \times 2^e \times \left(1 + \sum_{i=1}^{n} f_i 2^{-64i}\right),$$

という値を持つことになり，その精度は，10 進でおよそ $\log_{10} 2^{64\times n+1} \sim 19.26 \times n$ 桁となる．

4. 多倍長数の演算アルゴリズム

本節では，多倍長数の演算に用いたアルゴリズムを示す．

まず，多倍長数の四則演算，多倍長数と整数の乗算と除算は Alpha アーキテクチャのアセンブリ言語で記述した．その中では，C 言語や Fortran からは直接利用できない補助的な命令をいくつか利用し，64bit 整数の全ビットを有効に利用することを可能としている．例えば，64bit 整数同士の積は 128bit になるが，その上位 64bit を計算するために `umulh` インストラクションを利用している．また，メモリからの値のロードや整数の積など，その演算結果を利用するのにレイテンシ (latency) がある場合には，ソフトウエア・パイプライン (software pipelining) 技術 [3] などにより，ストールの発生を最小限に抑えるようにしている．特に整数の積では大きなレイテンシがあるため，多倍長数の積のアルゴルズムのループでは 2 段に渡るソフトウエア・パイプラインを用いている．

今回対象としている Alpha アーテキテクチャは，整数の演算に関して 2 つの演算を同時に行えるスーパ・スカラ型の CPU である．本設計では，この 2 つのパイプラインに対して同時に命令を発行して並行処理できるよう，命令の順序を工夫している．

多倍長数の加減算に関しては，初等的な筆算による手法を用いた．多倍長数の積に関しては，筆算による初等的な手法に加えて，従来より Karatsuba-Offmann のアルゴリズム [5]，フーリエ変換を利用するものなど，高速な手法がいくつか提案されてきた．これらの手法は，特に桁数が十分に大きくなるに従ってその効果が現れる．本研究の多倍長計算環境の主目的は整数論への適用ではなく，非適切な微分方程式や積分方程式の離散化で現れる数値的に不安定な数値計算を念頭に置き，100 桁から 500 桁程度の比較的小さな桁での数値計算を高速に行うことを想定している．筆算による手法と Karatsuba-Offmann のアルゴリズムを実装して演算速度の比較をした結果，この範囲の桁数では筆算によるほうが高速であったため，今回はこちらを採用した．除算には Ozawa による手法 [6] を採用している．その演算コストは乗算と同じく $O(n^2)$ である．

5. 数学関数とその他の数値計算アルゴリズム

本環境は，多倍長数の四則演算に加え，三角関数，指数・対数関数などの数学関数を実装しており，通常の科学技術数値計算を行うには十分な機能を有する．これらの数学関数は，還元公式により引数を適切な範囲に変換し，Taylor 展開などの級数展開により計算を行う [1, 11]．引数は §2 で定義したデータ構造を持つが，還元公式により引数を適切な範囲に変換した後，特に級数の計算に際しては，固定小数点形式として扱うほうが適切な場合が多い．そこで，級数の内部では固定小数点形式のデータ型を定義し，それらの加減乗算をアセンブラで記述した．数学関数は C++ 言語で記述されており，これらの固定小数点型のデータ構造を扱うための関数や浮動小数点形式のデータ構造に対する四則演算を呼び出して計算を行っている．

C 言語などで用意されている数学関数に加え，科学技術数値計算で有用と考えられる QR 分解や LU 分解のための

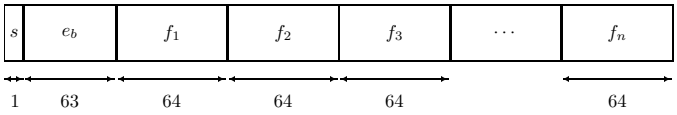

Fig. 1　多倍長数のデータ構造

計算ルーチンも実装している．なお，ここで用いたアルゴリズムは特に多倍長計算用のために設計したものではなく，汎用的に用いられているもの [7] を実装した．

6. インターフェース

多倍長数の出力に関して本環境は，C++ 言語での利用を想定しており，端末やファイルへの出力を実装している．その際，桁数や出力の書式は，C++ 言語の基底型に対するのと同じく，マニピュレータによって指定することができる [10]．

