様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　２８　年　　５　月　２９　日現在

機関番号：　２５４０３
研究種目：　基盤研究(C)（一般）
研究期間：　2013～2015
課題番号：　２５３３００７０

研究課題名（和文）数値計算におけるデータの実用的有効桁数を追跡する計算機構の開発

研究課題名（英文）Development of a practical tracing system for significant digits of variables in numerical calculations

研究代表者

北村　俊明（Kitamura, Toshiaki）

広島市立大学・情報科学研究科・教授

研究者番号：１０３２４６８３

交付決定額（研究期間全体）：（直接経費）　　3,800,000円

研究成果の概要（和文）：浮動小数点演算での精度低下により、数値計算で誤った結果を信用してしまう危険性がある。本研究では、桁落ちを検出することで、変数の有効桁数の追跡を行い、最終結果の有効桁数を推測することで、数値計算の信頼性向上を図るものである。厳密な有効桁追跡は前進誤差解析と同じで悲観的すぎるため、実用的なアルゴリズムとして、(1)丸め誤差は考慮しない、(2)演算回数に対しての桁落ち発生率を導入し、すでに有効桁が少なく桁落ち率が高いときは桁落ちした桁数の影響度を弱める、という方法で有効性を確認した。また、実用的システム構築の実現性についても検討した。

研究成果の概要（英文）：There is a risk which trusts a wrong result of numerical calculations due to a precision decline at a floating-point operation. In this research, to improve the reliability of numerical calculations, we plan to trace the significant digits by detecting cancellation of significant digits of variables at each operation, and to give the significant digits of final results. Unfortunately, a strict algorithm to find the significant digits of the result of operation from those of input operands, which is same as a forward error analysis, is too pessimistic. So, we propose more realistic algorithm to find a significant digits of result; (1) neglect the rounding error; (2) introduce “the occurrence rate of cancellation of significant digits”, and if the input operand’s significant digits are already small and that rate is high, we decrease the influence of cancelation to the result. We verified the validity of this algorithm, and also we studied feasibility of practical system development.

研究分野：　総合領域：情報学、計算機システム・ネットワーク

キーワード：　計算機アーキテクチャ　演算精度低下検出　長精度計算　桁落ち

１．研究開始当初の背景

コンピュータによる数値計算では、

- 演算結果の丸め
- オーバフロー／アンダフロー
- 桁落ち
- 情報埋没

などの要因により、計算した結果が正しい答えとなっていない場合がある。従来から、数値計算を専門に行っている研究者は、誤差解析や演算精度を変えて検算をするなど、多大な努力を払って結果の信頼性を確保してきた。また、これらの数値計算で使用される浮動小数点データ表現も IEEE754 という規格で統一され、表現形式の違いによる結果の異なりが無くなっただけでなく、オーバフロー／アンダフローのハードウェアによる検出、不正確例外による丸め誤差の検出（実態は、ほとんどの演算で検出されるため実用的ではないが）も規格として採用され上記の計算誤りに対して一部は注意が払われるようになった。

しかし、コンピュータの利用が一般的になり、日常生活で欠かせないシステムとなっているがこのような計算誤りに対する認識は、充分に認知されているとは言えず、「計算機の出した結果だから正しい」と思い込んでいる利用者も多い。実際、このような誤差や桁落ちの解析を行うには、充分な知識が必要とされ、一般利用者にそれを期待することは困難である。そこで、桁落ちや情報埋没の発生を検出するハードウェア機構を、浮動小数点演算器に付加することでユーザに通知するシステムを開発した。しかし、計算途中での精度低下が、最終的な演算結果の精度にどれほどの影響を及ぼすかは、一概には言えない。このことは、ある演算での精度低下を検知し報告することは、それを用いて最終結果の信頼性に結び付けられる知識を持った使用者には有用であるが、とにかく結果だけが欲しいような利用者にとっては、かえって混乱を招くことになる可能性もある。

