ハイパフォーマンス
コンピューティング 63－4
（1996. 10. 4）

分散メモリ型並列計算機による多倍長平方根の高速計算法

高橋大介[†] 金田康正[††]

東京大学大学院理学系研究科情報科学専攻[†]
東京大学大型計算機センター[††]

本稿では，分散メモリ型並列計算機において，多倍長桁数の平方根を高速に計算する方法について述べる．多倍長桁数の平方根の計算は，平方根の逆数に収束するニュートン法を適用することにより，多倍長桁数の加減乗算に帰着される．N桁の多倍長桁数の乗算はFFT(高速フーリエ変換)を用いれば$O(N\log N\log\log N)$のオーダーで求まるが，多倍長桁数の乗算の計算コンポーネントであるFFTの計算及び最終結果の正規化の部分を並列化した．さらに，実際の計算におけるキャリーおよびボローが確率的に発生しにくくなるように平方根の計算式を変形することで，高速に求めることができた．

Fast Multiple-Precision Calculation of Square Root on Distributed Memory Parallel Computers

Daisuke TAKAHASHI[†] Yasumasa KANADA[††]

Department of Information Science, Graduate School of Science, University of Tokyo[†]
Computer Centre, University of Tokyo[††]

This paper describes for the fast multiple-precision calculation of the square root on the distributed memory parallel computers. By using Newton method to the reciprocal of the square root, the calculation of the square root can be reduced to the multiple-precision addition, subtraction and multiplication. N digits multiple-precision multiplication can be realized with the computing complexity of $O(N\log N\log\log N)$ with FFT(Fast Fourier Transform). Calculation formula was modified carry or borrow in the for reducing actual calculation, then FFT and normalization parts were parallelized in the actual programs.

1　はじめに

平方根は，あらゆる科学技術計算のなかで最も基本的な演算であり，また使用頻度の高い初等関数でもある．

本稿では，分散メモリ型並列計算機により，多倍長桁数の平方根を高速に計算する方法について述べる．

多倍長桁数の平方根の計算は，平方根の逆数に収束するニュートン法を適用することにより，多倍長桁数の加減乗算に帰着される．N桁の多倍長桁数の乗算はFFT(高速フーリエ変換）を用いれば$O(N \log N \log\log N)$ のオーダーで求まるが，多倍長桁数の乗算の計算コンポーネントであるFFTの計算及び最終結果のFFTの計算及び最終結果の正規化の部分は並列化が可能である．

さらに，実際の計算におけるキャリーおよびボローが確率的に発生しにくくなるように平方根の計算式を変形することで，高速に求めることができる．

以下，2章でニュートン法による平方根の計算について，3章で多倍長桁数の加減乗算ルーチンの並列化について，4章で多倍長桁数の平方根の並列計算アルゴリズム，5章で並列化の評価結果について述べる．

2　ニュートン法による平方根の計算

ニュートン法により$\sqrt{a}$を求めるには，$\sqrt{a}$を$f(x) = x^2 - a = 0$の解として，$f'(x) = 2x$より，

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} \tag{1}$$

となる．ここで，式(1)を少し変形した，

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \tag{2}$$

の形が，教科書で紹介されることが多い．

式(1),(2)にはx_nによる除算が含まれており，多倍長の除算は，一般的に乗算より時間がかかるので，この方法は多倍長桁数の$\sqrt{a}$を求める場合は効率が悪い．

そこで，$\sqrt{a} = a \cdot 1/\sqrt{a}$であることを利用すると，まず$1/\sqrt{a}$を$f(x) = 1/x^2 - a = 0$の解として求め，その結果に$a$を掛けることにより，$\sqrt{a}$を求める．具体的には，$f'(x) = -2x^{-3}$より，

$$\begin{aligned} x_{n+1} &= x_n - \frac{1/x_n^2 - a}{-2x_n^{-3}} \\ &= x_n + \frac{x_n^3(1/x_n^2 - a)}{2} \\ &= x_n + \frac{x_n(1 - ax_n^2)}{2} \end{aligned} \tag{3}$$

