ハイパフォーマンス
コンピューティング 67－4
（1997. 8. 19）

分散メモリ型並列計算機による円周率の高精度計算

高橋大介† 金田康正†

本稿では，分散メモリ型並列計算機により高精度の円周率を高速に計算する方法について述べる．多倍長桁数の円周率は，ガウス-ルジャンドルの公式およびボールウェインの4次の収束の公式を用いると効率良く計算できることが知られている．これらの公式には平方根や4乗根，そして逆数計算が含まれているが，それらの計算にはニュートン法が適用できるので，最終的には全ての計算は多倍長桁数の加減乗算に帰着することができる．また，n 桁の多倍長桁数同士の乗算は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の計算および最終結果の正規化の部分を並列化した．その結果，1024プロセッサから成る分散メモリ型並列計算機HITACHI SR2201で515億桁余りの円周率の計算が40時間以内で終了することが分かった．

Fast Multiple-Precision Calculation of π on Distributed Memory Parallel Computers

DAISUKE TAKAHASHI† and YASUMASA KANADA†

This paper describes the fast multiple-precision calulation of π on distributed memory parallel computers. By using Newton method to the multiple-precision calculations of the square root, 4th root and inverse can be reduced to the multiple-precision addition, substraction 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). More than 51.5 billion decimal digits of π were calculated using the Gauss-Legendre algorithm and the Borwein's quartically convergent algorithm on the distributed memory parallel computer HITACHI SR2201.

1. はじめに

円周率の高精度計算は，昔から多くの人々によって行われてきた．円周率の計算の歴史は，数学の発展，計算技術の進歩といった興味ある話題を数多く含んでいる．

今世紀に入ってコンピューターが出現してからは，円周率の計算桁数は飛躍的に伸びることになった．1949年にENIACを用いて円周率2037桁を約70時間かけて求めたのに始まり，1973年にはCDC 7600を用いて100万桁が約23時間で求められた．これらの計算には，Machinの公式として著名な次の等式（あるいは類似の公式）

$$\frac{\pi}{4} = \arctan\frac{1}{5} - 4\arctan\frac{1}{239} \qquad (1)$$

をarctanのテイラー展開，

$$\arctan x = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots \qquad (2)$$

と組み合わせることによって実行されている．しかし，この方法では桁数を n とすると，計算量は $O(n^2)$ となってしまうため，桁数が大きくなると計算時間は桁数の自乗に比例して増加するという欠点があった．

さて，1976年にR. BrentとE. Salaminの2人によって全く独立に再発見された，楕円積分に関するGaussの計算法と，Legendreの関係式を用いた円周率を求める公式[1),2)]（以後，ガウス-ルジャンドルの公式と呼ぶ）を用いることで，n 桁の乗算の計算量を $M(n)$ とした場合，$O(M(n)\log n)$ で円周率が計算できることが示された．n 桁同士の乗算はFFT（高速フーリエ変換）を用いることにより，$M(n) = n \log n \log\log n$ にできることが知られている[3)]ので，結局この方法は桁数が大きくなるにつれてarctanのテイラー展開を用いる方法よりも有利になる．

この公式に基づく円周率の高精度計算については，1976~1979年頃にStanford大学のD. E. Knuthの学生であったR. Finkel, L. GuibasおよびC. Simonyiらによって，SIMD型並列計算機Illiac IVを用いて，概算4時間で3300万ビット（10進で約1000万桁）の円周率を計算する計画があったようである[2),4)]．こ

† 東京大学大型計算機センター
Computer Centre, University of Tokyo

の計算においては，ガウス-ルジャンドルの公式および Schönhage-Strassen の乗算アルゴリズム[5]を用いることになっていた．

しかし計画は困難を極め，求める桁数を 150 万桁程度に縮小したが，結局その計算も成功しないまま計画は失敗に終ったようである[4]．

1982 年以降は主にこのガウス-ルジャンドルの公式が円周率の計算に使用されている．これまでの円周率の記録である，1995 年にベクトル計算機 HITAC S-3800/480 を約 113 時間用いて計算した 64 億 4245 万桁でもこの公式が使われた[6]．

