Vol.2014-ARC-209 No.1
2014/3/6

情報処理学会研究報告
IPSJ SIG Technical Report

浮動小数点演算における桁落ちを対象とした有効桁数追跡アルゴリズムの検討と評価

安仁屋 宗石[1]　北村 俊明[1]

概要：浮動小数点演算では，桁落ち，情報落ち，丸め処理の影響で演算結果の精度が低下する場合がある．またユーザーは，桁落ち，情報落ちが，浮動小数点演算を含むアプリケーションの最終結果にどのような影響を及ぼすか知る事ができない．桁落ちに着目し，ユーザーに演算結果の有効桁数を提供する，従来手法では妥当な有効桁数を見積もれない場合があった．そこで本研究は，従来手法を元に，桁落ちビット数を演算前の桁落ち回数や有効桁数，桁落ち率を元に変化させ，その値を用いて有効桁数を計算することで，より妥当な有効桁数を追跡するアルゴリズムの検討と評価を行った．その結果，提案手法では従来手法と比較して，実際の精度により近い有効桁数を見積もれることを確認した．

1. はじめに

近年，計算機の著しい性能向上に伴い，幅広いの分野で計算機による数値計算が行われている．しかし，数値計算で使用する浮動小数点数は，数値の表現可能な範囲が限られているため，浮動小数点演算を行う際に，桁落ち，情報落ちによる精度低下を引き起こし，誤差が発生する要因となる．しかし現在，数値計算の浮動小数点演算規格として広く使用されている IEEE754[1] は，桁落ち，情報落ちに対する例外の定義が無いため，ユーザーが精度低下に気付かずに演算を繰り返し行と，実際とは全く異なった結果となる可能性があり，演算結果の信頼性保証が不十分である．浮動小数点演算の精度低下に関する研究の多くは，桁落ちや情報落ちの検出のみを行い，演算結果の信頼性が示されていない．また，従来の有効桁数を追跡する研究は，有効桁数を実際の精度よりかなり厳しく見積もってしまう場合があった．

そこで本研究は，従来手法において，単純に桁落ちしたビット数を追跡して有効桁数を計算したことが，有効桁数の見積りが厳しくなった原因と考え，桁落ちビット数の追跡方法に対して，3 つのアルゴリズムを適応し実験と評価を行った．それぞれ，演算引数の桁落ち回数を参照し，回数が増加すると桁落ちビット数を緩めるアルゴリズム，次に，演算引数の有効桁数を参照し，有効桁数が減少すると，桁落ちビット数を緩めるアルゴリズム，最後に，桁落ち率に着目して，桁落ちビット数を緩めるアルゴリズムである．その結果，従来手法では妥当な有効桁数を追跡できないケースにおいて，提案手法が実際の精度に近い有効桁数を見積もれることを確認した．さらに，提案手法のハードウェア化を見据え，PC エミュレータの一種である QEMU を用いて，仮想的に提案手法を用いて演算を行う FPU を実装し，実験と評価を行った．

[1] 広島市立大学
Hiroshima City University, 3-4-1 Ootsuka-higashi, Asami-nami, Hiroshima 739–2115, Japan

2. 研究背景

計算機を用いて数値計算を行う場合，実数を有限な浮動小数点数を用いて表現する．そのため，浮動小数点演算を行う際に精度低下が発生し，演算の結果が実際の値とは異なってしまう可能性がある．本節は，浮動小数点演算規格である IEEE754 について説明し，精度低下の要因となる桁落ち，情報落ちについて述べる．

2.1 浮動小数点演算規格

計算機では大抵の場合，数値を整数か浮動小数点数を用いて表現する．実数を扱う際には，一般的に浮動小数点数が利用され，仮数，基数，指数の 3 つで表現される．浮動小数点数を用いた演算は浮動小数点演算と呼ばれる．現在，最も広く用いられている浮動小数点演算標準である IEE754 は，基数を 2 とし，単精度，倍精度，4 倍精度等の形式と演算が定義されている．その中で，計算機を用いた数値計算において，最もよく用いられる倍精度の浮動小数点数で表現される値は式（1）で表すことができる．

$$(-1)^{exponent} \times 2^{exponent-1023} \times (1 + fraction) \quad (1)$$

© 2014 Information Processing Society of Japan

指数部ではバイアス表現を用いるため，$\times 2^1$ を表現したい場合の指数部の値は 1024 となる．仮数部の整数部分は常に 1 と定められており，仮数部のビット幅は 52bit だが，常に 1 となる暗黙の整数ビットを考慮すれば，その精度は 53bit である．これにより倍精度浮動小数点数は，実数を 53bit の有理数で近似して扱うことになる．この精度は 10 進数に換算すると，約 16 桁に相当する．また，単精度は，符号 1bit，指数部 8bit，仮数部 23bit から成る 32bit で表現され，4 倍精度は，符号 1bit，指数部 15bit，仮数部 112bit から成る 128bit で表現される．