となる．

ニュートン法による$\sqrt{a}$の計算は2次の収束，つまり正しく求まる桁数が毎回前回の2倍になるので，始めは初期値で与えた2倍の桁数で計算を始め，毎回桁数を2倍にしていけばよく，始めから全桁数の多倍長計算を行う必要はないことが知られている[1, 2]．

3　多倍長桁数の加減乗算ルーチンの並列化

3.1　多倍長加減算および単精度定数乗算ルーチンの並列化

多倍長桁数同士の加減算や多倍長桁数と単精度定数との乗算は，桁数をNとした場合，明らかに$O(N)$の計算量で行えることが分かる．

しかし，多倍長数同士の加減算や多倍長数と単精度定数との乗算において，並列化を阻害する要因は，キャリーおよびボローの処理である．

しかし，次に示すようなベクトル処理にも適応可能な工夫を行うことで，並列化が可能になる[3]．HPF[4]で記述された，多倍長桁数の並列加算プログラムは，次のようになる．入力となるデータは各要素が10000以下に正規化されて配列MAとMBに入っているものとする．

```
1  MA(1:N)=MA(1:N)+MB(1:N)
2  DO WHILE (ANY(MA(2:N) .GE. 10000))
3     MC(N)=0.0D0
4     MC(1:N-1)=MA(2:N)/10000
5     MA(2:N)=MA(2:N)+MC(2:N)-MC(1:N-1)*10000
6     MA(1)=MA(1)+MC(1)
7  END DO
```

まず，1行目の部分で，キャリーを考慮せずに足し算を行ってしまう．次に，2行目のANY文で，配列MAの各要素に10000以上の値があるかどうかをチェックする．もしあれば，4行目で，キャリーを求めたのちに，5行目でキャリーの補正を行う．なお，MCはキャリーを格納する配列である．

ここで注意したいのは，このキャリーの補正においては，いわゆるキャリーの伝搬を考慮していないと

いうことである．したがって，キャリーは完全に補正されていない場合があるので，各要素が10000未満になるまでループが繰り返されることになる．このプログラムでは，0.99999999…9+0.00000000…1のようにキャリーの伝搬が頻繁に起こる場合は，並列化率が低くなり，性能が低下する恐れがある．

キャリーの伝搬が頻繁に起こる場合は，桁上げ飛び越し (carry skip) 方式 [5] を用いるか，金田がベクトル計算機で行ったように [7]，桁上げ先見 (carry look-ahead) 方式を一次回帰演算で実現し，Recursive Doubling[6] によって並列化することが考えられる．

しかし，これらの工夫によっても，キャリーの伝搬の処理には大きなコストがかかるので，4.1節で述べるように，キャリーの伝搬そのものが発生しないように計算方法を考慮する必要がある．

多倍長桁数同士の減算や多倍長桁数と単精度定数との乗算も，加算と同様にして実現が可能である．

3.2 多倍長乗算ルーチンの並列化

多倍長桁数の乗算のアルゴリズムは種々あるが [8]，今回は，数千桁以上の乗算では，実際的に最も速いと考えられる，FFTを用いた乗算アルゴリズムを使用する場合について，並列化を行った．

FFTによる乗算ルーチンの具体的な実現方法は文献 [9] を参照されたい．

多倍長乗算ルーチンを並列化するにあたっては，FFTおよび正規化の部分の並列化が問題となるが，FFTに関しては，並列アルゴリズムが多く提唱されており，実際にライブラリとして提供されているものも多いので，高性能な並列FFTルーチンが利用できる．

問題は，正規化の処理であるが，多倍長加減算および単精度定数乗算ルーチンの並列化におけるキャリーの処理と本質的には同一であるので，この部分も同様にして並列化できる．