しかし，最近単体プロセッサの性能の伸びは限界に近づいてきていることから，さらに桁数を伸ばすためには，

- スケーラブルな並列処理による計算時間の短縮
- 分散メモリ方式による主記憶容量の増大

が必須である．

D. V. Chudnovsky と G. V. Chudnovsky らは自らが発見した，

$$\frac{1}{\pi} = \frac{6541681608}{640320^{3/2}} \sum_{k=0}^{\infty} \left(\frac{13591409}{545140134} + k\right) \frac{(6k)!}{(3k)!(k!)^3} \frac{(-1)^k}{(640320)^{3k}} \tag{3}$$

の公式を用い，自作の計算機を並列に用いて，円周率を 1996 年 3 月までに約 1 週間程度の計算時間で 80 億桁まで計算したとされている[7]．この方法だと，式 (3) において級数の各項を複数の計算機の各プロセッサに割り当てて独立に計算ができるので，並列実行は比較的容易と考えられる．しかし，この公式は桁数を n とすると，本質的には計算量は $O(n^2)$ となるので，arctan のテイラー展開による計算方法に比べると定数係数は小さくなっている☆とはいえ，桁数が大きくなると計算時間は桁数の自乗に比例して増えていくという欠点がある．

今回，我々が分散メモリ型並列計算機を用いて高精度の円周率を計算するにあたっては，主計算にボールウェインの 4 次の収束の公式[8]を用い，検証計算にガウス-ルジャンドルの公式を用いた．これらの公式には多倍長桁数の平方根や 4 乗根，そして逆数計算が含まれているが，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することができる．n 桁の多倍長桁数同士の乗算は 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 の計算および最終結果の正規化の部分を並列化した．また，FFT は分散メモリ型並列計算機において，高い並列化効率を得られる[9]ことが確認されていることからも分かるように，我々の計算方法は分散メモリ型並列計算機の性能を十分発揮させることができる可能性がある．

以下，第 2 章でガウス-ルジャンドルの公式について，第 3 章でボールウェインの 4 次の収束の公式について述べる．第 4 章で多倍長桁数の加減乗算アルゴリズムについて，第 5 章で多倍長桁数の平方根および 4 乗根，逆数計算アルゴリズムについて述べる．第 6 章で多倍長桁数の加減乗算の並列計算アルゴリズムについて，第 7 章で並列化の評価および円周率 515 億 3960 万桁計算の詳細について述べる．最後の第 8 章はまとめである．

2. ガウス-ルジャンドルの公式

ガウス-ルジャンドルの公式による円周率の計算方法は次のようになる．

$A = 1$, $B = 1/\sqrt{2}$, $T = 1/4$, $X = 1$ として，A と B の差が必要とする精度より大きな間，次に示すカッコ { とカッコ } で囲まれる部分を繰り返し実行する．

$$\{Y := A;\ A := \frac{A+B}{2};\ B := \sqrt{B \times Y};$$
$$T := T - X \times (Y - A)^2;\ X := 2 \times X\}$$

すると，π の値は $(A+B)^2/(4T)$ となる．ただし，A, B, T, Y の各値は，求めようとする精度以上で計算しておく必要がある．

この公式は 2 次の収束を示すので，求めようとする桁数を n とすると，$\log_2 n$ 回程度の反復で n 桁の π の値が求まる☆☆．

3. ボールウェインの 4 次の収束の公式

1980 年代以降，カナダの数学者ボールウェイン兄弟によって，それまでに知られている方法と同等の計算量を有する方法がいくつか発見されている[8]．ボールウェイン兄弟が発見し，これまでに多数桁の円周率計算で使用されている[6),11)]4 次の収束の公式の一つを以下に示す．

$a_0 = 6 - 4\sqrt{2}$ および $y_0 = \sqrt{2} - 1$ として

$$y_{k+1} = \frac{1 - (1 - y_k^4)^{1/4}}{1 + (1 - y_k^4)^{1/4}} \tag{4}$$

$$a_{k+1} = a_k(1 + y_{k+1})^4 - 2^{2k+3} y_{k+1}(1 + y_{k+1} + y_{k+1}^2) \tag{5}$$

の級数を求める精度まで繰り返し計算すると π の値は $1/a_k$．

ただし，a_k, y_k の値は求める精度以上の精度で計算しておく必要がある．

ボールウェインの 4 次の収束の公式においては，反

☆ 足し上げる項 1 つについて，約 14 桁ずつ計算精度が上がっていく．

☆☆ 式を変形することにより，計算量がさらに減ることが指摘されている[10]．

復回数はガウス-ルジャンドルの公式の約半分であるが，反復1回当たりの計算量がガウス-ルジャンドルの公式の約2倍になっていることから，結局トータルの計算量としてはどちらもほぼ同じとなる[10]☆.