そこで、今回、精度低下を検出しつつ、最終的な結果に対してどのような影響を持っているのかを、各変数、演算ごとに追跡し最終結果の信頼度、すなわち有効桁数を提示することを考えた。この時、丸め誤差も含めた厳密な有効桁数を求めると、前進誤差解析と同等となり、その結果は実際の精度に比べて非常に厳しいものとなり、往々にして有効桁数なしという結果に陥る。このため、実用的な有効桁数を求めるために、数学的には正確ではないが、実用的な有効桁数を求めるアルゴリズムを検討する必要がある。先行研究として、鈴木、大岩らによる研究（1994-HPC-53, V.94, pp.27-33, 1994）があり、この中で提案されているアルゴリズムは、多くのケースで近い精度を提示できているが、幾つかのケースでは、かなり厳しい予測となっており、改良の余地があると考えられる。

２．研究の目的

数値計算において、演算ごとに入力変数の有効桁数から結果の有効桁数を実用的に推測するアルゴリズムを検討し、実際に大規模な数値計算での実証実験を行って、その有効性を検証する。これにより、数値計算の誤差に対する知識を要求することなく、広く一般の人が信頼性の高い数値計算処理を実行可能となり、また、これらの知識を持つ人に対しても、結果の検証などの労力を簡略化することを目的とする。

３．研究の方法

(1) 有効桁数更新アルゴリズムの仕様検討

厳密な誤差解析であると、同じ有効桁数の演算でも丸め誤差を含む桁を演算することで誤差が拡大し有効桁が減少する。しかし、この考え方で有効桁数を求めると有効桁数が無くなってしまい現実的ではない。そこで、桁落ちなどを主に監視し、その他の条件による有効桁数計算は確率的に正しそうな桁数とする楽観的なアルゴリズムを考える。

(2) 有効桁数更新アルゴリズムの検証

上記のアルゴリズム（複数の候補を出すことを想定）に対し、それがどの程度の妥当性を示すか検証を行なう。これには、前研究で用いた手法の一つである仮想計算機システム QEMU を用い、シミュレーション環境を作ることや C++のオペレーターオーバーロードを用いることを考えている。変更すべき箇所等ノウハウが蓄積できているので、迅速に開発できると考えている。評価は、４倍精度などの長精度計算によって得られた結果から正しいと考えられる有効桁数を推定し、それと比較することで行ないたい。

(3) 有効桁数情報を持つ浮動小数点演算機構の検討

大幅な精度低下をもたらす真の減算時の正規化における桁落ちの桁数検出は、ハードウェアの浮動小数点加減算器にごく少量のハードウェアを追加することで可能である。有効桁情報や、それに付帯する情報を記録するためには、当初予定では、IEEE754 浮動小数点フォーマットの一部を使うことを考えたが、それにより仮数部の有効桁数が大幅に減少せざるをえないことになり、別領域に格納する必要が出た。このため、ハードウェア、ソフトウェアの機能分担を考え、システムとしての論理仕様を検討する。

(4) 実証検証

仮想計算機での演算環境では、その他の研究で行なわれているように実行時間に問題がある。このため真に必要とされている大規模数値計算の応用を実際に実行することは時間的に困難である。このため、最新のスーパーコンピュータの性能には遥かに及ばないが、実用的な速度で実行できる環境を作成し、広くユーザを募って実証実験を行ないたい。前研究では FPGA を用いたシステムを作成したがこれでは充分な速度を得られず、古

いテクノロジではあるが 0.18um の CMOS プロセスで実チップを開発する。既に桁落ち検出の演算器を実装した経験が有り、必要なチップ面積等見通しはできている。またこれにより詳細なハードウェア量のオーバーヘッド見積りも可能となる。

４．研究成果

(1) 先行研究

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[1]を紹介する。これは、仮数部の上位の方まで影響を及ぼす桁落ちに着目し、その影響のみを考慮して有効桁数を追跡する。アルゴリズムは、入力オペランドの有効桁数の少ない方から桁落ち桁数を引いたものとするものである。ここで、入力オペランドの指数部が異なる場合は、演算開始時に桁合わせが行われるため、指数部の小さいオペランドの有効桁数は、指数部の差だけ増やして比較する。桁落ち桁数は、入力オペランドの大きい方の指数部と、結果の指数部の差になる。