多倍長数の値の代入について，本環境は非常に大きな特徴を持つ．通常，多倍長型の変数を宣言して初期化もしくは代入を行うには，基底の型や同じ多倍長数型を利用する．本環境はこれに加え，文字列による初期化を可能としている．通常，C 言語や FORTRAN で利用可能なパッケージでは，10 進定数を記述した文字列による初期化が可能であることが多い．本環境では 10 進定数だけでなく，円周率 π などの組み込み型の数学定数や，それらの四則演算を記述することができる．

具体的に述べると，a, b, c を多倍長数型と宣言したとき，以下に挙げる方式での代入が有効である．

```
a = "3.14"
b = "#PI/2"
c = "(1.23-4.5)*(#PI+#E)"
```

右辺に現れる文字列は，再帰下降法 (recursive descent manner) により構文解析が行われ，ユーザが要求する精度での計算が行われて左辺に示す変数に値が代入される．ここに挙げた本環境の機能は，単に 10 進法でしか値を記述できない環境に比較して，ユーザプログラムをより簡潔に記述することを可能にしている．

他の FORTRAN90 や C++ 言語向けの機能と同様，演算子や入出力に関する演算子の多重定義も提供している．これにより，既存の C 言語で記述されたプログラムを容易に移植でき，かつ”型”の変更に対してロバストネスをもち，不正なメモリ・アドレスへのアクセスを防いでいる．

7. 多倍長計算環境の構成

本環境を実現する多倍長計算環境は，ライブラリと C++ 言語用に記述されたファイルから構成される．C++ 言語用に記述されたファイルには，多倍長数型のデータ構造の定義と演算子のオーバーロードを定めるもの，数学関数を記述したもの，そして数値計算に有用な外部ルーチンを記述したものの 3 つに分割されている．

現在のところ，Alpha アーキテクチャを対象に実装されており，Alpha 計算機上の Tru64 および Linux オペレーティング・システム上の GNU C++ コンパイラで ISO/ANSI Standard に準拠しているものでの動作を確認している．

8. 演算の速度

本設計の多倍長計算環境の演算部分は，C 言語の外部関数の集まりであるライブラリとして提供している．このライブラリは単独では動作せず，その速度性能はユーザ・プログラム，コンパイラ，ハードウエア等の計算環境を構成する他の要因にも依存する．本節ではライブラリを「問題解決のための計算環境の一部」として位置づけ，演算時間の測定を行った．

ベンチマーク問題として，1999 年に発表された多倍長演算に関する問題 [13] を採用した．具体的には、次の漸化式を満す $\{a_i\}_{i=1}^N$ を多倍長計算により数値的に求め、文献 [13] の結果との比較を行った。

$$
\begin{aligned}
a_0 &= 1, \\
a_{2k-1} &= 0 \\
a_{2k} &= -\frac{N}{2k}\sum_{j=1}^{k}\frac{1}{2j+1}a_{2(k-j)}, \\
&(k = 1, 2, 3, \cdots, \left[\tfrac{N}{2}\right])
\end{aligned}
$$

ここで生成される $\{a_i\}_{i=1}^N$ は，Chebyshev の数値積分公式

$$\int_{-1}^{1} f(x)dx \approx \frac{2}{N}\sum_{i=1}^{N} f(x_i)$$

の標本点 $\{x_i\}_{i=1}^N$ の根を与える代数方程式の係数で，

$$F_N(z) = \prod_{i=1}^{N}(z - x_i) = \sum_{i=0}^{N} a_i z^{N-i}$$

である。

計算時間の比較にあたり，文献 [13] で挙げられている次の 3 つの計算環境を対象に選んだ．

計算環境 A PentiumPro (200MHz, SPECint95 8.7, SPECfp95 6.8) と MPPACK[12]

