64 bit 計算機環境に適した多倍長数値計算環境の構築と非適切問題の数値計算

藤原宏志† 磯 祐介†

偏微分方程式の数値解析などの大規模数値計算への適用を念頭に置き，64 bit 計算環境に適した高速な多倍長数値計算環境の設計と実装を行った．本論文では，初めに多倍長計算環境の設計を論じ，他の多倍長計算環境による結果と比較しその高速性を示す．さらにこれを用いて行った非適切問題の数値計算の数値計算結果を示し，構築した計算環境の有効性を示す．なお，本研究による多倍長計算環境は web を介して広く公開されている．

Design of a Multiple-precision Arithmetic Package for a 64-bit Computing Environment and Its Application to Numerical Computation of Ill-posed Problems

Hiroshi Fujiwara† and Yuusuke Iso†

We design and implement a fast multiple-precision arithmetic package suited for the 64-bit computing environment, which we intend to apply to large scale computings such as numerical computation of partial differential equations. In this paper, we describe the design of the package and verify its high speed in comparison with a result reported in this journal.We show some numerical results as its important application to numerically unstable problems. The package is widely open through a web site.

1. はじめに

通常のワークステーションや汎用計算機で FORTRAN や C 言語を用いて科学技術計算を行う場合，計算機内において実数は有限桁浮動小数点数として表現される．これについて現在では IEEE754[1] に定められる倍精度方式が広く利用されており，この形式は 10 進法で約 15 桁の精度を持つ．

この浮動小数点方式は，2 進法展開有限桁の有理数で実数を近似することにほかならない．そのため一般には実数 x とその計算機内部で表現した浮動小数点数 $\bar{x}$ との間には，丸め誤差（rounding error）と呼ばれる誤差 $x-\bar{x}$ が生じる．またこの表現に起因して，丸め誤差を含む数どうしの演算を行う際には，桁落ちや情報落ちと呼ばれる計算機現象が発生する．したがって，現在の計算機上で有限桁浮動小数点方式を用いて信頼できる数値計算を行う際には，つねにこの丸め誤差の混入に留意する必要がある[2]．

偏微分方程式や積分方程式などの数値計算を行う場合，特にその離散化問題が数値的に不安定になる問題（ill-conditioned problems）では，この丸め誤差の累積が著しく増大して計算結果が意味をなさない場合がある．一方で近年の計算力学の研究においては，逆問題で代表される（Hadamard の意味で）非適切な問題（ill-posed problems）の数値計算の必要性が高まっており，数値的に不安定な問題の取扱い法の確立が望まれている．このような背景にあって著者らは，任意に桁数を設定し，その桁数による実数の表現と演算を行える多倍長数値計算環境の有効性に注目した．すなわち，問題に応じて，丸め誤差があらかじめ設定された桁数を超えて増大しないよう十分な桁数を確保して数値計算を実行することで，仮想的に丸め誤差のない数値計算を実現し，通常の倍精度環境では数値計算が困難とされていた数値的に不安定な問題の数値計算が可能になると考えた．

これまでにもいくつかの多倍長数値計算環境が実装され利用されてきた[3),4)]が，偏微分方程式の高速・高精度数値シミューレションを目的とする計算力学への今後の適用を想定した場合，これまでの計算環境では

† 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University

不十分と考えて本研究を行った．すなわち，これらの環境は 32 bit 計算機での動作が前提とされており，今後普及することが予想される 64 bit 計算機の性能を十分に活かすことが不可能と思われる．また数論の研究において我が国で広く利用されている UBASIC[5] はその高速性では定評があるが，扱いうるメモリ量に制限があるため，計算力学における偏微分方程式の数値計算などの大規模数値計算には適用できないという欠点があった．これらの状況を考慮し，著者らは今後の普及が予想される 64 bit 計算環境を有効に利用し，さらに微分方程式の大規模数値計算への適用を想定した多倍長数値計算環境を設計し実装した．さらにいくつかの数値的に不安定な問題に対して実装された計算環境を適用した数値実験を行い，その有効性と妥当性を検証し，多倍長数値計算が数値的に不安定な問題の数値計算において有効な手段となることを示唆するに至っている．