真の減算を行う場合以外は、桁落ちは発生しないので、入力オペランドの有効桁数の少ない方を結果の有効桁数とする。

(2) 提案アルゴリズム

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では、良い見積もりができていたが、ヒルベルト行列を係数とする連立方程式の計算のような、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れる。そこで提案手法は、桁落ちビット数を単純に有効桁数から引くのではなく、三つのパターンを考え、有効桁数の計算を行った。

① 桁落ち回数を基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 1 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* はユーザが設定可能な桁落ち回数のしきい値である。

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 Algorithm 2 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。

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 3 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。

Algorithm 1 演算引数の桁落ち回数を基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
```

Algorithm 2 演算引数の有効桁数を基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
```

Algorithm 3 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)and((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
```

(3) アルゴリズムの評価

先行研究と同様に、上記のアルゴリズムを C++のオペレーターオーバーロードを用いて実装し、２つの評価ワークロードを用いて評価を行った。

① ネイピア数を底とする指数関数のテイラー展開

$e^x = 1 + x + x^2/2! + \cdots + x^n/n! + \cdots$ において、x に負の値を代入して計算を行うと桁落ちが発生する例である。従来手法で実際の精度に近い見積もりができていたものである。

評価結果を２進桁数で表したものを表１に、１０進数で表したものを表２に示す。

表 1 各アルゴリズムの２進桁数による比較

x	実際	A1	A2	A3	従来	桁落
-4	44	53	44	44	44	12 回
-7	35	51	36	34	34	21 回
-9	28	44	37	29	29	25 回
-12	22	46	31	21	21	38 回
-17	6	43	27	7	7	53 回
-19	0	44	27	3	3	59 回
-21	0	42	25	0	1	63 回

表 2　10 進数での比較（e^x）

x	倍精度結果	正解値
-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、アルゴリズム１と２では、かなりゆるく見積もってしまうため、アルゴリズムとしてはかなり危険であることが分かった。しかし、アルゴリズム３は、従来手法と同等の予測ができることが分かった。

② ヒルベルト行列を係数とする連立方程式のガウス法による求解

従来手法では、有効桁数を非常に厳しく見積もる結果となる、ヒルベルト行列を係数とする以下の８元連立方程式を、ガウスの消去法で求解した。

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

表３にアルゴリズム３の結果を従来手法の結果とともに示す。

表 3 アルゴリズム 3 の 2 進桁数による比較

	実際	*TH*=16	従来	桁落ち
x_1	27	12	0	13 回
x_2	26	15	2	12 回
x_3	26	19	6	11 回
x_4	26	22	9	10 回
