多数桁計算における高速アルゴリズムの研究

Studies on Fast Algorithms for High-precision Computation

２００５年３月

早稲田大学大学院理工学研究科

後　保範

Yasunori Ushiro

概 要

本研究の目的は，多数桁で構成される計算システムの基となる高速アルゴリズムを確立し，その有効性を評価することである．高速アルゴリズムとして，高速フーリエ変換 (FFT) に剰余理論を取り込んだ高速剰余変換 (FMT)，級数に基づく多数桁計算の計算量を削減する分割有理数化法 (DRM 法) 及び多倍長精度の高速行列乗算法を考案した．また，これらアルゴリズムの適用例として，2002 年 11 月に達成した 10 進 1 兆 2411 億桁の円周率計算の世界記録，及び高速剰余変換による多数桁分割乗算を行った．FMT は FFT と結果的に同じ計算式となるが，多数桁乗算を目的に考案したもので，途中変換の意味が明確で多数桁乗算への各種応用に対する見通しが FFT より優れている．FMT の多数桁への応用として複素 FMT の直接利用，自然数 α に対して $\pm\alpha$ で巡回する乗算，2 段階 FMT による乗算及び分割乗算を考案した．DRM 法は，n 桁乗算の計算量を $O(M(n))$ とした場合に，入力の桁数が $O(1)$ の有理数級数のとき n 桁の級数値の計算量を $O(n^2)$ から $O(M(n)\cdot(\log n)^2)$ に削減し，入力の桁数が n 桁の実数のとき $O(M(n)\cdot n)$ から $O(M(n)\cdot(\log n)^3)$ に削減する方法である．また，DRM 法は連分数や基底変換にも適用可能で適用範囲が広い．多倍長精度の高速行列乗算は，中国剰余定理及び FFT の線形性を利用し，n 次元の行列乗算において，中国剰余定理及び FFT の適用回数をそれぞれ $O(n^3)$ から $O(n^2)$ に削減する方法である．高速剰余変換による多数桁分割乗算は，FMT の分割乗算の適用により FMT の要素数を m 分割して計算しても分割しない時と同じ計算量にする方法と，ダイレクト I/O により必要なデータを非連続に入出力して，総 I/O 量を分割数に依存させなくする方法で構成される．数値実験結果としてもそれを実証した．10 進 1 兆 2411 億桁の円周率世界記録達成は，同一メモリ量 (1T バイト) の計算機を使用して，前回の記録約 2061 億桁を 6 倍更新した．そのため，π の計算公式を算術幾何平均法 (AGM 法，ガウス・ルジャンドル公式) から Arctan 公式に変更し，DRM 法と 2 段階 FFT(2 段階 FMT) による多数桁乗算やその他多数の工夫を実施した．これらの研究結果は，多数桁計算の高速化アルゴリズムとして十分成果が出たと考える．

謝辞

まず最初に，指導教官を快く引き受けて頂き，本論文の内容及び提出に対する様々なご指導を頂いた，早稲田大学の大石進一教授に心から感謝の意を表します．本論文を提出できたのは一重に大石教授のご指導の賜物です．東京大学の金田教授には，円周率計算プロジェクトリーダとして，DRM 法を適用して円周率 1 兆 2411 億桁の世界記録挑戦への機会を与え頂き，更に DRM 法のより広い適用法の検討へのアドバイス及び論文内容に対する貴重なご意見を頂き，謹んで感謝の意を表します．また，円周率計算の世界記録達成のため，プログラムの並列化及び高速化及び本番計算において円周率プロジェクトに参加していただいた，東京大学情報基盤センターの黒田久泰助手，工藤誠氏 (現，(株) 日立製作所) 及び（株）日立製作所ソフトウエア事業部の五百木信洋氏，田中慎一氏，河村宏樹氏，藤田不二男氏及び情報公共事業部の篠原元氏，長谷部裕樹氏に感謝の意を表します．更に，世界記録達成のためのプロジェクトを組織的に支えて頂いた，(株）日立製作所エンタープライズサーバ事業部殿，ソフトウエア事業部殿，公共システム事業部殿，RAID システム事業部殿及び（株）日立電子サービス殿に感謝の意を表します．

