Vol.2021-ARC-245 No.3
2021/7/20

Ties to max magnitude丸めモードを利用した二段階丸め誤差の防止

小泉 透[1,a)]　入江 英嗣[1]　坂井 修一[1]

概要：浮動小数点数計算において，結果を一旦中間精度に丸めてから最終精度に丸める「二段階丸め」は，直接最終精度に丸めた場合と異なる結果を生むことがある．この問題を回避するアルゴリズムは知られているものの，特殊なハードウェアもしくは多数の浮動小数点数演算が必要である．本研究では，RISC-Vの浮動小数点数演算命令で使用可能な丸めモードを組み合わせることで、二段階丸めを行いつつも最近接丸めを保証できるソフトウェアアルゴリズムを提案する．提案アルゴリズムは，既存アルゴリズムと比べて浮動小数点数演算命令を 78%削減した．

1. はじめに

浮動小数点数演算において，丸め（rounding）は欠かせないものである．一般に浮動小数点数同士の和や積は同じ幅の浮動小数点数では表せない．したがって，計算を続けるためには，演算結果を何らかの方法で浮動小数点数に変換する必要がある．この時に行われる変換が，丸めである．

丸めモード，つまり演算結果をどのような戦略で浮動小数点数に変換するかは，大きく分けて四つ存在する [1]．最もよく使われるのは，最近接丸め（rounding to nearest）である．最近接丸めでは，演算結果に最も近い浮動小数点数を選択する．それ以外の丸め方法として，正の無限大への丸め（rounding toward positive），負の無限大への丸め（rounding toward negative），零への丸め（rounding toward zero）がある．正の無限大への丸めでは，演算結果を下回らない最小の浮動小数点数を選択する．負の無限大への丸めでは，演算結果を上回らない最大の浮動小数点数を選択する．零への丸めでは，演算結果の絶対値を超えない範囲で演算結果に最も近い浮動小数点数を選択する．

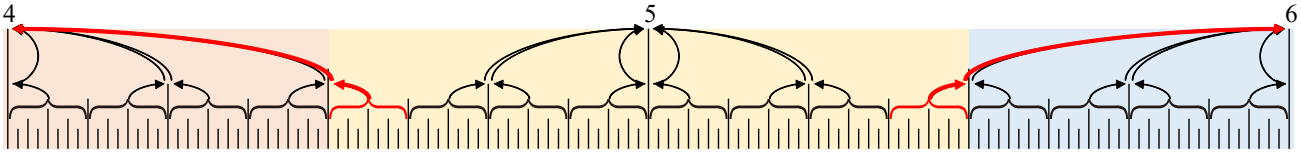
最近接丸めは最も使われる丸めモードであるが，ほかの丸めモードには存在しない注意すべき性質が二つある．一つ目は，「最も近い浮動小数点数」が一意に定まるとは限らないことである．二つ目は，丸めた結果を再び丸める場合，直接一回で丸めた場合と異なる結果を生む可能性が存在することである．

一つ目の問題には解決策があり，よく用いられる方法が二つある．この問題が発生するのは，隣接する浮動小数点数のちょうど中点が演算結果となった場合である．よく使われる解決法は，絶対値が大きいものを選ぶ方法（rounding to nearest, ties to away）および仮数部の末尾が 0 であるものを選ぶ方法（rounding to nearest, ties to even）である．前者は我々になじみ深い四捨五入と同様の動作をするものである．後者は丸める方向を一定にしないことでバイアスを避ける工夫である．以下では，前者を**四捨五入丸め**，後者を**偶数丸め**と呼ぶ．