なお，本研究における多倍長環境の設定と実装は，主として第 1 著者によるものである[6]．また本研究による多倍長環境は web[7] を介して広く公開されている．

2. 多倍長数のデータ構造

本設計の多倍長数値計算環境は，その設計方針を

- 64 bit 計算機上で高速に動作すること
- 偏微分方程式の数値計算などの大規模数値計算に適用可能であること
- エンドユーザから利用しやすいこと

としたことが特徴である．

計算機で扱う実数をまず $(-1)^s \times 2^e \times 1.F$ の 2 進法の形に正規化し，符合部 s を 1 bit，指数部 e に 63 bit，仮数部 F に $64 \times n$ bit を割り当て，計算機の内部に保持する．ここで n はコンパイル時に決定する精度のパラメータであり，プログラム上では制限を設けておらず，仮数部に $64 \times n$ bit を持つ多倍長数の精度は暗黙のうちに省略された 1 bit（hidden 1）を考慮すると 10 進法で約 $\log_{10} 2^{64n+1}$ 桁の精度を持つ．ここで，仮数部 F の表現に 64 bit 符号なし整数を利用し，それらの演算に ALU（Arithmetic Logical Unit）を利用したことが本設計の最大の特徴である．

計算機の内部のアーキテクチャレベルでは，数値処理に際して整数型と浮動小数点型の 2 つのデータ型を扱う．32 bit 計算環境では，上記の浮動小数点方式で多倍長数を表現する場合，その仮数部には IEEE754 に定められる倍精度浮動小数点数を利用する方が精度・演算速度の点において有利である．しかしこの方式では，掛け算のあふれを考慮すると，各要素を記憶する倍精度数のうちたかだか 26 bit，すなわち 26/64（約 40%）しか有効な情報を保持することができず，半分以上のメモリが無駄になるという欠点を持つ．しかし，今後の普及が予想される 64 bit 計算機では，演算の精度と速度を考慮した場合，特に加減算において，仮数部を保持するのに 64 bit 長の符号なし整数の配列を利用して ALU での演算を利用する方が有利であると考えた．この方法では，アセンブラ命令を補助的に利用することで桁あふれの処理を容易に行うことができ，符号なし整数の 64 bit すべてに有効な情報を保持させることができる．そのため，より少ないメモリ量で多倍長数を表現することが可能となり，大規模数値計算に適した環境といえる．また必要なメモリ数の減少は，メモリアクセスの回数を減少させることとなり，演算の高速化にも大きく寄与している．

現時点では本研究による多倍長計算環境は，多倍長数の四則演算，多倍長数と整数との乗除算，比較命令，代入，三角関数や指数関数などの基本的な組み込み関数を実装している．多倍長数どうしの加減算および整数との乗算には $O(n)$ の筆算によるアルゴリズムを利用した．また多倍長数の除算も筆算を利用している．乗算を高速化する研究は文献 8) に詳しく，研究過程においては筆算による $O(n^2)$ のアルゴリズムと $O(n^{\log_2 3})$ の Karatsuba のアルゴリズム[9] をそれぞれ実装して演算速度を比較した．本研究による多倍長計算環境は，1 章でも述べたとおり，偏微分方程式などで記述される非適切問題（ill-posed problems）の数値計算への適用を念頭においており，そこで必要となる桁数は 10 進法で数百桁程度と判断している．Karatsuba のアルゴリズムは桁数 n が十分大きい場合には非常に有効であるが，桁数が少ない場合には前処理および後処理が律速となりその有効性が活かされない．上述の数百桁の範囲の桁数での高速化を念頭におき本研究では筆算による方式を採用した．

多倍長数の除算には Ozawa のアルゴリズム[10] を採用した．このアルゴリズムにより基数 r，桁数 n で多倍長数の除算を行うと，$0 < n - m + 1 \ll r^{m-3}$ を満たす m に対して保護桁（guard digit）を $m-1$ 桁とって計算した場合，得られる商 Z' と真の商 Z は $|Z - Z'| < r^{-n}(1 + o(1))$ を満たす．本設計では $r = 2^{32}$ とし，n の値として 12 ∼ 52 程度での演算を想定している．これは 10 進法では 100 桁から 500 桁に相当する．この r，n の値のもとで $m = 4$ として Ozawa のアルゴリズムを実装した．そこでは商を MSB（most significant bit）から 32 bit ずつ予測する