ただし，キャリーの値が1とは限らないので，正規化時にキャリーが出現する確率は，加減算の場合より高くなる．

4 多倍長桁数の平方根の並列計算アルゴリズム

多倍長桁数の平方根の計算には，初期値をx_0として倍精度 (16桁程度) で与えた後に，次の式を反復毎に桁数を倍々にして計算する．

$$x_{n+1} = x_n + \frac{x_n(1-ax_n^2)}{2} \tag{4}$$

多倍長桁数同士の乗算においては，FFTを用いるが，その際分割を行うことにより，フーリエ変換の結果の再利用ができ，全体の計算量が減ることが知られている [10] が，インプリメントが複雑になるため，今回の並列化においてはこれを行わなかった．

4.1 正規化におけるキャリーの伝搬を防ぐ工夫

多倍長桁数の平方根の計算を並列化するには，3章で述べたように，多倍長桁数の加減乗算を並列化したものを用いるが，式 (4) をそのまま用いると，$ax_n^2 \approx 1$ であるので，乗算において正規化の際に，キャリーの伝搬が多発する恐れがある．したがって，さらに次のような変形を行う．

$$\begin{aligned} x_{n+1} &= x_n + \frac{x_n(1-ax_n^2)}{2} \\ &= x_n + \frac{x_n\{a(1-x_n^2)-(a-1)\}}{2} \end{aligned} \tag{5}$$

式 (5) においては，aが無理数であると仮定すると，$a(1-x_n^2) = a - ax_n^2 \approx a-1$ となり，値はランダムになると考えられる．したがって，キャリーの伝搬の起こる確率は低くなる．

しかし，これではaが整数もしくは有限小数の場合に適用できない．その理由としては，x_nは無理数となるが，$x_n^2 \approx \left(\frac{1}{\sqrt{a}}\right)^2 = \frac{1}{a}$となり，$x_n^2$が有限小数になってしまうため，2乗計算において正規化の際に，キャリーの伝搬が多発するからである．

したがって，このような場合は，次のような変形を行う．

$$\begin{aligned} x_{n+1} &= x_n + \frac{x_n(1-ax_n^2)}{2} \\ &= x_n + \frac{x_n\{(1+a-2ax_n)-a(1-x_n)^2\}}{2} \end{aligned} \tag{6}$$

式 (6) において，

$$\begin{aligned} a(1-x_n^2) &= a(1-2x_n+x_n^2) \\ &= a - 2ax_n + ax_n^2 \\ &\approx 1 + a(1-2x_n) \end{aligned}$$

となり，x_nは無理数であるから，キャリーの伝搬の起こる確率は低くなる．

ここで，x_nが無理数であると仮定したが，たとえば$\sqrt{4}=2$のような計算においては，x_nは有限小数となるので，式(6)は適用できない．

したがって，これらの工夫を適用するには，あらかじめ平方根の値がどうなるかを判定する必要がある．

また，これらの式の変形は多倍長桁数の逆数やn乗根にも適用可能である．

4.2 逐次処理の場合の計算時間

多倍長桁数の平方根の計算における計算時間を考察するにあたっては，多倍長桁数同士の乗算だけに注目し，他の演算，例えば多倍長桁数の加減算，多倍長桁数と単長数の乗除算等の計算時間は微小であると考え，無視する．

N桁の多倍長桁数同士の乗算の計算時間を$M(N)=cN\log_2 N$(cは比例定数)とし，N桁の多倍長桁数の平方根の計算では，$1, 2, 4, \cdots, N/2, N$桁のように，桁数を倍々にして反復計算するものとする．

すると，逐次処理の場合の計算時間T_sは，

$$\begin{aligned} T_s &= M(1)+M(2)+\cdots M(N/2)+M(N) \\ &= c(2N(\log_2 N-1)+2) \end{aligned}$$