2.2 桁落ち

桁落ちは，浮動小数点加減算において，演算結果の有効桁数が減少するこである．同符号で絶対値が非常に近い値同士の減算または，異符号で絶対値がほぼ等しい加算を行った場合，正規化によって有効桁数が減少することで発生する．桁落ちが発生する 10 進数の演算例を式（2）に示す．

$$1.003\times10^{10}-1.000\times10^{10} = 0.003\times10^{10} = 3.000\times10^{7} \quad (2)$$

演算前のオペランドは 4 桁の有効桁数を持つが，演算後は 1 桁しか有効桁数が無いことがわかる．また，左辺オペランドの「3」の桁が誤差を含む場合は，この桁落ちは精度低下を引き起こしている．しかし，3 の桁が正しく，それ以降の桁が全て 0 場合，精度低下は発生しない．

桁落ちによる精度低下は，誤差の影響が比較的少ない仮数部の下位ビットに位置していた不正確ビットが，桁落ちによって仮数部の上位方向にシフトされ，不正確ビットによる誤差の影響が大きくなることによるものである．

2.3 情報落ち

絶対値の大きさが極端に異なる値同士の加減算を行った場合，絶対値の小さい値が演算結果に反映されないことを情報落ちと呼ぶ．以下に 10 進数での有効桁数 5 桁式（3）と有効桁数 4 桁式（4）での例を示す．

$$2.0000 \times 10^4 + 1.0000 = 2.0001 \times 10^4 \quad (3)$$

$$2.000 \times 10^4 + 1.000 = 2.000 \times 10^4 \quad (4)$$

上記の通り，式（3）では正しい結果となるが，式（4）では，計算結果に絶対値の小さい値が反映されていないことが分かる．

浮動小数点加減算を行う場合，指数部の絶対値が大きい方に揃えて演算を行う．この時，絶対値の差が大きいと，小さい方の値は大きいく右シフトされ，仮数部の表現範囲から溢れてしまい，情報が欠落してしまう．これを回避する方法として，絶対値の大きく異なる加減算を行わないことが挙げられる．例として，値の総和を計算する際に，あらかじめソートを行い，絶対値が小さい値から計算する手法がある．

Algorithm 1 桁落ち検出と桁落ちビット数の計算

```
if exp_op1 > exp_op2 then
  big_exp ⇐ exp_op1
else if exp_op1 < exp_op2 then
  big_exp ⇐ exp_op2
else
  big_exp ⇐ exp_op1
end if
if big_exp > exp_result then
  桁落ち検出
  桁落ちビット数 = big_exp − exp_resul
  有効桁数 = 現在の有効桁数 − 桁落ちビット数
end if
```

3. 先行研究

本節は，本研究の元となった，鈴木らによる浮動小数点演算における精度を見積もる研究 [2] を紹介する．浮動小数点演算における精度低下は，演算ごとに桁落ち，情報落ち，丸め処理による有効桁数の変化を追跡することができれば，演算結果の精度を見積もることが可能である．しかし，桁落ち，情報落ち，丸め処理の 3 つを正確に追跡するには非常に高いコストを要する．そこで，この研究では，仮数部の上位の桁に影響を及ぼす桁落ちにのみを対象としている．従ってここでの有効桁数は桁落ちの影響を受けていない桁数を意味する．

この研究における有効桁数の計算方法について紹介する．まず，桁落ちの検出と桁落ちビット数の計算について説明する，この処理は全ての浮動小数点加減算にごとに行われる．桁落ちの検出は，演算引数の大きい方の指数部が演算結果の指数部より大きい場合に行う．また，桁落ちが検出された場合，演算引数の大きい方の指数部から演算結果の指数部を減算し，桁落ちビット数を計算する．桁落ち検出と桁落ちビット数を計算する疑似コードを Algorithm 1 に示す．exp_op1，exp_op2 はそれぞれ，演算引数の指数部を，exp_result は演算結果の指数部を示す．有効桁数は，現在の有効桁数から桁落ちビット数を減算して求められる．

