Vol. 40 No. 12 情報処理学会論文誌 Dec. 1999

テクニカルノート

ベクトル型インタフェースの対数ルーチン

寒川 光†

スカラー計算機ではスカラー型の数学関数ライブラリが使用されているが，RISC計算機など，スカラーパイプラインを備えた高性能計算機では，多くの引数に対してまとめて関数値を求めるベクトル型にすることで高速化できる．ベクトル型にすることでアンローリングが利用可能となり，パイプラインの稼動率を高められるからである．本論文では高精度テーブル法をベクトル型の対数関数について実装し，ライブラリと同程度の精度を維持したまま，2倍近い性能を達成する方法について述べる．

Logarithm Function with Vector Interface

HIKARU SAMUKAWA†

Elementary mathematical functions with scalar interface are commonly used in scalar computer environment. However, by changing their interface from scalar type to vector type in which many function values are computed in a single call, it is possible to improve their performance, because loop unrolling technique becomes effective. In this paper, we describe an implementation of a logarithm function of accurate table method with vector interface, which keeps same-level accuracy as current library routine and achieves nearly twice performance improvement.

1. はじめに

数値解析プログラムには，計算時間のかなりの部分を1つのライブラリ関数が占めるものがある．必要がある．一般にこのようなケースでは，ループによって多くの関数値が計算されている．

```
do i = 1, n
   x = ···
   y = log(x) + ···
   z(i) = ··· + y
enddo
```

このループを分割すると，多くの関数値をまとめて計算するベクトル型インタフェースの関数（以下，ベクトル型関数）サブルーチンを切り出せる．

```
do i = 1, n
   x(i) = ···
enddo
call logv(x, y, n)
do i = 1, n
   z(i) = ··· + y(i)
enddo
```

$logv$ は $x(1:n)$ を入力として $y(1:n)$ にその対数を返す．

本論文ではこのような状況を前提として，ベクトル型対数関数の，RISC計算機を代表とする，スカラーパイプラインを備える高性能プロセッサを対象とする高速化について述べる．

対数関数などの初等数学関数ルーチンの内部では，Horner法で多項式を計算するものが多く☆，その前後にも引数の近似計算区間への変換やその逆変換といった依存性の強い計算が多い．したがって，スカラー型インタフェースの関数（以下，スカラー型関数）ではパイプラインの稼働率を高めにくい．ベクトル型ならアンローリングが適用できるので，依存性による待ちを消去できる．最近の機種ではアンローリングの効果の上限を4倍と見積もれるものが多い☆☆．したがって，ベクトル型関数を自作すれば，チューニングにより性能向上が期待される．

比較的簡単に自作できる対数関数の近似計算法としては，2進数表現の引数 x の仮数部 f を取り出し，$\log x = \log f + n\log 2$ の関係から基本周期 $[0.75, 1.5)$ などに還元し☆☆☆，$z = (f-1)/(f+1)$ と変数変換して奇関数 $\log((1+z)/(1-z))$ を（倍精度用の場合）15次程度の多項式によって最良近似するのが有力である．しかしこの方法はライブラリと比較して，速度，精度の両面で魅力的でない．

† 日本アイ・ビー・エム株式会社東京基礎研究所
Tokyo Research Laboratory, IBM Research

☆ $(((((c_6 * x + c_5) * x + c_4) * x + \ldots$ の形なので依存性が強い．

☆☆ パイプラインの（本数）×（依存性の深さプラス1）を考える[1]．

☆☆☆ 対数関数は $x = 1$ でゼロをとるため，この近傍で精度を失いやすい．そこで基本周期を $[0.5, 1)$ と $[1, 2)$ を半分ずつ用いる．

これはRISCプロセッサでは除算が乗算に比べて遅いこと，基本周期のすぐ外側で精度を失いやすいことである．たとえば $x = 0.74$ の場合，$\log 1.48 - \log 2 = 0.392... - .693...$ と，2進数で1桁しか違わない2数の引算が精度を失わせる．精度面の改良は，部分的に4倍精度を用いるとか，基本周期を工夫するなど改良可能であるが，速度は（ベクトル型に不向きなため）かなり遅くなる．

そこでライブラリが採用しているような，基本周期を128から512程度の縮小区間に分割して，第 i 番目の区間では，その代表点 f_i でテーブル参照から得る値 $\log f_i$ と小さな修正量から関数値を高精度に計算する**高精度テーブル法**を，アンローリングを適用するために，全縮小区間で同一の処理を行う形で実装した[2]．