ことになるが，本環境では次によりこの予測を行った．

アルゴリズム 1（商の予測） $r = 2^{32}, n \in \mathbb{Z}$ $(n \geq 2)$ とし．

$$A = a_0 + a_1 r^{-1} + a_2 r^{-2} + \cdots + a_n r^{-n}, \quad (1)$$

$$B = 1 + b_1 r^{-1} + b_2 r^{-2} + \cdots + b_n r^{-n} \quad (2)$$

とおく．ここで a_i, b_i $(i = 1, \ldots n)$ は整数で

$$a_0 \in \{0, 1\}, \quad 0 \leq a_i < r, \quad 0 \leq b_i < r, \quad A < B \quad (3)$$

とする．A を B で割った商の小数部分の MSB 32 bit Q は

$$Q = \left\lfloor \frac{A}{B} \times r \right\rfloor \times r^{-1} \quad (4)$$

と表される．Q の予測値 q を次で決定する．ただし実数 p に対して $(p)_2$ によって p の 2 進法表現を表すものとする．

1. $(B)_2$ を小数点以下 53 位で切り上げて b とする．
2. $\tilde{d} = 1/b$ とし，$(\tilde{d})_2$ を小数点以下 53 位で切り捨てて d とする．
3. $(A)_2$ を小数点以下 65 位で切り捨てて a とする．
4. $(d \times a)_2$ を小数点以下 33 位で切り捨てて q とする．

このアルゴリズムの第 2 段階（2.）の除算の計算には FPU を用い，そのほかには適当に正規化して ALU を用いる．これにより，シフトを除いて a，b，d，q は丸め誤差 0 で求めることが可能となる．

このアルゴリズムに対し，簡単な計算により次のことが示される．証明の詳細は文献 6) による．

補題 2 条件 (1)，(2)，(3) のもとでは，アルゴリズム 1 によって定められた q は次の評価を満たす．

$$0 \leq A - B \times q < b \times 2^{-32} + q \times 2^{-52} + a \times b \times 2^{-52} + 2^{-64},$$

$$A - B \times q - 2 \times (B \times r^{-1}) < 0.$$

この補題により次の定理が従う．

定理 3 補題と同じ条件下で Q を式 (4) によって定めるとき，アルゴリズム 1 で定まる q は $q \leq Q$ を満足し，その誤差は 32 bit でたかだか 1 bit である．

この定理により，アルゴリズム 1 で予測された商が真の商 (4) と異なる場合でも，その差は 1 bit だけ大きいものであるということが分かり，その補正とアルゴリズムの続行は容易である．

本研究では，多倍長数の四則演算を実現するこれらの関数は，64 bit RISC アーキテクチャの 1 つである Alpha アーキテクチャ[11] を対象としてアセンブラで記述した．Alpha アーキテクチャのアセンブラ言語を用いることで，64 bit 符号なし整数の積を 128 bit で求める演算を利用してハードウェア資源を有効に利用するとともに，ここにソフトウエア・パイプラインを適用して高速化を図っている．

本設計の多倍長計算環境は，C 言語から利用することが可能なライブラリとインクルードファイルから構成される．π などのいくつかの数学定数はあらかじめライブラリ中に $64 \times 1023 \times \log_{10} 2$ 桁（約 19,700 桁）で用意した．

3. インタフェース

本設計の多倍長計算環境は，多倍長数を利用したプログラミングを行う際のプログラマの負担を軽減するため，C++ 言語のオーバロードを利用したポリモルフィックなインタフェースを提供している[12]．

例として，a，b，h を多倍長数とし $S = (a+b)*h/2$ の計算を想定する．本設計のように演算を外部関数で実装して各演算子をオーバロードすることにより，ユーザは S を計算するためにはプログラム中に

```
S = (a + b) * h / 2;
```