次に有効桁数の計算について説明する．桁落ちが発生した場合，有効桁数の計算は，演算引数の指数部が同じ場合と，演算引数が異なる場合の 2 つのパターンに分かれる．演算引数の指数部が同じ場合，演算結果の有効桁数は，演算引数の少ない方の有効桁数から桁落ちビット数を減算した結果が新たな有効桁数となる．演算引数の指数部が異なる場合は，桁合わせが行われるため，指数部が小さい方の有効桁数に，演算引数の指数部の差を加算する．これは，桁合わせに伴い，指数部が小さい方の仮数部が，指数部の差の分だけ右シフトを受け，既に桁落ちの影響を受けた仮数部の下位ビット桁が無視さるため，見かけ上の有効桁数が回復することによるものである．その後，少ない方の有

© 2014 Information Processing Society of Japan

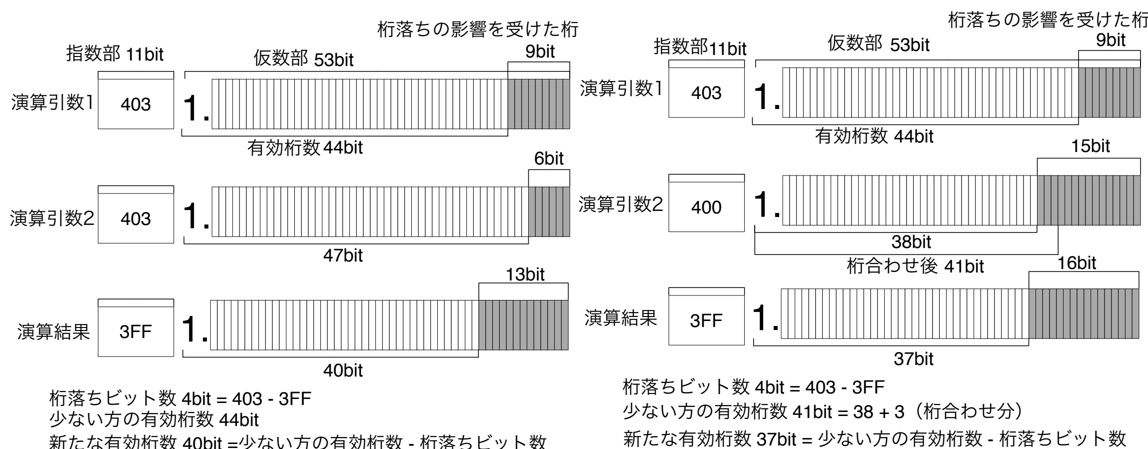

図 1　有効桁数の計算方法

効桁数を選択し，桁落ちビット数を減算した結果を演算結果の有効桁数とする．それぞれの場合における有効桁数を計算する例を図 1 に示す．左側が指数部が同じ場合，右側が指数部が異なる場合である．左側の例では，指数部が同じであるため，桁合わせの処理は行われない．桁落ちビット数は演算引数の指数部 403 から演算結果の指数部 3FF を減算した 4bit である．次に，少ない方の有効桁数を選択する，この例では演算引数 1 の 44bit を選択する．最後に選択した有効桁数から桁落ちビット数を減算した 40bit が演算結果の有効桁数となる．右側の例は，指数部が異なるため，桁合わせが行われる．その結果，指数部の小さい演算引数 2 の有効桁数は，38bit に桁合わせ分の 3bit が加算され，41bit となる．次に少ない方の有効桁数を選択する，この例では演算引数 2 の 41bit が選択される．最後に少ない方の有効桁数から桁落ちビット数の 4bit を減算し，演算結果の有効桁数は 37bit となる．また，桁落ちが発生しない場合でも，桁合わせを行う演算があれば，有効桁数を新たに計算し，見かけ上の有効桁数の回復にも対応している．最後に，浮動小数点乗除算の場合は，桁落ちが発生しないため，演算結果の有効桁数は，演算引数のうち，少ない方の有効桁数が適応される．上記のアルゴリズムを倍精度を対象に C++のオペレーターオーバーロードを用いて実現している．