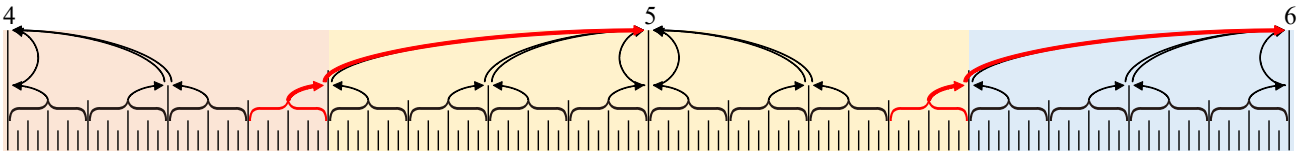
二つ目の問題は，二段階丸め（double rounding）と呼ばれ，一度丸めた結果をそれより低い精度にもう一度丸める場合に発生する．例えば，7.46 を四捨五入し 7.5 とした後，再び四捨五入して 8 とする場合などである．7.46 を直接整数に丸める場合，最近接の整数である 7 に丸めるべきだが，それと異なる 8 が得られてしまっている．この問題は，`double` で計算した後 `float` に丸める，といった場合にも発生する．演算結果は自動的に丸められてしまうため，二段階丸めが問題となることがわかっていてもこの問題を避けることは難しい．

本論文では，二段階丸めを行っても最近接丸めを保証できるソフトウェアアルゴリズムを提案する．提案ソフトウェアアルゴリズムでは，一段階目の丸めに零への丸めを使い，二段階目の丸めに四捨五入丸めを使う．提案ソフトウェアアルゴリズムは，既存のソフトウェアアルゴリズムと比べて浮動小数点数演算命令数を 78%削減した．提案手法は RISC-V の浮動小数点数演算命令で利用可能な丸めモードのみを用いており，特別なハードウェアを要求したり浮動小数点数演算命令以外の命令を使用したりはしない．したがって，提案手法は実プロセッサ上でも高速に動作すると思われる．

[1] 東京大学 大学院情報理工学系研究科
[a)] koizumi@mtl.t.u-tokyo.ac.jp



(a) 二段階目の丸めに偶数丸めを用いる場合.



(b) 二段階目の丸めに四捨五入丸めを用いる場合.

図 1 二段階丸めにより不正確な丸めが行われる原理．直接丸めた場合と結果が異なる場合を赤太矢印で示した.

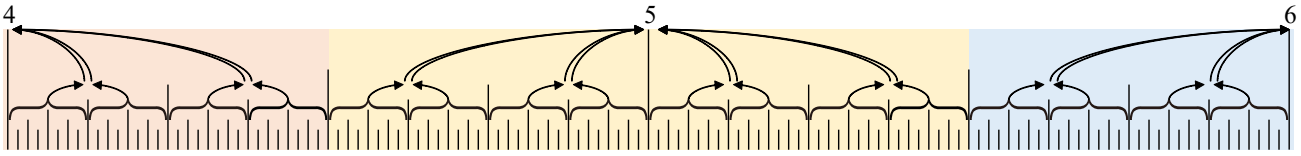


図 2 奇数丸めを用いたアルゴリズムが不正確な丸めを行わない原理．一段階目の丸めの結果が二段階目の丸めにおける丸め境界と一致することがないため，二段階丸めの問題が生じない.

2. 二段階丸め

二段階丸めの問題は，最近接丸めを二回連続で行った場合に発生しうるが，常に発生するわけではない．具体的には，以下の二条件がそろったときにのみ発生する．(1) 一段階目の丸めの結果が，二段階目の丸めの丸め境界と一致した場合．(2) 一段階目の丸めと二段階目の丸めの方向が一致した場合．ここで，最近接丸めの**丸め境界**とは，最近接の丸め先が二つあるような数のことである．また，丸めの**方向**とは，丸める前の値を v，丸めた後の値を r としたとき，$r - v$ の符号のことである．

図 1 は，二段階丸めがなぜ不正確な丸めにつながるかを図示したものである．結果的に正確な丸めが行われる場合を黒矢印で，不正確な丸めが行われてしまう場合を赤太矢印で，それぞれ表した．図からもわかるように，(1) 一段階目の丸めの結果が二段階目の丸めの丸め境界（背景色の境界）と一致し，かつ (2) 一段階目の丸めと二段階目の丸めの方向が一致したとき，最近接ではない値に丸められてしまう．