となる．

4.3 並列処理における演算時間

並列処理の場合の計算時間を考察するにあたっては，最も負荷の集中するプロセッサの計算時間がそのままトータルの計算時間になるものと仮定する．

4.3.1 ブロック分割の場合

ブロック分割の場合には，N桁の多倍長桁数はプロセッサ台数をPとすると，各プロセッサにN/P桁ずつブロック順に格納される．

N桁の多倍長桁数の平方根の計算では，$1, 2, 4, \cdots, N/2, N$桁のように，桁数を倍々にして反復計算するので，ブロック分割では一番目のプロセッサに負荷が集中し，結局全てのプロセッサで計算できるのは，N桁を計算する最後の1回の反復のみとなる．

ブロック分割の場合の演算時間T_{block}は，

$$\begin{aligned} T_{block} &= \left\{M(1)+M(2)+\cdots+M\left(\frac{N}{P}\right)\right\} \\ &\quad +\left\{\frac{1}{2}M\left(\frac{2N}{P}\right)+\frac{1}{4}M\left(\frac{4N}{P}\right)+\cdots+\frac{1}{P}M(N)\right\} \\ &\approx \frac{1}{P}\cdot cN\left\{\log_2 N(\log_2 P+2)-\frac{1}{2}(\log_2 P)^2\right\} \end{aligned}$$

となる．

4.3.2 サイクリック分割の場合

サイクリック分割の場合には，N桁の多倍長桁数はプロセッサ台数をPとすると，各プロセッサにN/P桁ずつサイクリックに格納されるので，ブロック分割の場合のように，負荷が集中することはない．

サイクリック分割の場合の演算時間T_{cyclic}は，

$$\begin{aligned} T_{cyclic} &= \left\{M(1)+\frac{1}{2}M(2)+\cdots+\frac{1}{P}M(P)\right\} \\ &\quad +\frac{1}{P}\left\{M\left(\frac{2N}{P}\right)+M\left(\frac{4N}{P}\right)+\cdots+M(N)\right\} \\ &\approx \frac{1}{P}(2cN\log_2 N) \end{aligned}$$

となり，ブロック分割の場合に比べて演算時間が少なくなっていることが分かる．

4.4 並列処理における通信時間(正規化の部分)

多倍長桁数の平方根を計算する場合において，ここでは多倍長桁数同士の乗算の通信時間を考察する．通信時間は，FFTの部分および正規化の部分に分けられる．ここで，FFTの通信時間は，ブロック分割とサイクリック分割では変わらないと仮定する．したがって，本稿では正規化の部分の通信時間のみを考察することにする．

4.4.1 ブロック分割の場合

ブロック分割の場合には，N桁の多倍長桁数はプロセッサ台数をPとすると，各プロセッサにN/P桁ずつブロック順に格納されている．したがって，多倍長桁数同士の乗算においては，最終的に正規化を行う際のプロセッサ間通信は，隣のプロセッサにキャリーを1(word)だけ送れば良いことになる．

通信におけるレイテンシをL，通信スループットをWとすれば，ブロック分割の場合の通信時間T_{block}は，

$$\begin{aligned} T_{block} &= (\log_2 N+1)\left(L+\frac{1}{W}\right) \\ &= L(\log_2 N+1)+\frac{1}{W}(\log_2 N+1) \end{aligned}$$

となる．

4.4.2 サイクリック分割の場合

サイクリック分割の場合には，N桁の多倍長桁数はプロセッサ台数をPとすると，各プロセッサにN/P桁ずつサイクリックに格納されている．したがって，多倍長桁数同士の乗算においては，最終的に正規化を行う際のプロセッサ間通信は，隣のプロセッサに N/P(word) のキャリーを送らなければならない．そのため，ブロック分割に比べて，通信時間は多くなる．

通信におけるレイテンシをL，通信スループットをWとすれば，ブロック分割の場合の通信時間T_{cyclic}は，

$$\begin{aligned} T_{cyclic} &= \left(L+\frac{1}{W}\right)(\log_2 P+1) \\ &+\left\{\left(L+\frac{2}{W}\right)+\left(L+\frac{4}{W}\right)+\cdots+\left(L+\frac{N/P}{W}\right)\right\} \\ &= L(\log_2 N+1)+\frac{1}{W}\left(\log_2 P+\frac{2N}{P}-1\right) \end{aligned}$$