この様に，桁落ちや，演算前の有効桁数を追跡することにより演算結果の有効桁数を見積もることができる．しかし，この従来手法では，有効桁数を実際の精度より，かなり厳しく見積もってしまう場合があった．これは，桁落ちによって仮数部の下位の桁に追加された 0 が正しい場合もあるためだと考えられる．そこで，提案手法では，桁落ちビット数を変化させて，有効桁数を計算することで，より実際の精度に近い有効桁数を見積もれる様にした．

Algorithm 2 演算引数の桁落ち回数を元にしたアルゴリズム

if $big_exp > exp_result$ **then**
　桁落ち検出
　if $big_NC > TH$ **then**
　　桁落ちビット数 $= (big_exp - exp_result)/2$
　else
　　桁落ちビット数 $= big_exp - exp_result$
　end if
　有効桁数 = 現在の有効桁数 − 桁落ちビット数
end if

4. 提案手法

従来手法は，ネイピア数を底とする指数関数のテイラー展開の様に，桁落ち回数が多くなるに従い，実際の精度が 0 になってしまう計算や，桁落ちの回数が少ない計算では有効であったが，ヒルベルト行列を係数とする連立方程式の計算の様な，桁落ち回数と実際の精度が比例関係に無い場合は，実際の精度よりかなり厳しく有効桁数を見積もってしまう問題があった．これは，桁落ちによって仮数部の下位の桁に挿入される 0 が，正しい場合もあるが，桁落ちしたビットは全て有効桁数から除外されてしまうことが原因だと考えらる．そこで提案手法は，桁落ちビット数を単純に追跡するのではなく，三つのパターンに変化させ，有効桁数の計算を行った．これらのアルゴリズムは倍精度を対象に C++のオペレーターオーバーロードを用いて実装した．

4.1 桁落ち回数を元にした有効桁数のアルゴリズム

単純に桁落ちビット数を緩めただけでは，有効桁数の見積りが甘くなってしまう．そのため，ある程度桁落ちの影響を受けた場合は，それ以降に桁落ちの影響を受けても，最終結果の有効桁数には影響が少ないと仮定した．そこで，演算前の引数の桁落ち回数を参照し，桁落ち回数が多くな

Algorithm 3 演算引数の有効桁数を元にしたアルゴリズム

```
if big_exp > exp_result then
  桁落ち検出
  if big_TRACK > TH then
    桁落ちビット数 = (big_exp − exp_result)/2
  else
    桁落ちビット数 = big_exp − exp_result
  end if
  有効桁数 = 現在の有効桁数 − 桁落ちビット数
end if
```

れば，桁落ちビット数を緩めて有効桁数を計算する．この場合の疑似コードを Algorithm 2 に示す．ここで，big_NC は演算引数の桁落ち回数が多い方の桁落ち回数，TH はユーザーが設定可能な桁落ち回数のしきい値である．

4.2 演算引数の有効桁数を元にした有効桁数のアルゴリズム

演算引数の有効桁数がある程度，桁落ちの影響を受けていた場合，それ以降に桁落ちが発生しても，最終結果への影響は少ないと仮定し，桁落ちビット数を緩めることで，実際の精度に近づくのではないかと考えた．そこで，演算引数の有効桁数を参照し，有効桁数が少ない場合は，桁落ちビット数を緩めて，有効桁数を計算する．この場合の疑似コードを Algorithm 3 に示す．実際には，有効桁数ではなく，桁落ちの影響を受けたビット数を使用して条件分岐を行っている．ここで，big_TRACK は演算引数の大きい方の桁落ちを受けたビット数，TH はユーザーが設定可能な桁落ちを受けたビット数のしきい値である．

4.3 桁落ち率に着目した有効桁数のアルゴリズム

演算引数が桁落ちの影響をどの程度受けているかを判断するため，演算引数の桁落ち率という概念を導入した．演算引数の桁落ち率はその演算引数が用いられた浮動小数点減算回数の内，何回桁落ちが発生するかという割合である．このアルゴリズムでは，桁落ち率が高く，演算引数の有効桁数が少ない場合，それ以降に発生する桁落ちが最終結果に影響する可能性は低いと仮定し，桁落ちビット数を緩めて有効桁数を計算する．このアルゴリズムの疑似コードを Algorithm 4 に示す．ここで，ncc は桁落ち回数，subc は演算引数が用いられた，浮動小数点減算回数，big_TRACK は演算引数の大きい方の桁落ちを受けたビット数，TH はユーザーが設定可能な桁落ちを受けたビット数のしきい値，RATE は桁落ち率のしきい値で 0.9 に設定した．