3. 関連研究

3.1 奇数丸めを用いる方法

二段階丸めの問題を解決するために，奇数丸めを用いたアルゴリズムが提案されている [2]．奇数丸めは，浮動小数点数で表せない演算結果を丸める際，演算結果を挟む二つの浮動小数点数のうち仮数部の末尾が 1 となる浮動小数点数を選ぶ丸めモードである．奇数丸めを用いたアルゴリズムでは，一段階目の丸めに奇数丸めを使い，二段階目の丸めに偶数丸めを使う．この方法は，一段階目の丸めの結果が二段階目の丸めにおける丸め境界と一致することがない[*1]ことから，二段階丸めの問題が生じない（図 2）．この事実は，定理証明支援系 Coq を用いて証明されている [2]．

奇数丸めを用いたアルゴリズムは，理論上は興味深いものの，多くの商用プロセッサ上で動かすことには適していない．奇数丸めは IEEE-754 Standard に含まれておらず，多くのプロセッサはこれをサポートしていないためである．精度が最重要視される用途向けに，無誤差変換 [3]・分岐命令・整数演算を利用して奇数丸めを実現する方法が提案されている [2]．また，ハードウェア回路を実装する用途には有用であり，除算器の実装に活用された例がある [4].

3.2 倍倍精度演算を用いる方法

二段階丸めの問題を解決するため，倍倍精度演算が用いられることもある．倍倍精度演算では，値は二つの倍精度浮動小数点数の和として保持され，倍精度浮動小数点数よりも高い精度で演算を行うことができる．倍倍精度演算を用いると，丸め誤差を計測することが可能である [3]．これを組み合わせれば複数の丸め誤差の合計が算出可能であり，二段階丸めの問題を解決することができる．しかし，倍倍精度演算は多くの浮動小数点数演算命令を実行する必要があるため，一般に低速である.

[*1] 丸める前の値が浮動小数点数で表せる値だった場合を除く．その場合はそもそも二段階丸めの問題は発生しない.

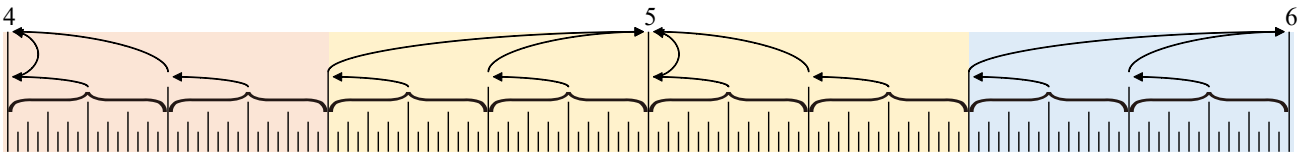


図 3 提案手法が不正確な丸めを行わない原理．一段階目の丸めの結果が二段階の丸め境界と一致する場合，一段階目の丸めの方向と二段階目の丸めの方向は必ず異なる．

4. 提案手法

我々は，一段階目の丸めには零への丸めを使い，二段階目の丸めに四捨五入丸めを用いる方法を提案する．この方法は，二段階丸めを行っているにもかかわらず，直接四捨五入丸めした場合と同じ結果を生成する．

提案手法では，2 章で述べた二条件を同時に満たすことがないため，二段階丸めの問題が生じない．条件「(1) 一段階目の丸めの結果が，二段階目の丸めの丸め境界と一致した場合」は満たすことがあるが，その場合でも条件「(2) 一段階目の丸めと二段階目の丸めの方向が一致した場合」を満たすことはない．なぜなら，条件 (1) が満たされる場合，一段階目の丸めは零に近づく方向に行われ，二段階目の丸めは零から遠ざかる方向に行われるからである．

図 3 は，提案手法がなぜ不正確な丸めを行うことがないのかを図示したものである．図からもわかるように，(1) 一段階目の丸めの結果が二段階目の丸めの丸め境界と一致することはあるが，その場合でも (2) 一段階目の丸めと二段階目の丸めの方向が一致することはない．したがって，提案手法は常に最近接丸めを行う．