以下本論文では，2章で対数関数に対する高精度テーブル法，3章でそのベクトル型関数向きの実装方法，4章で精度と計算性能について述べる．

なお関数の精度に関しては，“数学的に正確な関数値を想定し，これを最も近くの浮動小数点数に**正しく丸める確率**”で評価する．IBMのRISC System/6000（以下，RS/6000）シリーズで提供される数学ライブラリ（以下，ライブラリ）の精度は非常に高く，ほとんどの区間（後述）で約99.9%以上の引数が正しく丸められる．本論文で述べる方法もライブラリと同程度の高精度を維持した．

2. 近似式の計算と精度の制御

本章では一般的に用いられている高精度テーブル法の概要を，倍精度対数関数計算について述べる．

(1) 縮小区間

引数 x を基本周期に変換した値を f，第 i 番目の縮小区間の代表点を f_i とする．$\log f = \log f_i + \log(1 + (f - f_i)/f_i)$ の関係を用いて変数変換し，

$$z = \frac{f - f_i}{f_i} \tag{1}$$

$\log(1+z)$ を次の最良近似による多項式で求める．

$$\log(1+z) = c_n z^n + \cdots + c_2 z^2 + c_1 z \tag{2}$$

(2) 高精度テーブル法

$\log f_i$ をテーブル値 $Tab(f_i)$，近似計算された $\log(1+z)$ を補正項 $Cor(f, f_i)$ と書き改めると，次の一般的な式で表せる．

$$\log f = Tab(f_i) + Cor(f, f_i) \tag{3}$$

これを高精度で計算するためには，絶対値の大きいほうのテーブル値の精度が高くなければならない．その実装法は2通り考えられる．$Tab(f_i)$ を4倍精度にする方法[3]と，代表点 f_i を区間の中点から微小量 ϵ_i 振って，$\log f_i$ が倍精度で正確に表現される（54ビット目以降にゼロが並ぶ）ようにする方法である[4]．後者のほうが計算量を増やさずに実装できるのでこれを用いた．

次に式(3)が正しい丸めを実現する確率を考える．説明のために右辺の2項はともに正で，$Cor(f, f_i) < Tab(f_i) \times 2^{-3}$ と仮定する．2項の仮数部を2進数で $Tab(f_i) = 1.t_1 t_2 \ldots t_{52} 000 \ldots$ および $Cor(f, f_i) = 0.001 r_1 r_2 \ldots r_{52}$ と小数点の位置を揃えて表すと，その和が正しく丸められない確率 P_r は，$\{r_{50} r_{51} r_{52}\}$ のビット構成が $\{100\}$ あるいは $\{011\}$ となる場合の半分と考えられる[5]．したがって，この場合は $P_r = 2^{-3}$ で，正しく丸められる確率は87.5%となる．このことから，正しく丸められる確率を99.9%に制御するという要請には，$Cor(f, f_i)$ を $Tab(f_i)$ よりも10ビット以上小さくする必要があることが分かる．対数関数は $f = 1$ でゼロをとるので，縮小区間の幅も $\log f_i$ に合わせて細かくする必要がある．さらに $f = 1$ の特に近傍の区間では，テーブル参照せずに，多項式だけで計算する．

(3) もとの周期での関数値

基本周期からの変換に用いる定数は $\log 2 = H + h$ と表したとき，16進数で $H = 3fe62e42fefa3800 = .6931...$，$h = 3d2ef35793c76730 = .5498... \times 10^{-13}$ と分割し，式(2)の最後の $+z$ の前に h の項を加える☆．すなわち

$$(...) \cdot z^2 + n \cdot h + z + \log f_i + n \cdot H \tag{4}$$

を左から右へ，絶対値の小さいものから計算する[5]．

このように高精度テーブル法では高精度の $Tab(f_i)$ を必要とするため，1縮小区間あたり16バイトのテーブルを必要とする（4倍精度の $Tab(f_i)$ か，$\log f_i$ と ϵ_i の近似値である ε_i）．