5. 評価

従来手法と提案手法を比較し，実際の精度にどれだけ近い有効桁数を見積もれるかを確認するため，2 つの例を用いて評価を行った．

Algorithm 4 桁落ち率に着目したアルゴリズム

```
if big_exp > exp_result then
  桁落ち検出
  if ((big_TRACK > TH)&&((ncc/subc) > RATE)) then
    桁落ちビット数 = (big_exp − exp_result)/2
  else
    桁落ちビット数 = big_exp − exp_result
  end if
  有効桁数 = 現在の有効桁数 − 桁落ちビット数
end if
```

5.1 ネイピア数を底とする指数関数のテイラー展開

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$$

x に負の値を代入して計算すると桁落ちする例である．従来手法では実際の精度に近い有効桁数を見積もれていた．有効桁数見積りの結果を表 1 に，倍精度で計算した値と，実際の値を 10 進数で表現したものを表 2 に，16 進数で表現したものを表 3 に示す．ここで，P1，P2，P3 はそれぞれ提案手法の，演算引数の桁落ち回数を元にしたアルゴリズム，演算引数の有効桁数を元にしたアルゴリズム，桁落ち率に着目したアルゴリズムである．P1 の TH は 4 回，P2，P3 は 16bit である．

表 1　各種法の有効桁数の比較

x	P1	P2	P3	従来手法	実際の精度	桁落ち
-4	53	44	44	44	44	12 回
-7	51	36	34	34	35	21 回
-9	44	37	29	29	28	25 回
-12	46	31	21	21	22	38 回
-17	43	27	7	7	6	53 回
-19	44	27	3	3	0	59 回
-21	42	25	0	1	0	63 回

表 2　10 進数での比較（e^x）

x	倍精度	実際の値
-4	$1.83156388887344E-02$	$1.83156388887341E-02$
-7	$9.11881965566013E-04$	$9.11881965554516E-04$
-9	$1.23409804015408E-04$	$1.23409804086679E-04$
-12	$6.14421272946162E-06$	$6.14421235332820E-06$
-17	$4.18833681709330E-08$	$4.13993771878516E-08$
-19	$3.25274838407556E-09$	$5.60279643753726E-09$
-21	$9.25455304561266E-09$	$7.58256042791190E-10$

この結果から，まず，提案手法 P1，P 2 は従来手法と比較し，有効桁数を緩く見積もってしまうことが確認できる．これは，P1，P2 では，計算の序盤で，ある程度桁落ちの影響を受けると，それ以降の桁落ちビット数を削減して有効桁数を計算しているためである．$x = -4$ で既に 12 回桁落ちが発生しているため，桁落ちビット数が削減され，提案手法 P2 の有効桁数見積りは緩くなっている．また，P3 では $x = -7$ までは妥当な有効桁数を見積もっているが，そ

© 2014 Information Processing Society of Japan

表 **3**　16 進数での比較 (e^x)

x	倍精度	実際の値
-4	3F92C155B8213D40	3F92C155B8213CF4
-7	3F4DE16B9C2647C9	3F4DE16B9C24A98F
-9	3F202CF224FE3512	3F202CF22526545A
-12	3ED9C54C55BB8DBD	3ED9C54C3B43BC8B
-17	3E667C68029344E3	3E6639E3175A689D
-19	3E2BF0DE8D226DB2	3E381056FF2C5772
-21	3E43DFBE8D2E7E34	3E0A0DB0D0DDB3EC

れ以降の見積りは緩くなっている．これは，$x = -9$ 以降の計算では，桁落ちにより，有効桁数が 37 桁以下になり，桁落ちビット数が削減され見積りが緩くなったと考えられる．しかし，提案手法 P3 は従来手法と比較し，より実際の精度に近い有効桁数を見積もっていることが分かる．これは，P1，P2 とは異なり，桁落ちビット数を緩める条件に，桁落ち率を含めたため，計算の序盤で有効桁数の見積りが緩くならなかったためだと考えられる．

従ってこのような，桁落ち回数が増加するにつれ，実際の精度が 0 に向かって減少するような単純な例においては，従来手法より提案手法 P3 僅かにが適している．しかし，次に示すヒルベルト行列の様な，桁落ち回数と実際の精度が比例関係に無い計算の場合，従来手法では有効桁数を非常に厳しく見積もってしまう問題がある．

