A-019

# Knuth と Dekker のアルゴリズムに基づく無誤差変換を用いた多倍長演算アルゴリズムの提案と評価

# Proposal and Evaluation of Algorithm for Multiple-Precision Arithmetic Using Error-Free Transformation Based on Knuth's and Dekker's Algorithm

**野口 剛史** † **古賀 雅伸** † **元山 忠** † **矢野 健太郎** ‡
Goshi Noguchi Masanobu Koga Atsushi Motoyama Kentaro Yano

## 1. はじめに

近年，多倍長演算によって有効桁数を十分確保し，丸め誤差等の影響を小さくし，数値的に不安定な問題を解決しようとする考え方が注目されている．C 言語では GMP, MPFR[1]，C++言語では ExFlib[2] と多くの多倍長演算ライブラリが開発されている．また，無誤差変換を使うことで，多倍長演算ライブラリを用いずに，高精度な総和が計算可能なアルゴリズム [3] も提案されている．一方，プラットホーム非依存な環境である Java では，多倍長クラス BigDecimal が標準実装されているが，四則演算のみをサポートしており，科学技術計算には不充分である．

本研究では，[3] のアルゴリズムを基にした，多倍長演算アルゴリズムを提案し，Java で多倍長演算ライブラリとして実装・評価する．実装するライブラリは sin などの数学関数もサポートするので，科学技術計算にも適用可能である．

## 2. 無誤差変換を用いた高精度総和

無誤差変換に用いる Knuth と Dekker のアルゴリズムを説明する．Knuth のアルゴリズム [4] は以下の通りである．

Knuth のアルゴリズム
function$[x,\ y]$=TwoSum$(a,\ b)$
$x = \mathrm{fl}(a+b)$
$c = \mathrm{fl}(x-a)$
$y = \mathrm{fl}((a-(x-c))+(b-c))$

ただし，浮動小数点数の集合を $F$ とし，$a, b, c, x, y \in F$，fl で囲まれた部分は浮動小数点演算を示す．この TwoSum を用いると，厳密に

$$a + b = x + y \quad (|y| \leq \varepsilon|x|)$$

を満たす $x$ と $y$ が得られる．ただし，$\varepsilon$ はマシンイプシロンを示す．次に，Dekker のアルゴリズム [5] を示す．

Dekker のアルゴリズム
function$[x,\ y]$=TwoProduct$(a,\ b)$
$x = \mathrm{fl}(a \times b)$
$[a_H,\ a_L]$=Split$(a)$
$[b_H,\ b_L]$=Split$(b)$
$y = \mathrm{fl}(a_L \times b_L - (((x - a_H \times b_H) - a_L \times b_H) - a_H \times b_L))$

Split アルゴリズム
function$[a_H,\ a_L]$=Split$(a)$
$c = \mathrm{fl}((2^s+1)\cdot a)$
$a_H = \mathrm{fl}(c-(c-a))$
$a_L = \mathrm{fl}(a-a_H)$

ただし，$a_H, a_L, b_H, b_H \in F$，$(s = \frac{\text{仮数部 bit 長}}{2} + 1)$ である．この TwoProduct を用いると，厳密に

$$a \times b = x + y \quad (|y| \leq \varepsilon|x|)$$

を満たす $x$ と $y$ が得られる．次に，TwoSum を利用し，高精度総和を計算する SumK アルゴリズム [3] を示す．ここで $P$ は総和する要素 $p_i (i = 1, \ldots, n)$ を持つ配列である．

SumK アルゴリズム
function res = SumK$(P, \mathrm{K})$
for $k = 1 : \mathrm{K}-1$
$P$= VecSum$(P)$;
end
res= $\mathrm{fl}(p_n + \sum_{i=1}^{n-1} p_i)$

VecSum アルゴリズム
function $P$ = VecSum$(P)$
for $i = 2 : n$
$[p_i, p_{i-1}]$ = TwoSum$(p_i, p_{i-1})$
end

SumK アルゴリズムを用いることで，$K$ 倍の内部精度で求めた総和が計算可能になる．

## 3. 無誤差変換を用いた多倍長演算

### 3.1 NSum アルゴリズム

本研究で考案した NSum アルゴリズムを以下に示す．このアルゴリズムは N 個の浮動小数点数の総和を無誤差で計算できるアルゴリズムである．$|X|$ と $|P|$ は配列の要素数を表す．引数 N は結果 $X$ の要素数であり，$\mathrm{N} = |P|$ で総和を無誤差で計算できる．

NSum アルゴリズム
function $X$ = NSum$(P,\ \mathrm{N})$
$X = \phi$
for $i = 1 : \mathrm{N}$
$p_i$ = SumK$(P, |P|+1)$
$X = X \cup p_i$
$P = P \setminus \{p_i\}$
end
X = renormalization$(X)$

renormalization アルゴリズム
function $X$ = renormalization$(X)$
for $i = 1 : |\mathrm{X}|-1$
for $j = 1 : |\mathrm{X}|-1$
$[a_{|X|-j}, a_{|X|-j+1}]$ = QuickTwoSum$(a_{|X|-j}, a_{|X|-j+1})$
end
end

renormalization は，$|x_{i+1}| \leq \varepsilon|x_i|$ を確実に満たすために行う正規化処理であり，ここでは，$|a| \geq |b|$ の時に使える．

†九州工業大学
‡福岡工業短期大学

TwoSum の高速版である QuickTwoSum を用いている．なお，$P$ の要素が絶対値の降順にソートされているとき，次の QuickNSum アルゴリズムを用いることができる．