4. 多倍長数の加減乗算アルゴリズム

ガウス-ルジャンドルの公式，ボールウェインの4次の収束の公式では，前述の A, B, T, Y 及び a_k, y_k は，正確に計算しようとする桁数+αの精度でもって繰り返し計算を行わなくてはならない☆☆．したがって，いかに速い多倍長演算ルーチンが作成できるかに計算時間の大部分がかかることになる訳である．

4.1 多倍長加減算および単精度定数乗算アルゴリズム

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．

4.2 多倍長乗算アルゴリズム

多倍長桁数の乗算のアルゴリズムは種々あるが[3]，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考察を行う．

浮動小数点演算による FFT で乗算を行う際には誤差の影響が出てくるため，FFT の実現方法にもよるが，IEEE 方式の倍精度浮動小数点計算では，1万進数の場合には最悪の場合，10進換算で3355万桁程度でも誤差の影響が出て，正確に乗算ができない場合がある．

これは，1万進数同士の積を求める際の FFT の計算で，4桁×4桁＝8桁の数値の数千万項の和を求めているために，そこで仮数部が overflow してしまい，その結果情報落ちが生じているのが主な原因である．

しかし，Mersenne 数の素数判定でも使用されている[12]次のような工夫を行うことにより，FFT における誤差の影響を防ぐことが可能である．

例えば，

```
|   3.|1415|9265|3589|7932|
```

のように格納されているデータを

```
|   3.|1416|-735|3590|-2068|
```

のように前処理してから FFT を計算することにより，途中で和を計算した場合に，プラスマイナスで打ち消しあうことになるので，仮数部の overflow を防ぐことができる．

具体的には，以下のような FORTRAN プログラムに示すような処理を行う．

```
1  DO I=N,2,-1
2     IF (X(I) .GE. BASE*0.5D0) THEN
3        X(I)=X(I)-BASE
4        X(I-1)=X(I-1)+1.0D0
5     END IF
6  END DO
```

ここで，BASE は基数であり，配列 X(1:N) には 0～BASE-1 に正規化された値が入っているものとする．

このような工夫を行うことにより，IEEE 方式の倍精度浮動小数点計算では，10万進数の場合でも，10進換算で1億桁程度の計算でも誤差の影響を避けることができ，正確に乗算が行えることが数値実験の結果，確認されている．

この工夫は各桁の数がランダムに出現する場合にのみ適用可能なので，0.44444444…のような無限小数同士の積では FFT の誤差の影響を避けることができない．したがって，このような場合には適当なバイアスを加えてから前処理を行い，FFT を計算した後にバイアス分を引くというような工夫をする必要がある．

また，FFT により多倍長桁数の乗算を行う際は，倍精度実数配列1個に格納できる多倍長桁数は10進で4桁～5桁程度となってしまうために，通常の $O(n^2)$ や Karatsuba の $O(n^{\log_2 3})$ のアルゴリズム[3]に比べて，記憶領域を多く必要とする．このような場合は，二次記憶を用いて FFT を計算する方法も考えられる．しかし，最近のプロセッサは高速化されているにも関わらず，二次記憶装置はそれほど高速化されていないため，かなりの時間が I/O に費やされてしまうことが予想される．実際に並列計算機上で二次記憶を用いた FFT を実現し性能を評価した結果からも分かるように[13]，最近の並列計算機においては，二次記憶を用いて FFT を行うのは現実的でない．

したがって，FFT を主記憶のみで計算する場合には，乗算が可能な桁数が少なくなってしまうが，このような場合には $O(n^{\log_2 3})$ の乗算アルゴリズムと FFT による乗算アルゴリズムを組み合わせて計算することが考えられる．