目 次

表 目 次

図 目 次

第1章　はじめに

1.1　目的

本研究の目的は，多数桁で構成される計算システムの基となる高速計算アルゴリズムを確立し，その有効性を評価することである．現在の計算機は，整数計算は32ビット整数及び64ビット整数で，実数計算は4バイトの単精度浮動小数点，8バイトの倍精度浮動小数点がハードウエアに実装されている．また，大型計算機には16バイトの4倍精度浮動小数点がソフトウエアでサポートされ，FORTRANやC言語で使用できる．しかし，64ビット整数や4倍精度浮動小数点より長い桁数での計算が必要なものがある．簡単な例として，正方行列Aの要素a_{ij}が，$a_{ij} = 1/(i+j-1)$で表現されるヒルベルト行列を係数とする連立一次方程式では，4倍精度計算でもわずか数十次元で，精度不足のため数値解が計算できない．また，数値を実数の近似値としてではなく，代数的に正確に表現する数式処理システムでは，64ビット整数の値を超える多数桁計算が必要である．一般に，α桁の有理数を係数とするn次元の連立一次方程式の解を正確に計算するには，$O(\alpha \cdot n^2)$桁の有理数計算が必要である．そのため，MATLABに代表される解析ソフトウエアに多倍長精度計算パッケージを含んでいる．MATLABでは桁数の長い乗算は，FFT（高速フーリエ変換）を使用して高速化するような機能も装備されている．また，多数桁計算の常識が最近変化してきている．従来，n桁のπの計算は，Arctanの級数展開公式を使用すると$O(n^2)$の計算量が必要とされていたが，n桁の乗算の計算量を$M(n)$とすると分割有理数化法（DRM法）を使用することで，$O(M(n) \cdot (\log n)^2)$で計算可能になった．これは，1兆桁のπ計算で評価すると，これまで約20年間πの世界記録更新に利用されてきた，算術幾何平均法（AGM法，ガウス・ルジャンドル法）と同等の計算量になる．また，n桁の多数桁の乗算はFFTを使用することにより，$O(n \cdot \log n \cdot \log\log n)$の計算量で計算できるが，FFTを$m$個に分割すると$m$倍に増加すると言われていたのが，分割しない場合と同じ計算量にできることが分かった．そのため，多数桁計算で必要な主要高速計算アルゴリズムを網羅的に研究する必要がある．この観点から，本研究では多数桁計算において主要なものを選択し，下記の計算アルゴリズムを研究した．

(1) 高速剰余変換（Fast Modulo Transformation，FMT）

(2) 分割有理数化法（Divide and Rationalize Method，DRM法）

(3) 多倍長精度の係数を持つ行列乗算

また，(1) 及び (2) のアルゴリズムの適用評価として下記の数値実験を行った．

(4) 多数桁の分割乗算

(5) 円周率の 1.2 兆桁計算

これらの研究結果により，多数桁計算における高速化アルゴリズムとして十分成果が出たと考え，学位論文として纏めた．

1.2 概要

多数桁計算システムで最も重要かつ基本となる計算アルゴリズムは，多数桁の乗算に関するものである．その理由は，n 桁の加減算の計算量は $O(n)$ であるのに対して，筆算方式（筆算と同じ計算方式のこと）による乗算の計算量は $O(n^2)$[24] になり，除算や平方根は乗算の定数倍 [24] で計算できるためである．多数桁計算の計算量の削減アルゴリズムは種々考案されており，桁数 n が大きい場合は高速フーリエ変換（FFT）[29] が一番高速で，乗算の計算量は $O(n \cdot \log n \cdot (\log \log n))$ となる．しかし，FFT は本来，信号処理や偏微分方程式を数値的に解くために考案されたものであり，多数桁乗算が目的ではない．そのため，FFT を使用した多数桁乗算は計算過程が直感的に把握しにくく，多数桁乗算の応用への見通しが悪いと見ることもできる．また，FFT だけを使用した多数桁乗算は，途中変換でメモリを多く使用する欠点がある．使用メモリ量を削減する方法として，Karatsuba 法 [30] と合わせて使用する方法があるが，メモリ量を削減するため Karatsuba 法の適用回数を増やすと，計算量が増加するジレンマがある．そこで，FFT に剰余理論を取り込んだ高速剰余変換（Fast Modulo Transformation, FMT）[1] を定義し，多数桁乗算への応用として複素 FMT の直接利用，巡回乗算，2 段階 FMT 及び分割乗算を考案した．複素 FMT の直接利用では，FFT による多数桁乗算に実 FFT[24] を利用していたのを，半分の要素の複素 FMT で計算可能にした．実 FFT は，入力が実数の複素 FFT の結果が共役複素数になるのを利用した計算で，複素 FFT 結果から変換すると最初と最後の要素を特別扱いする必要があり，分散メモリでの並列化等が複雑で性能劣化の元になっていた．複素 FMT が直接使用できれば並列化が容易になる利点がある．また，多数桁乗算の FMT による表現が理論的に容易になる．巡回乗算は FFT による正巡回と負巡回乗算 [37] が知られていたが，FMT では ± 1 だけでなく自然数 α に対して $\pm\alpha$ で巡回する多数桁乗算も可能である．2 段階 FMT による乗算とは，1 要素に多数桁を持つ $2N$ 要素の乗算を $P = \omega^N + 1$ を法とする整数 FMT で行い，そこで発生する各要素ごとの負巡回乗算にも FMT を使用する方法である．P を法とする整数 FMT を上位 FMT，各要素ごとの乗算に使用する FMT を下位 FMT と呼ぶ．基底 ω は 2 以上の任意の整数を選ぶことができる．多数桁の乗