と記述すれば十分である．オーバロードにより実現される静的ポリモルフィズムでは，コンパイル時に自動的に演算を実現する適切な外部関数が結び付けられる．また $S = (a + b) * h/1.5$ への変更に際してもユーザプログラムを

```
S = (a + b) * h / 1.5;
```

と修正してコンパイルを行うだけで十分である．一方，C 言語で同様の計算を実現するには，プログラマ自身が計算式 $S = (a + b) * h/1.5$ の構文解析を行い，一時変数 `tmp1`，`tmp2` を用意して

```
multiple_add(a, b, tmp1);
multiple_mul(tmp1, h, tmp2);
multiple_i_div(tmp2, 2, S);
```

と記述する必要がある．さらに S の値を $S = (a + b) * h/1.5$ とする場合，最後の文を

```
multiple_i_div(tmp2, 1.5, S);
```

と記述するだけでは第 2 引数の型が適さない可能性があり正常な動作が保証されない．

また，本設計の計算環境では，演算子 “`=`” をオーバロードすることで，C 言語の NULL 文字終端文字列により多倍長数に値を代入する機能を有する．この文字列中には 10 進法で記述された定数，あらかじめライブラリ中に保持されている定数およびそれらの四則を記述することが許される．この文字列中の式は，再帰下降法によりプログラム実行時に構文解析が行われ，多倍長数への代入が行われる．

これらの関数や前節で論じた演算を実現する関数は，アセンブラまたは C 言語により記述・実装されている

表 1 計算に要した時間（単位：時間）
Table 1 Execution time in hours.

N	演算桁数	A	B	C	D	比 $\left(\frac{\text{環境 } C}{\text{環境 } D}\right)$
4096	905	0.903	0.439	0.013	0.0063	2.06
8192	1805	13.33	6.671	0.145	0.049	2.96
16384	3610	213.4	—	1.265	0.39	3.24
20480	4520	561.1	—	2.015	0.76	2.65

が，それらを C++言語でオーバロードされた演算子関数（operator functions）から呼び出すことにより，ポリモルフィックなインタフェースが実現されている．さらに本設計では多倍長数の出力に際して，マニピュレータ（manipulator）を用いて，`setiosflags` により出力形式を，また `setprecision` により出力精度の指定を行うことが可能である．

以上のようなポリモルフィズムの利用により，ユーザ・プログラムの可読性の向上，すでに C 言語で書かれてきたプログラムの移植性の向上，型のミスマッチに対するセキュリティの向上，プログラムの変更に対する強度の向上などが期待される．

4. 演算速度の比較

演算速度の比較は，四則演算の処理速度で論じられることが多いが，本研究のように科学技術数値計算への適用を目的とする場合は具体的なベンチマークを用いて比較する方が数値計算の現場に携わるユーザにとって有益と考えられる．すなわち，ハードウェア環境も含めて，具体的な問題の全体的な処理時間を測定・比較する方が，ユーザに対して有益な情報を与えると考えた．

本設計の計算環境の高速性を示すため，本誌に 1999 年に発表された益本らの問題[13] をベンチマーク問題として採用し，比較・検討を行った．この問題は，次のチェビシェフ多項式に起因する漸化式

$$
\begin{aligned}
&a_0 = 1,\\
&a_{2k} = -\frac{N}{2k}\sum_{j=1}^{k}\frac{1}{2j+1}a_{2(k-j)},\\
&a_{2k+1} = 0.\\
&\left(k = 1, 2, \ldots, \frac{N}{2}\right)
\end{aligned}
$$

の数値計算である．次の 4 つの環境で処理時間の測定・比較を行った．なお，計算環境 A，B，C とその結果は文献 13) からの引用である．

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

計算環境 B POWER2（66 MHz，SPECint95 3.41，SPECfp95 10.2）と MPPACK

計算環境 C POWER2（66 MHz）の 48 台並列システムと MPPACK

これらと比較をする本設計の多倍長演算環境は次のとおりである．

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

各々の計算環境で計算に要した時間を**表 1** に示す．益本らは並列計算により高速化を図っているが，本設計の多倍長計算環境ではさらに高速化が実現されていることが分かる．