計算環境 B POWER2(66MHz, SPECint95 3.41, SPECfp95 10.2) と Fortran90 で記述したプログラム

計算環境 C POWER2(66MHz) 48 台並列 と Fortran90 で記述したプログラム

これに対し，本研究で示す多倍長計算環境は次の通りである．

計算環境 D Alpha 21164A (500MHz, SPECint95 15.0, SPECfp95 20.4) と本設計の多倍長計算環境

多項式の次数	演算桁数	環境 A [13]	環境 B [13]	環境 C [13]	環境 D (本設計の環境)
4096	905	0.91 時間	1583.2 秒	49.7 秒	22.4 秒
8192	1805	13.3 時間	20724.3 秒	523.1 秒	174.5 秒
16384	3610	213.4 時間	—	4554.8 秒	1394.1 秒
20480	4520	561.1 時間	—	7255.2 秒	2716.9 秒

Table 1　計算に要した時間

各々の環境で数値計算に要した時間を表 1 に示す．文献 [13] では，漸化式の並列計算アルゴリズムと多倍長計算の分散処理によって演算時間の高速化を試みているが，本環境を利用することにより極めて短時間で結果が得られていることがわかる．

なお，本環境の多倍長計算環境では MPI (Message Passing Interface) を利用した並列計算が実行できることを注意しておく．しかしこの点については，多倍長数のクラスを MPI 環境に適する形へと改良し，かつ行列計算などを並列処理するためのルーチンを別途準備することが今後の課題として挙げられる．

9. 結言

本研究では，今後の普及が予想される 64bit 計算機上の性能を活かして高速動作を実現する多倍長計算環境の設計を行い，64bit 計算環境の一つである Alpha アーキテクチャ上に実装を行った．また，インターフェースおよび基本的な数学関数の実装を行い，通常の数値計算を行うには十分な機能を有するに至った．今後は，その他の有用な関数の実装を行うと共に [9]，IA-64 などの他のアーキテクチャへも実装を行い，環境の整備を図る予定である．

なお，本研究で構築した高速多倍長数値計算環境は，インターネットを通して配布している [14]．

謝辞

本研究を行うにあたり，京都大学名誉教授の一松信先生より貴重なご助言を頂きました．この場をかりて厚く御礼申し上げます．

参考文献

(1) M. Abramowitz, I. A. Stegun, Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables, Dover (1970).

(2) Alpha Architecture Handbook, Compaq Computer Corporation (1998).

(3) J. S. Evans, R. H. Eckhouse, Alph® RISC Architecture for Programmers, Prentice Hall (1999).

(4) IEEE Standard 754-1985 for Binary Floating-Point Arithmetic (Reprinted in SIGPLAN, **22** (1987) pp.9–25).

(5) D. E. Knuth, The Art of Computer Programming 3rd ed., Vol. 2, Seminumerical Algorithm, Addison Wesley (1998).

(6) K. Ozawa, *A Fast $O(n^2)$ Division Algorithm for Multiple-Precision Floating-Point Numbers*, J. of Information Processing, **14** (1991) pp.354–356.

(7) W. H. Press et al, Numerical Recipes in C, Cambridge University Press (1997).

(8) D. M. Smith, *A FORTRAN Package for Floating-Point Multiple-Precision Arithmetic*, Transactions on Mathematical Software, **17** (1991) pp.273–283.

(9) D. M. Smith, *Fortran90 Software for Floating-Point Multiple Precision Arithmetic, Gamma and Related Functions*, Transactions on Mathematical Software, **27** (2001) pp.377–387.

(10) B. Stroustrup, The C++ Programming Language 3rd ed., Addison-Wesley (1999).

(11) 一松信，初等関数の数値計算，教育出版 (1974).

(12) 平山弘，C++ 言語による高精度計算パッケージの開発，日本応用数理学会論文誌，**5** (1995) pp.307–318.

(13) 益本博幸，藤野清次，小野令美，児島彰，ある 20480 次代数方程式の係数の計算に対する多倍長演算の並列化，情報処理学会論文誌，**40** (1999) pp.4159–4168.

(14) `http://sumire.acs.i.kyoto-u.ac.jp/~fujiwara/exflib/`