5.2　ヒルベルト行列を係数とする連立方程式

従来手法では有効桁数を非常に厳しく見積る結果となる，以下のヒルベルト行列を係数とする 8 元連立方程式に，それぞれの提案手法を使用して評価を行った．解法はガウスの消去法を用いた．倍精度で計算した値と，実際の値を 10 進数で表現したものを表 4 に，16 進数で表現したものを表 5 に示す．

$$\begin{pmatrix} 1 & 1/2 & \dots & 1/8 \\ 1/2 & 1/3 & \dots & 1/9 \\ \vdots & \vdots & \ddots & \vdots \\ 1/8 & 1/9 & \dots & 1/15 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_8 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

表 **4**　10 進数での比較

x	倍精度	実際の値
x1	$6.40000006236578E+01$	$6.40000000000000E+01$
x2	$-2.01600003019065E+03$	$-2.01600000000000E+03$
x3	$2.01600003630446E+04$	$2.01600000000000E+04$
x4	$-9.24000018344100E+04$	$-9.24000000000000E+04$
x5	$2.21760004658867E+05$	$2.21760000000000E+05$
x6	$-2.88288006268354E+05$	$-2.88288000000000E+05$
x7	$1.92192004268743E+05$	$1.92192000000000E+05$
x8	$-5.14800011584717E+04$	$-5.14800000000000E+04$

表 **5**　16 進数での比較

x	倍精度	実際の値
x1	40500000029DA5C0	4050000000000000
x2	C09F800007EA0F60	C09F800000000000
x3	40D3B00005F2B838	40D3B00000000000
x4	C0F68F00078384AC	C0F68F0000000000
x5	410B1200098A968C	410B120000000000
x6	C1119880066B361A	C111988000000000
x7	4107760008BE0D04	4107760000000000
x8	C0E92300097D7DC2	C0E9230000000000

まず，演算引数の桁落ちを受けた回数を元に有効桁数を計算するアルゴリズムを使用して実験を行った．結果を表 6 に示す．ここで，5，4，3 はそれぞれ，Algorothm 2 の TH の値である．この結果から，TH が 5 の場合は有効桁数の見積りは厳しく，3 の場合の見積りは緩くなる．しかし，4 の場合に，見積もった有効桁数が実際の精度に近いことを確認した．従って，大きな桁落ちが発生する演算が，演算引数の桁落ち回数が 4 になる前に行われていると考えられる．

表 **6**　桁落ちを受けた回数を元にしたアルゴリズムの有効桁数

	5	4	3	実際の精度	従来手法	桁落ち
x1	16	20	24	27	0	13 回
x2	17	21	25	26	2	12 回
x3	19	23	27	26	6	11 回
x4	20	24	28	26	9	10 回
x5	21	25	29	26	11	9 回
x6	22	26	30	26	13	8 回
x7	22	26	33	26	14	7 回
x8	22	26	29	26	14	7 回

次に，演算引数の有効桁数を元にしたアルゴリズムを使用して計算を行った．結果を表 7 に示す．ここで，10，16，17 はそれぞれ，Algorithm 3 の TH の値である．この結果から，TH が 10 の場合は有効桁数の見積りが緩く，17 の場合は厳しく見積もっている．しかし，16 の場合は実際の精度に近い有効桁数の見積りが行えることを確認した．従って，大きな桁落ちが発生する演算は，演算引数の有効桁数が 37 桁になる前に行われると考えられる．

表 **7**　演算引数の有効桁数を元にしたアルゴリズムの有効桁数

	10	16	17	実際の精度	従来手法
x1	23	20	16	27	0
x2	24	21	17	26	2
x3	26	23	19	26	6
x4	27	24	20	26	9
x5	28	25	21	26	11
x6	29	26	22	26	13
x7	29	26	22	26	14
x8	29	26	22	26	14

© 2014 Information Processing Society of Japan

最後に，桁落ち率に注目したアルゴリズムを使用して計算を行った．結果を表 8 に示す．Algorithm 4 の TH は 16bit とした．この結果から，x1 から x3 までの有効桁数

表 8 桁落ち率に着目したアルゴリズムの有効桁数

	提案手法	実際の精度	従来手法	桁落ち
x1	12	27	0	13 回
x2	15	26	2	12 回
x3	19	26	6	11 回
x4	22	26	9	10 回
x5	24	26	11	9 回
x6	25	26	13	8 回
x7	26	26	14	7 回
x8	26	26	14	7 回