5. 数値的に不安定な問題への適用

本設計の多倍長数値計算環境の応用面での有効性を示すために，次の Cauchy-Riemann 方程式の初期値問題を考える：

$$
\frac{\partial}{\partial t}\begin{pmatrix}u\\v\end{pmatrix} = \begin{pmatrix}0 & 1\\-1 & 0\end{pmatrix}\frac{\partial}{\partial x}\begin{pmatrix}u\\v\end{pmatrix} \quad t > 0, x \in \mathbb{R}, \tag{5a}
$$

$$
\begin{pmatrix}u\\v\end{pmatrix}(0, x) = \begin{pmatrix}-x^2\\-0.5\end{pmatrix} \quad x \in \mathbb{R}. \tag{5b}
$$

この初期値問題は，亀裂の位置・形状などの測定を目指した電気ポテンシャル CT 法（electric potential computed tomography method）[14] に現れる Laplace 方程式の初期値問題と同値である．この電気ポテンシャル CT 法は，非破壊検査の 1 つの手法として知られており，応用上の観点から重要な問題であるが，厳密解を構成する有効な手段がなく，数値的に近似解を構成する手法をとらざるをえない．しかし，この初期値問題は Hadamard の意味で非適切となることが知られており[15]，その差分解法は不安定であることが知られている．

この差分法の数値計算に対しては，Quasi-Reversibility 法[16] など不安定性を緩和する正則化法の研究も行われているが，正則化パラメータの適切な決定という別の問題が生じる．一方でこの差分法に対しては後述のとおり，あるクラスでの収束性が証明されており，計算誤差のない環境では差分法の収束性が示されている．

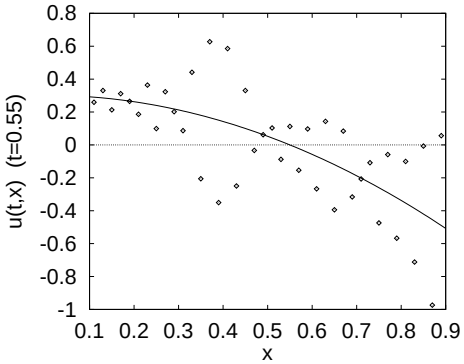

図 1 倍精度での計算結果（$u(t,x)$）（◇：数値解，点線：厳密解）
Fig. 1 Result by the double precision ($u(t,x)$). (◇: numerical solution, dotted line: exact solution).

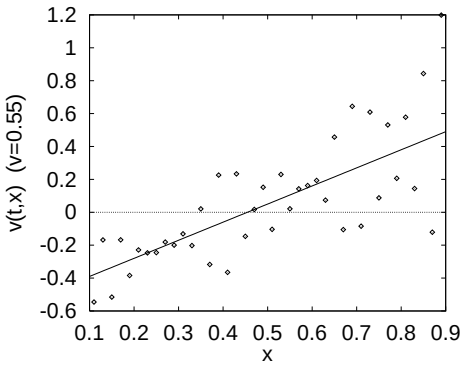

図 2 倍精度での計算結果（$v(t,x)$）（◇：数値解，点線：厳密解）
Fig. 2 Result by the double precision ($v(t,x)$). (◇: numerical solution, dotted line: exact solution).

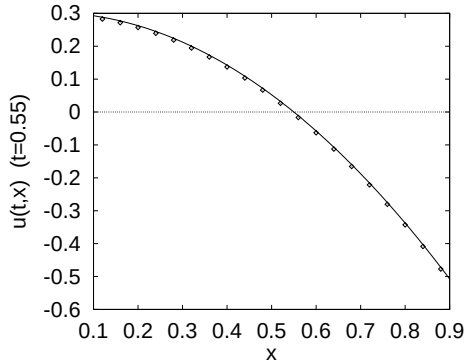

図 3 100 桁での計算結果（$u(t,x)$）（◇: 数値解，点線：厳密解）
Fig. 3 Result by 100 digits in radix-10 ($u(t,x)$). (◇: numerical solution, dotted line: exact solution).