例えば，FFT で計算可能な桁数が1億桁であるとすれば，1億桁までは FFT を用いて乗算を行い，それ以上の桁数の乗算は，1億桁進数同士の乗算を Karatsuba のアルゴリズムを用いて乗算を行うのである．この場合は，計算桁数が大きくなると実質的な計算量は $O(n^{\log_2 3})$ に近づいていくことになる．

5. 多倍長桁数の平方根および4乗根，逆数計算アルゴリズム

5.1 多倍長桁数の平方根

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，

$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} \tag{6}$$

となる．ここで，式(6)を少し変形した，

☆ 式を変形することにより，計算量がさらに減ることが指摘されている[10]．

☆☆ ニュートン法とは異なることに注意．

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right) \tag{7}$$

の形が，教科書で紹介されることが多い．

式(6),(7)にはx_k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，

$$\begin{aligned} x_{k+1} &= x_k - \frac{1/x_k^2 - a}{-2x_k^{-3}} \\ &= x_k + \frac{x_k(1 - ax_k^2)}{2} \end{aligned} \tag{8}$$

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),14)]．

5.2 多倍長桁数の4乗根

aの4乗根$a^{1/4}$も，平方根と同様にニュートン法を用いて計算する．まず$a^{-1/4}$を$f(x) = 1/x^4 - a = 0$の解として，

$$x_{k+1} = x_k + \frac{x_k(1 - ax_k^4)}{4} \tag{9}$$

で求めた後に3乗し，それにaを掛けて$a^{1/4}$を求める．

5.3 多倍長桁数の逆数

aの逆数a^{-1}も，平方根や4乗根と同様にして，$1/a$を$f(x) = 1/x - a = 0$の解として，

$$x_{k+1} = x_k + x_k(1 - ax_k) \tag{10}$$

で求まる．

6. 多倍長桁数の加減乗算の並列アルゴリズム

6.1 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．

ところが，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．

Fortran 90で記述された，多倍長桁数の並列加算プログラムの核となる部分は，次のようになる．

```
1  MA(1:N)=MA(1:N)+MB(1:N)
2  DO WHILE (ANY(MA(2:N) .GE. BASE))
3    MC(N)=0
4    MC(1:N-1)=INT(MA(2:N)/BASE)
5    MA(2:N)=MA(2:N)+MC(2:N)-MC(1:N-1)*BASE
6    MA(1)=MA(1)+MC(1)
7  END DO
```

基数をBASEとしたとき，入力となるデータは各要素が0~BASE-1に正規化されて配列MAとMBに入っているものとする．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BASE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，キャリーを求めた後に，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，各要素が基数BASE未満になるまでループが繰り返されることになる．

しかし，多倍長桁数の計算において各要素の値はランダムで，正規化されているものと仮定すれば，例えば基数を10^8とした場合にキャリーが2回続けて出現する確率は，$0.5 \times (1/10^8)^2 = 5 \times 10^{-17}$となり，$0.99999999\cdots9 + 0.00000000\cdots1$などの特別な場合を除いては，事実上問題はないことが分かる．

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[15)]を用いるか，桁上げ先見(carry look-ahead)方式を一次回帰演算で実現し，recursive doubling[16)]によって並列化することが考えられる．今回の計算では，実現が容易である桁上げ飛び越し方式を並列化して，キャリーの伝搬を防いでいる．

また，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．

6.2 多倍長乗算ルーチンの並列化

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．

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案されており，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．