となり，ブロック分割に比べると通信時間が多くなることが分かる．しかし，演算時間のオーダーは $O((N/P)\log_2 N)$ であることを考えると，正規化における通信時間は，ある程度は無視できるので，演算時間と通信時間を加えた，トータルの計算時間から考えると，サイクリック分割が有利と考えられる．

5 並列化の評価

並列化の評価に際しては，$\sqrt{2}$の計算を，プロセッサ数Pと桁数Nを変化させて，実行時間を測定することにより行った．

計算機としては，MIMD 型並列計算機 SR2201 (256PE, 総主記憶 64GB) を用いた．計算時間の測定に際しては，256PE すべてをシングルユーザーで使用し，経過時間を測定した．

並列 FFT ルーチンとしては，メーカー提供の2基底の実数 FFT を用いた．通信ライブラリには，PARALLELWARE[11, 12] を用いた．プログラムは，全て FORTRAN で記述し，コンパイラは，日立の最適化 FORTRAN77 V02-00 を用い，最適化オプションとして-W0,'pvec,opt(o(s),fold(2),prefetch(1),ischedule(3),rapidcall(1))' を指定した．

多倍長桁数のデータ構造としては，多倍長桁数を10進3桁ごとに区切り，サイクリックに32ビット整数の配列に格納することとする．また，多倍長桁数の乗算において FFT を計算する際には，32ビット整数の配列のデータを64ビット浮動小数点数の配列にコピーしている．

利用可能な主記憶容量の関係で計算桁数は，($N = 3\cdot2^{18} \sim 3\cdot2^{26}$) 桁まで変化させ，プロセッサ数を $P = 1 \sim 256$ と変化させて測定した．

結果は表 1 のようになった．表 1 において，* となっているのは，主記憶容量の不足のために実行できなかったことを示している．

表 1. 多倍長桁数の$\sqrt{2}$の計算時間 (単位 秒)

$P\backslash N$	$3\cdot2^{18}$	$3\cdot2^{20}$	$3\cdot2^{22}$	$3\cdot2^{24}$	$3\cdot2^{26}$
1	5.782	*	*	*	*
2	6.940	26.720	*	*	*
4	3.959	14.263	*	*	*
8	2.756	8.485	34.016	*	*
16	1.883	5.074	18.528	*	*
32	1.977	3.422	11.311	42.048	*
64	2.323	2.967	6.951	24.514	*
128	4.007	5.771	8.946	20.855	56.631
256	6.281	8.349	11.424	20.305	37.203

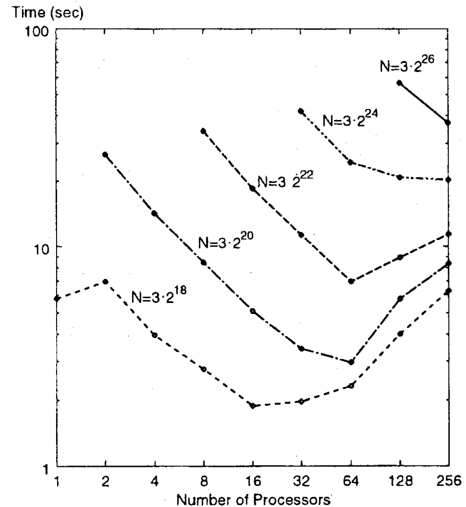

図 1. 多倍長桁数の$\sqrt{2}$の計算時間

多倍長桁数の$\sqrt{2}$の計算時間を図 1 に示す．

図 1 から分かるように，$\sqrt{2}$の計算において，$N = 3\cdot2^{20}, 3\cdot2^{22}$桁では，64 台まではプロセッサ台数の増加に伴い実行時間が短縮されているが，128, 256 台と増加すると，逆に実行時間が増えてしまっている．

これは，多倍長桁数同士の乗算を計算する際に，FFT を計算しているが，プロセッサ数が増加するに従って，一度に送るデータ量が少なくなるため，演算時間に対して通信の立ち上がり時間が無視できなくなってくるためと考えられる．