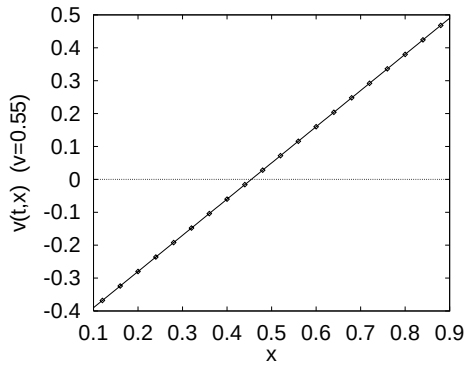

図 4 100 桁での計算結果（$v(t,x)$）（◇：数値解，点線：厳密解）
Fig. 4 Result by 100 digits in radix-10 ($v(t,x)$). (◇: numerical solution, dotted line: exact solution).

ここでは，この初期値問題に対する差分法による数値計算について，丸め誤差の影響を隠蔽できる本研究の多倍長計算環境と，通常の倍精度計算環境で計算を行い，具体的な数値的不安定問題への多倍長環境の適用の有効性を示す．

この Cauchy-Riemann 方程式に対する厳密解は次のとおりである．

$$\begin{pmatrix} u \\ v \end{pmatrix}(t,x) = \begin{pmatrix} t^2 - x^2 \\ 2tx - 0.5 \end{pmatrix}.$$

方程式 (5) に時間・空間方向の双方に前進差分法を適用することで，次の差分スキームを得る．

$$\frac{U^{k+1}(x) - U^k(x)}{\Delta t} = \frac{V^k(x+\Delta x) - V^k(x)}{\Delta x}, \tag{6a}$$

$$\frac{V^{k+1}(x) - V^k(x)}{\Delta t} = -\frac{U^k(x+\Delta x) - U^k(x)}{\Delta x}, \tag{6b}$$

$$U^0(x) = -x^2, \tag{6c}$$

$$V^0(x) = -0.5. \tag{6d}$$

すでに述べたとおり，方程式 (5) の非適切性（ill-posedness）はその差分スキーム (6) の不安定性を引き起こし，有限桁の浮動小数点方式を利用した数値計算では混入する丸め誤差の増大により，厳密解とはかけ離れた数値解しか得ることができない．実際，通常の倍精度で得られる数値解を**図 1**，**図 2** に示す．いずれも $\Delta t = \Delta x = 0.01$ とし，$t = 0.55$ における数値解を求めた．厳密解を破線で，計算された数値解を点で示している．横軸は x を，縦軸が $u(t,x)$ もしくは $v(t,x)$ を表す．図 1，図 2 により，◇ で示された数値計算の値は，厳密解と著しく異なっていることが分かる．

しかし一方で Hayakawa の研究[17] により，解析函数の枠内ではスキーム (6) の厳密解は (5) の厳密解に，Δt，$\Delta x \to 0$ に従ってコンパクト一様収束することが証明されている．したがって，本研究で構築する多倍長計算環境を利用し，丸め誤差の増大が要求される

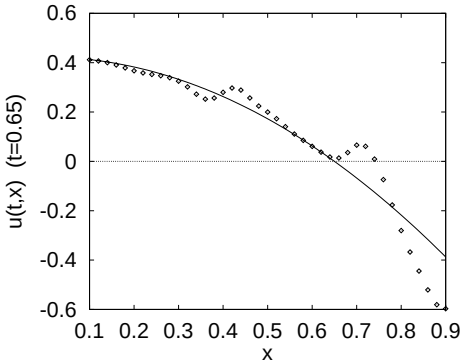

図 5 100 桁での計算結果（$u(t,x)$）（◇：数値解，点線：厳密解）
Fig. 5 Result by 100 digits in radix-10 ($u(t,x)$). (◇: numerical solution, dotted line: exact solution).

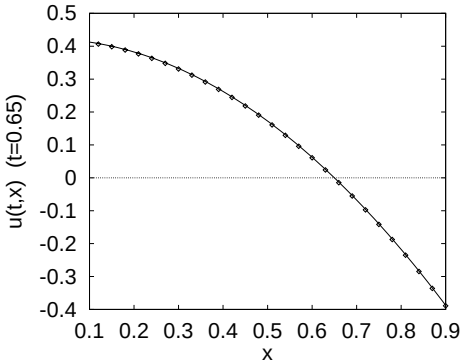

図 7 120 桁での計算結果（$u(t,x)$）（◇：数値解，点線：厳密解）
Fig. 7 Result by 120 digits in radix-10 ($u(t,x)$). (◇: numerical solution, dotted line: exact solution).