7. 並列化の評価

並列化の評価に際しては，円周率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

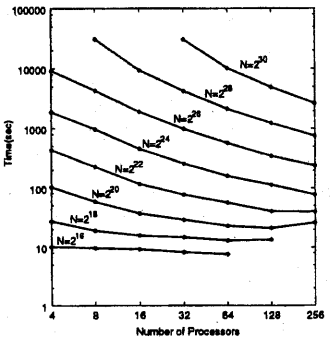

図1 HITACHI SR2201 による円周率の計算時間（ガウス–ルジャンドルの公式，N は計算桁数，MPI を使用）

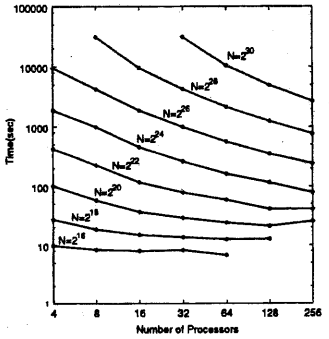

図2 HITACHI SR2201 による円周率の計算時間（ボールウェインの 4 次の収束の公式，N は計算桁数，MPI を使用）

ことにより行った．

計算機としては，分散メモリ型型並列計算機 HITACHI SR2201（1024PE，総主記憶 256GB）のうち，4PE～256PE を用いた．計算時間の測定に際しては，使用 PE 全てをシングルユーザーで使用し，経過時間を測定した．

並列実数 FFT ルーチンとしては，文献 9) による 2,3,5 基底の複素 FFT を元にして，実数 FFT の性質を利用し高速に計算できるように改良したものを用いた．通信ライブラリには，MPI[17]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 V02-05B を用い，最適化オプションとして-W0,'opt(o(ss),approx(0))' 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 8 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，10 進 8 桁ごとに格納されている 32 ビット整数の配列のデータを，10 進 4 桁ごとに 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 n は，$n = 2^{16} \sim 2^{30}$桁まで変化させ，プロセッサ数 P を $P = 4 \sim 256$ と変化させて測定した．

図 1，図 2 から分かるように，ガウス–ルジャンドルの公式および，ボールウェインの 4 次の収束の公式のいずれにおいても，$n = 2^{20}$桁の円周率の計算においては 128 台まではプロセッサ数の増加に伴い実行時間が短縮されているが，256 台にな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の FFT の計算において全対全通信を行っている[9]ために，プロセッサ数が増加するにしたがって一度に送るデータ量が少なくなるため，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．

しかし，今回作成したプログラムでは，全対全通信において，データが小さい時にはプロセッサ間通信のレイテンシーの影響の小さい shuffle-exchange アルゴリズムを使うようにし，データが大きい時には総データ転送量の少ない pairwise アルゴリズムを使うように切り替えているので，プロセッサ間通信のオーバーヘッドが極力抑えられている．したがって，データが小さく，プロセッサ台数が多い場合（例えば $n = 2^{16}$ 桁で，プロセッサ数が 64 台の場合）であってもそれほど並列化効率は低下していないことが分かる．

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．

7.1 円周率 515 億 3960 万桁計算の詳細

並列化の評価の結果，スケーラビリティや主記憶容量を考慮すると，HITACHI SR2201（1024PE，総主記憶 256GB）においては 40 時間以内で円周率 500 億桁余りの計算が可能であるという見込みが立ったので，円周率 515 億 3960 万桁計算を行った．

今回作成したプログラムのプロセッサ間通信ライブラリとしては，MPI を用いた版と，より高速なリモート DMA 転送[18]を用いた版の 2 種類を作成した．今回は主計算には MPI 版を用い，検証計算にはリモート DMA 転送版を用いている．

今回計算に用いた並列計算機 HITACHI SR2201 の要素プロセッサは 32bit プロセッサであるため，このままでは 32bit を超える多倍長桁数は計算できない．したがって、各 PE 内のローカルアドレスを 32bit とし，全 PE にまたがるグローバルアドレスを 52bit（IEEE 方式の倍精度浮動小数点の仮数部）として，プログラムを作成した．

今回作成したプログラムは，許される記憶容量に応じて FFT で使用する実メモリ量を変更することが可能なようになっている．したがって，MPI 版ではこのメモリパラメーターを変更するだけで，FORTRAN77 と MPI が動作する並列計算機ならば，このプログラムの実行が可能である．

この円周率 515 億 3960 万桁計算を，計算時間，使

用記憶容量の面からまとめると，次のようになる．

主計算について:

計算開始	: 1997 年 6 月 6 日 22 時 29 分
計算終了	: 1997 年 6 月 8 日 3 時 32 分
経過時間	: 29 時間 3 分 11 秒
主記憶容量	: 212 GB
アルゴリズム	: ボールウェインの 4 次の収束のアルゴリズム

検証計算について:

計算開始	: 1997 年 7 月 4 日 22 時 11 分
計算終了	: 1997 年 7 月 6 日 11 時 19 分
経過時間	: 37 時間 8 分 16 秒
主記憶容量	: 188 GB
アルゴリズム	: ガウス-ルジャンドルアルゴリズム

円周率の小数点以下 51,539,607,451 桁から 51,539,607,500 桁までは，

89257 77153 58156 10377 74346
45737 03050 02638 01698 31214

となった．

ボールウェインの 4 次の収束の公式による 18 回の反復と，ガウス-ルジャンドルの公式による 35 回の反復による $3 \times 2^{34} = 51,539,607,552$ 桁の円周率の計算結果は，最後の 42 桁（丸め誤差）を除き全て一致していた☆．

8. ま と め

本稿では，分散メモリ型並列計算機により高精度の円周率を高速に計算する方法について述べた．多倍長桁数の円周率は，ガウス-ルジャンドルの公式およびボールウェインの 4 次の収束の公式を用いると，効率良く計算できることが知られている．これらの公式には平方根や 4 乗根，そして逆数計算が含まれているが，ニュートン法を適用することにより，加減乗算に帰着することができる．

また，n 桁の多倍長桁数同士の乗算は 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 の計算および最終結果の正規化の部分を並列化した．

その結果，1024 プロセッサで主記憶容量 256 GB の分散メモリ型並列計算機 HITACHI SR2201 を用いて，515 億桁余りの円周率の計算が 40 時間以内で終了することが分かった．

参 考 文 献

1) Brent, R. P.: Fast Multiple-Precision Evaluation of Elementary functions, *J. ACM*, Vol. 23, pp. 242–251 (1976).
2) Salamin, E.: Computation of π Using Arithmetic-Geometric Mean, *Math. Comp.*, Vol. 30, pp. 565–570 (1976).
3) Knuth, D. E.: *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Mass. (1981).
4) 柴田昭彦: 円周率 1000 万桁への歩み, 数理科学, No. 225, pp. 65–73 (1988).
5) Schönhage, A. and Strassen, V.: Schnelle Multiplikation grosser Zahlen, *Computing (Arch. Elektron. Rechnen)*, Vol. 7, pp. 281–292 (1971).
6) 高橋大介, 金田康正: 円周率—高速計算法と統計性—(3), 情報処理学会第 37 回プログラミングシンポジウム報告集, pp. 73–84 (1996).
7) Blatner, D.: online document about π (1997). online at `http://www.joyofpi.com/`.
8) Borwein, J. M. and Borwein, P. B.: The Arithmetic-Geometric Mean and Fast Computation of Elementary Functions, *SIAM Rev.*, Vol. 26, pp. 351–365 (1984).
9) 高橋大介, 金田康正: 分散メモリ型並列計算機による 2, 3, 5 基底一次元 FFT の実現と評価, 並列処理シンポジウム JSPP'97, pp. 369–376 (1997).
10) 高橋大介, 金田康正: 多数桁の円周率を計算するための公式の改良：ガウス-ルジャンドルの公式とボールウェインの 4 次の収束の公式, 情報処理学会論文誌 (投稿中).
11) Bailey, D. H.: The Computation of π to 29,360,000 Decimal Digits Using Borwein's Quartically Convergent Algorithm, *Math. Comp.*, Vol. 50, pp. 283–296 (1988).
12) Slowinski, D.: private communication (1990).
13) 高橋大介, 金田康正: 並列計算機における二次記憶を用いた一次元 FFT の 実現と評価, 情報処理学会研究報告 97-ARC-123, pp. 7–12 (1996).
14) 伊理正夫: ニュートン法の実際, 数理科学, No. 218, pp. 10–16 (1981).
15) 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).
16) Stone, H.: An Efficient Parallel Algorithm for the Solution of Tradiagonal System of Equcations, *JASM*, Vol. 20, No. 1, pp. 27–38 (1973).
17) Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard, Version 1.1* (1995).
18) 日立製作所: HI-UX/MPP リモート DMA 転送使用の手引 6A20-3-021 (1996).

☆ ただし一致している全ての桁数が正しいという保証は無く，たまたま最後の 1 桁が合ってしまったということが生じ得ることに注意しておく．