提案手法が実現する丸めが最もよく使われる偶数丸めではなく四捨五入丸めであるのは，標準的な丸めモードを組み合わせるだけでは偶数丸めを実現することができないからである．四捨五入丸めはわずかにバイアスがかかるという特性を持つが，これは二重丸めによる誤差の問題よりはるかに軽微な問題であり，通常無視しうる．したがって，最近接丸めが実現できれば，それが偶数丸めでなくても十分有用である．

提案手法は，奇数丸めを用いたアルゴリズムを，RISC-V プロセッサ上で動かすことに適した形に変形したアルゴリズムとみることが可能である．提案手法を用いて一段階目で k-bit に丸める場合，それは奇数丸めを用いたアルゴリズムを用いて一段階目で $(k+1)$-bit に丸める場合と関連する．具体的には，提案手法における一段階目の丸めの結果 Z と，奇数丸めを用いたアルゴリズムにおける一段階目の丸めの結果 O の間には，$Z + 0.5\mathrm{sign}(Z)\mathrm{ulp}(Z) = O$ の関係が成立する[*2]．ここで，$\mathrm{sign}(Z)$ は Z の符号，$\mathrm{ulp}(Z)$ は Z の仮数部の最終桁の重み [5] である．

[*2] ただし，丸める前の値が浮動小数点数で表せる値だった場合，奇数丸めは値を変えないため，$Z = O$ が成立する．この特別扱いは，奇数丸めを用いたアルゴリズムが偶数丸めを実現することに役立っている．

5. 評価

5.1 評価方法

提案手法の有用性の確認は，(1) 正しく二段階丸めを回避できているか，および (2) 既存の二段階丸めを回避するコードと比較し命令数を削減出来ているか，という観点から行った．対象となるルーチンは，指数関数の結果が非正規化数となる場合のルーチンを選択した．このルーチンは，$\mathrm{tbl} \times (1 + \mathrm{poly}) \times 2^{-1022}$ を計算するものであり，$\mathrm{tbl} \times (1 + \mathrm{poly})$ は正規化数となるが $\mathrm{tbl} \times (1 + \mathrm{poly}) \times 2^{-1022}$ は非正規化数となる．非正規化数は正規化数よりも仮数部が短いため，$\mathrm{tbl} \times (1 + \mathrm{poly})$ を計算する際の丸めに加え，2^{-1022} を掛ける際にも丸めが生じる．つまり，評価に用いるこのルーチンは，ナイーブに計算すると二段階丸めによる誤差の影響を受けるものである．既存の二段階丸めを回避するコードとしては，ARM 社が MIT ライセンスで公開している倍倍精度演算を利用したコード [6] を改変して使用した．

以下は，評価に用いたソースコードである．

```
double naive(double tbl, double poly) {
  double t = fma(tbl, poly, tbl);
  double ret = t * 0x1.p-1022;
  return ret;
}

double care(double tbl, double poly) {
  double t = fma(tbl, poly, tbl);
  double err = fma(tbl, poly, tbl - t);
  double hi = 1.0 + t;
  double lo = 1.0 - hi + t + err;
  double ret = fma(hi+lo, 0x1.p-1022, -0x1.p-1022);
  return ret;
}

double proposed(double tbl, double poly) {
  double t;
  asm("fmadd.d %0, %1, %2, %3, rtz"
      : "=f"(t) : "f"(tbl), "f"(poly), "f"(tbl));
  double ret;
  asm("fmul.d %0, %1, %2, rmm"
      : "=f"(ret) : "f"(t), "f"(0x1.p-1022));
  return ret;
}
```

naive は，ナイーブな計算方法を実装したものである．

care は，倍倍精度演算を用いて二段階丸め誤差を回避するコードを fma 関数を用いて書き換えたものである．err は一回目の計算の丸め誤差であり，その計算には Knuth の丸め誤差計測手法 [3] に類似する技術が用いられている．1.0 + t としている部分は，桁落ちを意図的に起こすコードであり，所望の桁での最近接丸めを実現する技法である．1.0 - hi + t はその際の丸め誤差（二回目の丸め誤差）であり，Knuth の丸め誤差計測手法により計算されている．lo は二回の丸め誤差を合計した値であり，これを hi に足しこむことで，二重丸めによる誤差の影響を極力回避している．