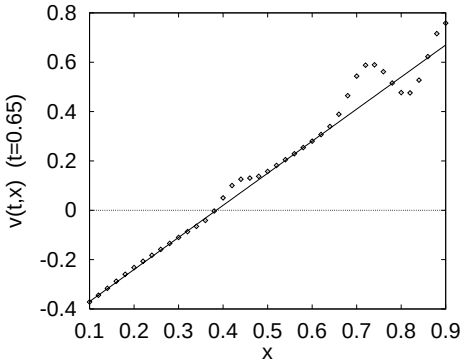

図 6 100 桁での計算結果（$v(t,x)$）（◇：数値解，点線：厳密解）
Fig. 6 Result by 100 digits in radix-10 ($v(t,x)$). (◇: numerical solution, dotted line: exact solution).

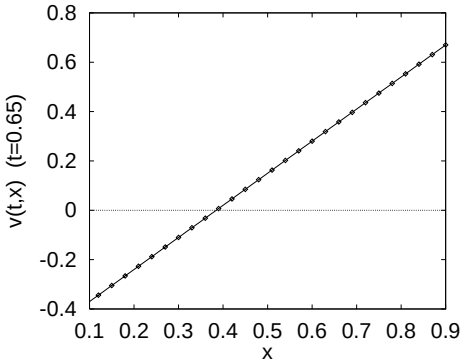

図 8 120 桁での計算結果（$v(t,x)$）（◇：数値解，点線：厳密解）
Fig. 8 Result by 120 digits in radix-10 ($v(t,x)$). (◇: numerical solution, dotted line: exact solution).

精度に達しないように十分な桁数を確保して数値計算を行うことにより仮想的に丸め誤差のない数値計算が可能となり，Hayakawa の結果が計算機上で実現されたと判断している．その計算結果は**図 3**，**図 4** のとおりである．数値計算は 10 進法で 100 桁の精度を確保して行い，差分化パラメータを $\Delta t = \Delta x = 0.01$ とし，$t = 0.55$ における数値解を図示した．

この例でも分かるとおり，有限桁の多倍長数値計算は，数値的に不安定な問題の数値計算に有効であるが，一方で問題の不安定性を本質的に解決しているわけではないことに注意を払う必要がある．たとえば同じ多倍長計算環境でも，時間・空間方向の分割幅のパラメータを $\Delta t = \Delta x = 0.0025$ に変更すると，$t = 0.65$（260 ステップ）での数値解が**図 5**，**図 6** となる．計算ステップの増加により累積丸め誤差が増大し，計算精度が損われていることが分かる．すなわち，これらの分割数に対しては 100 桁の精度は不十分であることが分かる．一方，同じ離散化パラメータに対して 120 桁を確保して数値計算を行うと，再び**図 7**，**図 8** のように厳密解を十分近似する数値解を得ることができる．

以上の結果のとおり，これまでは直接的な数値計算不可能と考えられてきた数値的に不安定な問題の数値的取扱いに対して，多倍長数値計算環境が 1 つの有効な手段となりうることが分かる．これは，たとえば，非破壊検査における電気ポテンシャル CT 法の解像度が向上することを意味しており，工学の観点からも有用な応用が期待されると考えている．しかし必要な計算桁数については事前の数学的評価や工学的先験情報の活用が必要であり，この点については具体的な問題に沿った事例研究が必要と思われる．

また典型的な非適切問題である第 1 種積分方程式の数値計算に対する本システムの適用例は，文献 18) で示されている．

6. 結　　論

64 bit 計算環境向の多倍長数値計算環境を構築し，

いくつかの数値実験に適用してその高速性と有効性を確認した．その結果，従来 32 bit 計算環境向に構築されたパッケージを用いて並列プログラムを記述するよりも，より高速に数値計算を行うことができた．また，本設計の多倍長数値計算環境は大規模な数値計算を高速に処理することが可能であるという特色を生かし偏微分方程式の数値計算を実行した結果，これまで数値計算が困難であると考えられてきた数値的に不安定と呼ばれる問題に対して，多倍長計算環境が有効な計算手段となることが示唆された．