3. 実装方法

前章で述べた高精度テーブル法はスカラー型を前提としたもので，このままベクトル型にしてもアンローリングを適用できない．以下，ベクトル型のための改変について述べる．

対数関数の高精度テーブル法では，z を十分な精度で求めるかどうかが重要である．Galは $\log f_i$ だけでなく，f_i の逆数も高精度になるような f_i（$\log f_i$ を50ビットめから連続する14ビットがゼロで，かつ f_i の逆数 p_i を53ビットめから連続する10ビットがゼロ）を選び，これもテーブルに加えている[5]．しかしこのような f_i が十分狭い ϵ_i 幅の範囲に見つかるかどうかが明らかでなく，またテーブル生成プログラムの単純化のために，ここではニュートン法反復を1回行い z の精度を改善する方法を用いた．これによって f_i による割算を数回の乗算と加算に置き換えられる．

テーブルは *if* ブロックなしで全区間を同一の処理にするため，1縮小区間あたり4項（32バイト）になった．区間数も増え，1区間あたりのデータ量も増えるので，テーブルサイズはスカラー型に比較するとかなり大きくなる．

なおベクトル型関数サブルーチン *dlogv* と，あわせて同じテーブルを用いてビット単位に同一の結果を出力するスカラー型関数 *dlogs* も用意する．

(1) 縮小区間

引数の仮数部からテーブルのインデックスを *if* ブロックなしで作ると，基本周期は $[0.75, 1.5)$ なので，縮小区間の幅が f が1以下の場合は1以上の場合の半分になる．具体的には，1以上では区間幅が1/512，1以下では1/1024の512区間とした．これらの区間は仮数の上位9ビットを直接対応付け

☆ 対数関数の場合，式(2)の c_1 は倍精度用の最良近似では1になる．

て，第1区間 $[1, 513/512), \ldots$，第512区間 $[1023/1024, 1)$ とする．

また1の最近傍の第509区間から第512区間と第1，第2区間は多項式のみで計算するが，他の区間と処理を同じにするために，代表点を $f_i = 1$ とする（$\varepsilon_i = 0$，$p_i = 1$，$\log f_i = 0$ というテーブル値を使用する）．これらの区間では，区間の中点に $\varepsilon_i = 0$ を加えても代表点にならないので，形式的な区間中点 f_{mi} もテーブルに加え，通常の区間は区間中点を，$f = 1$ の近傍区間は1を f_{mi} にセットしておく．

(2) **近似式**

1の最近傍の6区間をカバーする $[255/256, 257/256]$ で被近似関数を8次多項式によって最良近似した[6]．"8次" はここでライブラリと同程度の精度を維持するために必要になる．区間幅の異なる区間に対しては，区間幅に適した次数と係数を用いるのが最良近似の通常の使用法であるが，同一の処理とするために，幅の狭い区間でも8次多項式を用いる．

(3) **テーブルの作成**

縮小区間の中点 f_{mi} で4倍精度で求めた $\log f_{mi}$ を倍精度に丸めた値を a とすると，関数 $F(x) = \log x - a$ をゼロとする x をニュートン法で求めて f_i の近似値とする．テーブルには a，$\epsilon_i = f_{mi} - f_i$ を倍精度に丸めた ε_i，f_i の逆数を倍精度に丸めた値 p_i，および f_{mi} の4つを入れる．精度的に最も厳しい第3区間では，$f_{m3} = 1029/1024 = 1.0048828125$，$\varepsilon_3 = .815\ldots\times 10^{-17}$，$\log f_3 = .004828\ldots$，$p_3 = .995\ldots$ になっている．f_i は ε_i により，$\log f_i$ は54ビット目以降の暗黙のゼロのためにともに106ビットの精度があるが，p_i の精度は53ビットである．

(4) **精度改良**

$z = ((f - f_{mi}) + \varepsilon_i) * p_i$ と計算すると精度が不足する．精度改良には，近似計算プログラムに1回だけニュートン法反復を入れる方法を用いた．具体的には，除算 $z = a/b$ を逆数 $p \approx 1/b$ を用いて $z \approx a * p$ と1次近似された値から，補正項 $cor = -(b * z - a) * p$ を算出しておき，式(4)の $+z$ の計算で反映する．

これと合わせて加算で発生する誤差も次のように改善する．

$$c = a + b$$
$$cor = (c - a) - b$$
$$\ldots$$
$$c = c - cor$$

この方法を $(f - f_{mi}) + \varepsilon_i$ や式(4)の $+z$ の演算などに適用した（最終結果を計算する段階で補正項どうしを加えてから，最終結果に反映させる）☆．