したがって，桁数Nが小さい時は，並列化の効果はあまり期待できない．今回の結果では，256プ

ロセッサにおいては1億桁を超える辺りから，並列化の効果が発揮できることが分かった．

また，N桁の乗算にFFTを用いた場合，畳み込みや正規化の計算量は，$O(N)$であるが，FFTの計算量は$O(N \log N \log\log N)$であるので，Nが大きくなればなるほど，実行時間におけるFFTの比率が高くなる．

多倍長桁数の平方根の計算においては，多倍長桁数同士の乗算の時間にほぼ比例することから，FFTの計算時間が多倍長桁数の平方根の計算時間に大きく影響する．

なお，256プロセッサで$N = 3 \cdot 2^{26}$桁の$\sqrt{2}$を計算した場合の計算時間におけるFFTの比率は，約85%程度であった．

6 まとめ

本稿では，分散メモリ型並列計算機において，多倍長桁数の平方根を高速に計算する方法について述べた．多倍長桁数の平方根の計算は，平方根の逆数に収束するニュートン法を適用することにより，多倍長桁数の加減乗算に帰着される．N桁の多倍長桁数の乗算はFFT(高速フーリエ変換)を用いれば$O(N \log N \log\log N)$のオーダーで求まるが，多倍長桁数の乗算の計算コンポーネントであるFFTの計算及び最終結果の正規化の部分を並列化した．さらに，実際の計算におけるキャリーおよびボローが確率的に発生しにくくなるように平方根の計算式を変形することで，演算量及び通信量を減らすことができ，高速に求めることができた．

今後の課題としては，プロセッサ数を増やした場合やデータ分割をブロックサイクリックにした場合についての実行時間の詳細な解析及び性能向上が挙げられる．

参考文献

[1] Brent, R.P.: Fast Multiple-Precision Evaluation of Elementary functions, *J. ACM*, Vol.23, No. 2, pp. 242–251 (1976).

[2] 伊理正夫：ニュートン法の実際，数理科学，Vol. 218, pp. 10–16 (1981).

[3] 高橋大介，金田康正：分散メモリ型並列計算機による高速多倍長計算，情報処理学会研究報告 96-HPC-60, pp. 31–36 (1996).

[4] High Performance Fortran Language Specification Version 1.0, High Performance Fortran Forum, (1993).

[5] Lehman, M. and Burla, N.: Skip Techniques for High-Speed Carry Propagation in Binary Arithmetic Units, *IRE Trans. Elec. Comput.*, Vol. Ec-10, pp. 691-698 (1961).

[6] Stone, H.: An Efficient Parallel Algorithm for the Solution of Tradiagonal System of Equcations, *JASM*, Vol. 20, No. 1, pp. 27–38, (1973).

[7] Kanada, Y.: Vectorization of Multiple-Precision Arithmetic Program and 201,326,000 Decimal Digits of π Calculation, *Supercomputing 88: Volume II, Science and Applications, Ed. by Joanne L. Martin and Stephen F. Lundstrom*, IEEE Computer Society Press, pp. 117-128, 1989, IEEE Conference held at Kissimmee FL, Nov. 14-18 '88.

[8] Knuth, D.E.: The Art of Computer Programming, Vol. 2 : Seminumerical Algorithms, Addison-Wesley, Reading, Massachusetts (1981).

[9] 金田康正：計算機によるπの計算，数学セミナー，日本評論社，Vol. 3, pp. 12–18 (1989).

[10] 高橋大介，金田康正：多倍長平方根の高速計算法，情報処理学会研究報告 95-HPC-58, pp. 51–56 (1995).

[11] 日立製作所：HI-UX/MPP PARALLELWARE ユーザーズガイド –FORTRAN– 6A20-3-400 (1996).

[12] 日立製作所：HI-UX/MPP PARALLELWARE リファレンス –FORTRAN– 6A20-3-401 (1996).