QuickNSum アルゴリズム

function $X$ = QuickNSum($P$, N)
$X = \phi$
for $i = 1 : \mathrm{N}$
  $P = \mathrm{VecSum}(P)$
  $X = X \cup p_{|P|}$
  $P = P \setminus \{p_{|P|}\}$
end

### 3.2 データの構成

前節で述べたアルゴリズムを用いた多倍長演算方法を以下に説明する．データの構成としては，$n$ 個の浮動小数点型（double，float など）を用いて，一つの多倍長型を保持する．多倍長型 $A$ は，

$$A = a_1 + a_2 + \cdots + a_n \quad (|a_{i+1}| \leq \varepsilon |a_i|,\ a_i \in F)$$

で計算できる．この多倍長型の精度は，保有する浮動小数点型の精度の $n$ 倍となる．例えば，$a_i$ が double 型で，$n = 4$ の場合，A は 8 倍精度になる．

### 3.3 加減算

提案する多倍長型の加算方法を示す．ただし，merge は絶対値順のマージ処理である．

多倍長型の加算

function $X$ = add($A$, $B$)
$P = \mathrm{merge}(A, B)$
$X = \mathrm{QuickNSum}(P, \max(|\mathrm{A}|, |\mathrm{B}|))$

減算もほぼ同じ方法で実装できる．

### 3.4 乗算

多倍長型の乗算方法を以下に示す．

多倍長型の乗算

function $X$ = multiply($A$, $B$)
$P = \phi$
for $i = 1 : |A|$
  for $j = 1 : |B|$
    $P = \mathrm{merge}(P, \mathrm{TwoProduct}(a_i, b_j))$
  end
end
$X = \mathrm{QuickNSum}(P, \max(|A|, |B|))$

## 4. 性能評価

実装した多倍長ライブラリ JND と，主に整数型で実装されている多倍長演算ライブラリ Apfloat[6] の速度比較のグラフを図 1，図 2 に示す．

JND は加減算においては，10 進 79 桁 (10 倍精度) 程度まで，乗除算においては，10 進 31 桁 (4 倍精度) まで JND の方が高速である．除算に関しては，JND の方がかなり高速である．JND が高精度域で速度が劣る原因としては，JND の演算速度は，QuickNSum の引数である $P$ の要素数 ($|P|$) の 2 乗の比例する為である，なお加減算では，$|P| = 2n$ 乗算では，$|P| = 2n^2$ である．Apfloat の速度上昇が緩やかな理由として，並列処理が行われているからと考えられる．

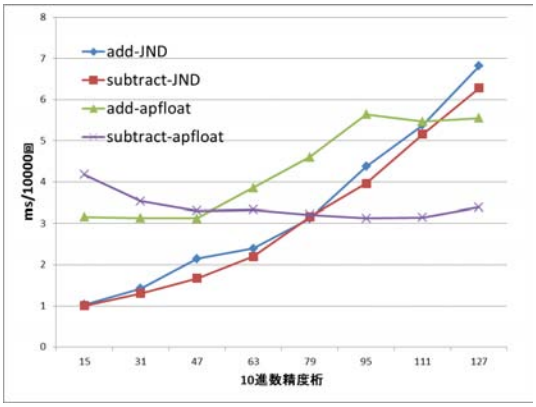


図 1: JND と Apfloat の add,subtract の速度比較

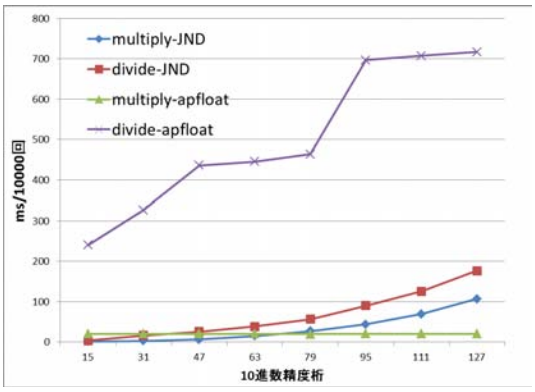


図 2: JND と Apfloat の multiply,divide の速度比較

## 5. まとめ

本研究では，Knuth・Dekker のアルゴリズムを基にした無誤差変換を用いて多倍長演算ライブラリ (JND) を開発した．sqrt，sin などはテイラー展開やニュートン法で実装した為，プラットフォームに依存しない多倍長の科学技術計算が可能となった．また，JND は，汎用多倍長型 MPFloat[7] に対応しているので，MPFloat を使って実装された，半正定値計画問題ソルバー [8] などに応用できる．今後は並列化などを行うことで高速化を行う予定である．

## 参考文献

[1] GMP web pages. http://gmplib.org/.
[2] 藤原宏志. Multiple-Precision Arithmetic Library exflib, 2006. http://www-an.acs.i.kyoto-u.ac.jp/~fujiwara/exflib/.
[3] 荻田武史, Siegfied M.Rump, 大石進一. 高精度内積計算とその応用, 2003.
[4] D.E.Knuth. The Art of Computer Programming （日本語版）. アスキー, 2004.
[5] T.J.Dekker. *A Floating-point Technique for Extending the Available Precision*. Numer. Math, 1971.
[6] Apfloat home pages. http://www.apfloat.org/.
[7] 山村英介, 古賀雅伸, 矢野健太郎, 山田健治. 多倍長計算を用いた制御系設計パッケージ. 第 52 回システム制御情報学会, 2008.
[8] 呉志輝, 古賀雅伸, 中島大雅, 元山忠. Evaluation of semidefinite programming solver based on reliable computing with multiple precision arithmetic. 第 11 回 SICE 制御部門大会, 2011.