多倍長計算の単純適用が，数値的に不安定と呼ばれる問題すべてに対する汎用的な解決手段とは考えにくい．しかし，本研究で紹介した非適切問題に対しては，正則化法などとの組合せにより，将来の数値解析手法の発展に寄与するものであると考えている．

本研究で提案する多倍長計算環境はインターネットを通じてのダウンロードし，利用することが可能である[7]．

謝辞 本研究の遂行にあたって，京都大学名誉教授の一松信先生に貴重なご助言をいただきました．また（株）三和システム開発および日本学術振興会科学研究費（課題番号：13440031）のご援助をいただきました．この場を借りて御礼申し上げます．

参 考 文 献

1) IEEE: IEEE Standard 754-1985 for Binary Floating-Point Arithmetic, *SIGPLAN*, Vol.22, No.2, pp.9–25 (1987).
2) Goldberg, D.: What Every Computer Scientist Should Know About Floating-Point Arithmetic, *ACM Computing Surveys*, Vol.23, pp.5–48 (1991).
3) Smith, D.M.: A FORTRAN Package for Floaring-Point Multiple-Precision Arithmetic, *Trans. Mathematical Software*, Vol.17, pp.273–283 (1991).
4) 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，Vol.5, pp.307–318 (1995).
5) 木田祐司：UMASIC86 多倍長計算用 BASIC NEC PC-9801 とその互換機用 8.7 版ユーザーズマニュアル，日本評論社 (1994).
6) 藤原宏志：多倍長計算環境の構築と非適切問題の数値計算，修士学位論文，京都大学大学院情報学研究科 (2000).
7) http://sumire.acs.i.kyoto-u.ac.jp/~fujiwara/exflib.html
8) Knuth, D.E.: *The Art of Computer Programming, Semi Numerical Algorithm*, 3rd ed., Addison-Wesley (1998).
9) Karatsuba, A. and Ofman, Y.: Multiplication of multidigit numbers on automata, *Soviet Physics Doklady*, Vol.7, pp.595–596 (1963). (English translation).
10) Ozawa, K.: A Fast $O(n^2)$ Division Algorithm for Multiple-Precision Floating-Point Numbers, *J. of Information Processing*, Vol.14, pp.354–356 (1991).
11) Compaq: *Alpha Architecutre Handbook*, Compaq Computer Corporation (1998).
12) Stroustrup, B.: *The C++ Programming Language*, 3rd ed., Addison-Wesley (1999).
13) 益本博幸，藤野清次，小野令美，児島 彰：ある 2048 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，Vol.40, pp.4159–4168 (1999).
14) 久保司郎：逆問題，培風館 (1992).
15) イ・ゲ・ペトロフスキー：偏微分方程式論，東京図書 (1958).
16) Lattès, R. and Lions, J.-L.: *The Method of Quasi-Reversibility: Application to Partial Differential Equations*, American Elsevier (1969).
17) Hayakawa, K.: Convergence of Finite Difference Scheme and Analytic Data, *Publ. Res. Inst. Math. Sci.*, Vol.24, pp.759–764 (1998).
18) Fujiwara, H. and Iso, Y.: Numerical Challenge to Ill-posed Problems by Fast Multiple-Precision System, *Theoretical and Applied Mathematics 50*, pp.419–424 (2001).

(平成 14 年 4 月 6 日受付)
(平成 15 年 1 月 7 日採録)

藤原 宏志

昭和 51 年生．平成 11 年京都大学理学部卒業．現在，同大学大学院情報学研究科複雑系科学専攻博士後期課程在籍（平成 15 年 3 月修了予定）．非適切問題の数値解析，多倍長数値計算環境の設計等の研究に従事．日本数学会会員．

磯 祐介（正会員）

昭和 33 年生．昭和 63 年京都大学大学院理学研究科数理解析専攻修了．理学博士．現在，京都大学大学院情報学研究科教授．偏微分方程式および非適切問題の数値解析の研究に従事．日本数学会，日本応用数理学会等に所属．