これらの精度改良は非常に強力で，第3区間の端では z と $\log f_i$ は2進数で2桁の差しかないが，これを10桁の差がある場合と同等の精度まで改善する．これらの精度改良は依存性が強いが，ベクトル型ではアンローリングによって依存性が解消できるので，スカラー型で使用する場合よりも計算速度への影響は小さい．

(5) **特殊な値の処理**

非数 NaN や無限大 Inf などの特別な値に対しては，log(±0) は −Inf，log(負数) は NaNQ，log(+Inf) は +Inf，log(NaN) は NaNQ とする．また不正規化数も特殊扱いする．これらを別処理するために *if* ブロックが必要になる．これは4重のアンローリングを適用した場合，$x(i : i + 3)$ のすべてが，最大の正規化数 (7feffff...f) より小さく，正の最小の正規化数 (0010000...0) よりも大きい場合にループ内で計算し，そうでないものが1つでもあるときはすべてスカラー型関数 *dlogs* によって処理するためのものである．この *if* ブロックは，実際に *dlogs* で処理される値がめったに出現しないので，計算時間に与える影響はそれほど大きなものではない（後述）．

☆ 被加数，加数の絶対値の大小関係が既知なので適用しやすい．

4. 精度の評価と計算速度

4.1 精度の評価

高精度テーブル法では，関数の定義域全体に一様に変数を分散させて誤差を調べても，精度が高すぎて検定しにくい．これは基本周期の外側では関数値の絶対値が大きくなるので，基本周期よりも高い精度が得られるからである．また基本周期全体を乱数によって検定しても，99.99%以上の点に対して正しく丸められる．

そこで特定の比較的狭い区間を限定して調べる方法をとった．各区間で *dlogv* による値を，Fortran ライブラリによる4倍精度 (*qlog*) の値と比較し，正しく丸められる数を数える．同時に Fortran ライブラリの倍精度関数 (*dlog*) も同様の方法で検定し，同程度の精度が達成されているかどうかを調べた．区間の選び方は次の2通りである．

縮小区間ごとの検定 *dlogv* の各縮小区間で一様乱数による100,000点を計算する．

縮小区間に依存しない検定 基本区間を10区間に等分割し，それぞれで一様乱数による100,000点を計算し検定する．ここで最も成績の悪い区間をさらに10区間に等分割し，それぞれで検定する．この操作をもう一度行う．

はじめの検定法で最も成績の悪い区間は，1の最近傍の外側で，99.894%であった（この区間で *dlog* は99.852%）．*dlog* の最も悪い成績は99.747%だったが，*dlog* のほうが成績の良かった区間の数は512区間中381と半数を超えた．

2番目の検定法で最も成績の悪い区間は (0.996124, 0.996125) で，99.851%だった（*dlog* はこの区間で99.850%）．これらの結果から，ほぼライブラリの組込み関数と同程度の精度が達成されたと判定した．

精度改良の効果については，たとえばニュートン法反復による除算の精度改良を外すと，最悪の区間は3番目の区間（1の最近傍の隣）に現れ，96.342%であった（改良付きではテーブルを参照する縮小期間で最悪のものは99.988%）．縮小区間での変数 z を4倍精度（除算も4倍精度）で求め，式(4)の $+z + \log f_i$ を4倍精度で計算すると99.998%になる．P_r で見ると，4倍精度計算で0.002%，倍精度では3.7%，ニュートン法反復は0.01%なので，精度と性能の妥協点としては効果的ということができる．

4.2 計算速度

計算速度は IBM の RS/6000 の 595 型（POWER2, 135 MHz）で測定した．コンパイラは XL Fortran の V5.1

表1 計算時間比
Table 1 Computational time ratio.

関数名	時間比	クロック数	備考
dlog	1.00	59	倍精度 library
qlog	3.22	193	4倍精度 library
dlogs	1.26	75	スカラー型
dlogv	0.57	33	ベクトル型
dlogv	0.50	29	特殊数処理なし

を，オプション "–O3 –qarch=pwr2–qstrict" で使用した☆.

(1) ライブラリとの比較

表1 に *dlog* を1とした計算時間の比を示した．なお計測に用いたデータには特殊な値は含まれない．またデータ点数は2,000,000点である．

ベクトル型の *dlogv* はライブラリの1.8倍の速度を達成している．この計算時間は1関数値を約33クロックで求めているが，これは除算2回程度である．なお浮動小数点演算量は1関数値に対し42flop（このうち約1/3が精度改良）なので，計算機のピーク性能値の約1/3である☆☆.