を厳しく見積もってしまっているが，それ以降は妥当な有効桁数を見積もっている．これは，ヒルベルト行列を係数とする連立方程式のような悪条件下では常に桁落ち率が高く，さらに，計算の序盤で大きく桁落ちが発生するためであると考えられる．

以上の 2 つの例の結果から，3 つめの提案手法である，桁落ち率に着目したアルゴリズムが，最も良好な有効桁数を見積もっていることが確認できる．このように，単純に桁落ちビット数を追跡しても，妥当な有効桁数を見積もることは出来ない．一方，提案手法であれば，底がネイピア数の指数関数のテイラー展開を計算する場合のような，桁落ちが頻発して，最終的には精度が 0 になってしまう計算や，ヒルベルト行列を係数とする連立方程式といった，異なる性質をもつ計算においても妥当な有効桁数を見積もる事が可能であることを確認した．

6. 有効桁数を追跡する浮動小数点演算器の実装に向けた実験

我々は以前に，桁落ちを検出する浮動小数点演算器 [3] を開発した．今後，この演算器を発展させ，有効桁数を追跡できる様にしたいと考えている．そこで，PC エミュレータの一種である QEMU を用いて，QEMU が持つ浮動小数点演算器に，演算引数の有効桁数を元にするアルゴリズムを適応して，実際に有効桁数が見積もれるか実験を行った．このアルゴリズムを選択した理由は，ハードウェアでは，オペレーターオーバーロードのように，一つの変数に多彩な情報を保持させるのが困難なためである．また，浮動小数点数に有効桁数の情報を持たせるために，倍精度のフォーマットをベースとし，仮数部の下位 8bit に有効桁数保持するフォーマット図 2 を使用した．実験には，アルゴリズムを評価した際と同じ，ヒルベルト行列を係数とする 8 元連立方程式を用いた．その結果を表 9 に示す．提案手法の TH は 16bit に設定した．

この表より，ある程度正確な有効桁数が見積もれること

図 2 有効桁数を保持するフォーマット

表 9 QEMU 上で提案手法を適応した場合の有効桁数の比較

	提案手法	実際の精度	従来手法
x1	11	19	0
x2	13	18	0
x3	15	18	0
x4	16	18	1
x5	17	18	3
x6	18	18	5
x7	18	18	6
x8	18	18	6

を確認した．精度が低下してる理由は，有効桁数の情報を浮動小数点表現に持たせるために，下位 8bit を使用したためである．

現在は，演算引数の有効桁数を元にするアルゴリズムを用いているが，今後は桁落ち率を用いるアルゴリズムの実装を検討したい．

7. まとめ

浮動小数点演算において，桁落ちを対象に有効桁数を見積もる従来手法を元に，桁落ちビット数を変化させることで，異なる性質の計算において，より実際の精度に近い有効桁数を計算するアルゴリズムの検討を行った．その結果，桁落ち率に着目し，さらに演算引数の有効桁数を用いるアルゴリズムが最も良好な有効桁数を見積もれることを確認した．さらに，有効桁数を追跡する浮動小数点演算器の実装に向けて，PC エミュレータの一種である QEMU に，演算引数の有効桁数を元に有効桁数を計算するアルゴリズムを使用するため，専用のフォーマットを用いて実験を行い，有効桁数が見積もれることを確認した．

今後の課題として，桁落ち回数や，有効桁数の情報を活用し，より幅広い計算に対して妥当な有効桁数が見積もれるように，動的に桁落ちビット数を可変させることや，桁落ちによる，仮数部の下位桁へ挿入される 0 が，実際に正しいかどうかを判断して，有効桁数の見積りに生かせる機構の開発が挙げられる．

参考文献

[1] IEEE Computer Society: IEEE Standard for Floating-Point Arithmetic IEEE Standard 754 (2008).
[2] 鈴木弘，大岩元：浮動小数点演算における精度見積もりアルゴリズムとその評価，*1994-HPC-53*, Vol. 94, pp. 27–33 (1994).
[3] 金子啓太，北村俊明：精度低下検出を行う浮動小数点演算器の検討と評価，*2011-ARC-193*, Vol. 16, pp. 1–6 (2011).

© 2014 Information Processing Society of Japan