x_5	26	24	11	9 回
x_6	26	25	13	8 回
x_7	26	26	14	7 回
x_8	26	26	14	7 回

x_1からx_3までの有効桁数は厳しめに見積もっているものの、それ以降は、実際に近い見積もりとなっている。

③ ヒルベルト行列を係数とする連立方程式の CG 法による求解

CG 法のような解が更新される反復法では、追跡に必要な情報を保持し続けると見積もり結果の有効桁数が 0 になってしまう問題があった。そこで、解が更新される場合にはユーザが追跡情報をリセットする操作を行うことである程度の見積もり結果を得ることができた。表４に結果を示す。

表 4 CG 法による比較

	実際	*TH*=16	桁落ち
x_1	25	29	10 回
x_2	24	29	10 回
x_3	24	29	10 回
x_4	24	29	10 回
x_5	23	53	10 回
x_6	24	53	10 回
x_7	23	29	10 回
x_8	23	29	10 回

かなり緩く見積もってしまっており、特にx_5,x_6は大幅に間違っている。これらの改善は、今後の課題である。

(4) 実証実験のためのシステム設計検討

実用的な数値計算において本提案のシステムを用い、実際に結果の有効桁数や、この精度追跡を行うことの有効性を検証するために、実証実験を行うことを検討している。

このため、実際に VDEC を通じてローム 0.18um プロセスで LSI を試作した。残念ながら、試作したチップには、設計上の不具合があり、使用することはできなかったが、不具合を修正した設計データを用いて、CAD ツールの出力結果から、実装規模を評価した。システムとしては、使用するテクノロジの制約から、市販のプロセッサのような動作速度は期待できないため、実用的な実行時間を目指すために、ベクトル処理手法を用いて高性能化を図った。図１に概略図を示す。

また、精度評価を行うためにも、IEEE754 の４倍精度演算もサポートした。ただし、ハ

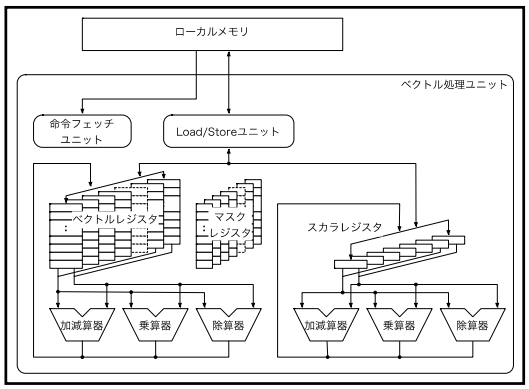

図 1　システム概略図

ードウェアを効率よく使用するために、４倍精度演算は、倍精度演算器を２つ連結する形で実現した。ベクトル演算機構としては、同時に加減算系と乗除算系の２種類の演算が可能であり、また、演算器はそれぞれ倍精度では８エレメントを同時演算できる（４倍精度では４エレメントを２サイクル毎）。

論理ゲートの面積としては、ほぼ 10.5㎜^{2} 程度で収まり、配線領域を取っても、5㎜ 角程度で実装可能である。図２に各機能ブロックの面積比を示す。

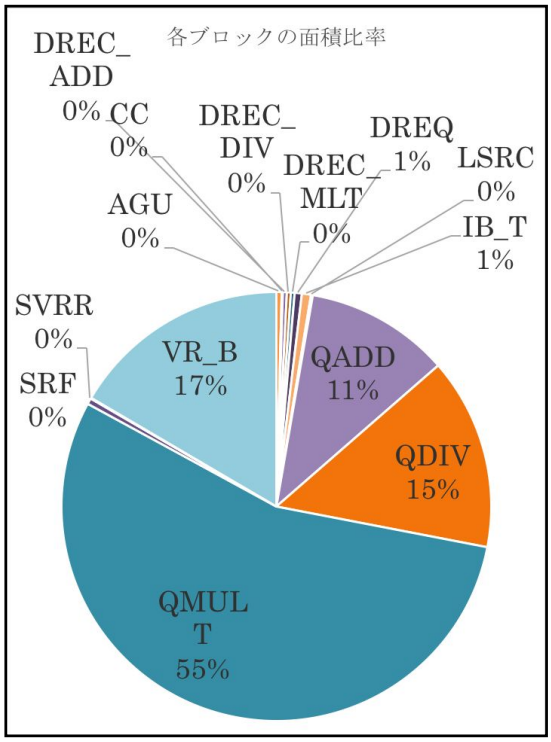

図 2　各機能ブロックの面積比

面積比からわかるように、乗算器が半分程度の面積を占めている。除算器は、性能を追求せずに、処理時間はかかるが面積の小さな構成法を採ったので、比率は小さい。有効桁追跡に関してのハードウェアサポートは加減算器で桁落ち桁数（すなわち、正規化時のシフト桁数）を出力するだけなので、増加する面積は QADD の１%以下であり、全体としては、ほとんど無視できる程度である。

今後、実証実験を実施するためには、アルゴリズム３のさらなる改良と、それをインプリメントするソフトウェア環境の開発が課題となる。

＜参考文献＞

[1] 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、1944-HPC-53、Vol.94、pp.27-33(1994)

５．主な発表論文等

〔雑誌論文〕（計　０件）

〔学会発表〕（計　２件）

① 安仁屋宗石、北村俊明：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、2014-ARC-209、pp,1-6(2014)

② 安仁屋宗石、北村俊明：精度保証に向けてのコンピュータ・アーキテクチャからの取り組み、応用数理学会 2015 年年会、(2015)

〔その他〕

ホームページ等

６．研究組織

(1)研究代表者
北村　俊明（KITAMURA, Toshiaki）
広島市立大学・大学院情報科学研究科・教授
研究者番号：10324683
(3)研究協力者
安仁屋宗石（ANIYA Souseki）
広島市立大学・大学院情報科学研究科・博士後期課程学生