また，この速度は縮小区間の数を256に減らした版でもほぼ同様であった．すなわち，テーブルのサイズを，キャッシュミスを誘発するほど大きくしなければ，縮小区間の数は多いほうが精度的に有利で，これを節約して，式の次数を上げると，逆効果になる可能性が高い．

(2) 条件付きルーチン

自作の数学関数ルーチンのメリットは，変数に何らかの有利な条件が付けられる場合に大きくなる．たとえば，変数がすべて正の正規化された値である，ということが保証されていれば，特殊な値を考慮するための *if* ブロックを省くことができ，さらに12%の速度向上が得られる．

5. おわりに

対数関数は全縮小区間で被近似関数を同一にする式の変形が可能である☆☆☆．したがって，ループ計算部分では，多項式の係数 c_8 から c_2 をすべてレジスタに置いたままの状態で，異なる縮小区間に入る引数を処理できる．しかしMflop/s性能値で見ると，このような処理形態から直観的に期待されるほどの性能値は得られていない．これはHorner法部分の計算量が全体の1/3程度と少ないことに起因している．ライブラリと比較しても1.8倍という（意外に小さな）性能比は，全区間で同一の処理としたために，大部分の区間では必要以上に高次の多項式を用いていることや，$f=1$ から十分に離れた区間でも精度改良を行うといった，冗長な計算の影響と考えられる．しかしこのような状態であっても，RS/6000ではベクトル型のほうが速いことが分った．高速化はアンローリングによるパイプラインの稼働率の向上によっているので，一般にアンローリングの効果が大きいプロセッサでは同様の性能向上が期待できる．高精度の近似式を作成することは手間のかかる仕事であるが，複数のスカラーパイプラインを搭載し，記憶装置の容量も十分なワークステーションでは，重要なアプローチと考えられる．

自作ルーチンをアプリケーションに埋め込むメリットは，変数の範囲が限定される（たとえば引数がすべて基本周期に入る）場合や，sin と cos をペアで求める場合などでは大きくなる．また注意事項として，自作ルーチンとライブラリルーチンが併用されたとき，格納されていた関数値と計算し直した関数値が比較され（同じ引数について）$\log(a) \neq \log(a)$ という現象が現れることである．これを防ぐためには同じ結果を与えるスカラー型も自作して，そのプログラムでその関数呼び出しをすべて置き換える必要がある．

最後に言語処理系による利用可能性として，ビット単位に同一の結果を与えるスカラー型とベクトル型を用意して，コンパイラがループ変換によってベクトル型を呼び出すようにすれば，ユーザからは透過な形で利用（高速化）できることを付け加えたい．

☆ "–O3" は最適化のレベルが3を意味し，ビット単位の一致を無視した最適化を適用する．この場合 "–qstrict" なしでは，精度改良のためのコードが最適化機能によって移動されて精度改良がなされない．

☆☆ 42 flop を理論最大性能値 $33*4$ で割る．

☆☆☆ 三角関数 $\sin f = \sin f_i \cos(f-f_i) + \cos f_i \sin(f-f_i)$ や指数関数も同一の式に変形でき，同様のMflop/s性能値が期待できる．

参考文献

1) 寒川 光：RISC超高速化プログラミング技法，共立出版 (1995).
2) 寒川 光：ベクトル型インターフェースの対数ルーチン，情報処理学会 HPC 研究会報告，76, pp.61–66 (1999).
3) Tang, P.T.P.: Table-Driven Implementation of the Logarithm Function in IEEE Floating Point Arithmetic, *ACM Trans. Math. Softw.*, Vol.16, pp.378–400 (1990).
4) Agarwal, R.C., Cooley, J.W., Gustavson, F.G., Shearer, J.B., Slishman, G. and Tuckerman, B.: New Scalar and Vector Elementary Functions for the IBM System/370, *IBM J. Res. Develop.*, Vol.30, pp.126–144 (1986).
5) Gal, S. and Bachelis, B.: An Accurate Elementary Mathematical Library for the IEEE Floating Point Standard, *ACM Trans. Math. Softw.*, Vol.17, pp.26–45 (1991).
6) 浜田穂積：近似式のプログラミング，培風館 (1995).

(平成11年5月31日受付)
(平成11年9月 2 日採録)