proposed は，提案手法を実装したものである．提案手法は標準的な C 言語の範囲で記述できないため，インラインアセンブリを用いて記述した．行われる演算は naive と同じであるが，丸めモード指定のみが異なる．一回目の演算における丸めを指定する rtz は round towards zero の略であり，零への丸めを意味する [7]．二回目の演算における丸めを指定する rmm は round to nearest, ties to max magnitude の略であり，四捨五入丸めを意味する [7]．

このソースコードは，RISC-V 向けの GNU GCC 10.1.0 を用いてコンパイルした．コンパイルオプションは-O2 である．動作の確認は，RISC-V 命令セットシミュレータ Spike 1.0.1-dev を用いて行った．

5.2 二段階丸めを回避できているかの確認

動作の確認は，二重丸めの影響によりナイーブな実装では正確な丸めが行われない引数を一つ用いて行った．具体的には，tbl=1.0，poly=-0x1.3fffe0dec01d9p-26 である．これは，exp(-0x1.6232bdd7d3cd2p+9) を計算する際に実際に現れるものである．tbl × (1 + poly) の計算を無限精度で行った場合，得られる答えは 0x1.ffffff60000f909ff138p-1023 となる．これを正しく丸めた答えとしては，0x1.ffffff60000fap-1023 が期待されるものである．

care および proposed は，正しく丸めた答えである 0x1.ffffff60000fap-1023 を出力した．一方，naive は，誤った答え 0x1.ffffff60000f8p-1023 を出力した．つまり，提案手法は正しく二段階丸めを回避している．

5.3 命令数の比較

命令数の比較は，コンパイルされたコードに含まれる浮動小数点数演算命令の数を数えることで行った．

naive および proposed は，浮動小数点数演算命令を 2 つ含んでいた．一方，care は，浮動小数点数演算命令を 9 つ含んでいた．つまり，提案手法は二段階丸めを回避する既存アルゴリズムより浮動小数点数演算命令が 78%少ない．

6. おわりに

結果を一旦中間精度に丸めてから最終精度に丸める「二段階丸め」は，直接最終精度に丸めた場合と異なる結果を生むことがある．本論文では，標準的な丸めモードを組み合わせることで二段階丸めの問題を解決するソフトウェアアルゴリズムを提案した．実際のルーチンを用いた評価では，提案方法は正しく二段階丸めを回避できることが確かめられた．また，既存の二段階丸めを回避するコードと比較し，浮動小数点数演算命令を 78%削減した．

謝辞 本研究の一部は共同研究「高性能コンピュータシステムに関する研究」（日本電気株式会社）による．

参考文献

[1] IEEE: IEEE Standard for Floating-Point Arithmetic, *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84 (2019).
[2] Boldo, S. and Melquiond, G.: Emulation of a FMA and Correctly Rounded Sums: Proved Algorithms Using Rounding to Odd, *IEEE Transactions on Computers*, Vol. 57, No. 4, pp. 462–471 (2008).
[3] Knuth, D. E.: *The Art of Computer Programming*, Vol. 2, Addison-Wesley, 3rd edition (1998).
[4] Moore, J., Lynch, T. and Kaufmann, M.: A mechanically checked proof of the AMD5/sub K/86/sup TM/ floating-point division program, *IEEE Transactions on Computers*, Vol. 47, No. 9, pp. 913–926 (1998).
[5] Muller, J.-M.: On the definition of ulp(x), pp. 1–16 (2005).
[6] Arm Limited: Double-precision e^x function, https://github.com/ARM-software/optimized-routines/blob/master/math/exp.c (2018).
[7] Waterman, A. and Asanović, K.: The RISC-V Instruction Set Manual Volume I: Unprivlegged ISA (2019).