算において，FFT と FMT は導出法は異なるが計算式は同一で，FFT に整数上の変換も含めればFMT での理論はFFT にそのまま適用できる．

多数桁計算システムで次に重要な計算アルゴリズムの一つは，多数桁関数の計算である．三角関数や指数関数等の数学関数は Taylor 展開で無限級数に展開できる．多数桁関数で表される π（円周率）や e（自然対数の底）等の数学定数は無限級数で入力値が $O(1)$ 桁の有理数のものである．一方，多数桁関数の計算をシステム的に閉じるには，入力値が結果と同一精度の桁数の必要がある．そこで，入力値の桁数が $O(1)$ 桁と短い有理数の場合に有効な計算法と，入力値が結果と同一の桁数の場合に有効な計算法を考案した．第 1 の方法は，有理級数の和の計算にトーナメント方式を適用し 2 項づつ通分して有理数化し，多数桁除算で目的の桁数の実数にする方法である．第 2 の方法は多数桁精度の入力値 X に分母の桁数が β，2β，4β，$\cdots$，$2^p\beta$ 桁づつの有理数に分解し，各分割ごとに関数値を計算し，それらから加法定理を使用して X での関数値を計算する方法である．二つの計算方法を合わせて分割有理数化法（Divide and Rationalize Method，DRM 法）[7] と名付ける．無限級数で展開される関数の通常の計算方法では，精度的に必要な項数で打ち切りそれらの和を計算する．そのため，n 桁乗算の計算量を $M(n)$ とするとき，入力の桁数が $O(1)$ 桁の有理数の場合は n 桁精度の関数値計算に $O(n^2)$ の計算量が，入力値が n 桁精度の場合には $O(M(n)\cdot n)$ の計算量が必要である．これに対して，n 桁精度の関数値の計算に DRM 法を適用すると，入力値が $O(1)$ 桁の有理数の場合は計算量を $O(M(n)\cdot(\log n)^2)$ に，入力値が n 桁精度で加法定理が適用できる場合は，計算量を $O(M(n)\cdot(\log n)^3)$ に削減できる．本方法は関数値の計算で有名な Brent のアルゴリズム [25] より適用範囲が広く，連分数の計算や基底変換にも適用可能という広い適用性を持ち，これまで知られているアルゴリズムより単純で分かり易いという特長を持つ．

関数計算と同様に多数桁計算で重要なアルゴリズムとして，多倍長精度（桁数が比較的短い多数桁）の値を要素に持つ行列計算がある．現在の計算機は，10 進 16 桁程度の倍精度浮動小数点演算はハードウエアで高速に行える．また，4 倍精度演算も FORTRAN や C 言語で利用可能なものが多いが，ソフトウエア実行であるため計算速度は倍精度演算に比較して 1 桁程度遅くなる．ここでは，4 倍精度以上の多数桁を係数とする連立一次方程式の解の計算における高速化を目的に，その基本となる行列乗算の高速化アルゴリズムを考案した．多数桁の乗算には入力値を一定の桁数に分割し筆算方式で計算する方法以外に，中国剰余定理（百五減算）[40] 及び FFT を使用する方式がある．しかし，多数桁乗算でこれらの方法が筆算方式より高速になるのは数百桁と計算桁数が多い時である．一方，多数桁の値を係数とする n 次元の行列乗算にこれらの方式を適用する場合は，それらの線形性が利用可能である．この点に着目すると，n 次元の行

列乗算に利用する，中国剰余定理及び FFT の適用回数を $O(n^3)$ から $O(n^2)$ に削減 [5] して，多数桁の乗算が可能になる．この原理を利用して，多倍長精度の値を係数にもつ n 次元の行列乗算に，$O(n^2)$ 回の中国剰余定理又は FFT を適用する高速計算アルゴリズムを考案した．桁数が比較的短いときに中国剰余定理が有利で，長くなると FFT が有利となる．行列乗算において，中国剰余定理は 4 倍精度から筆算方式より高速になる．本計算では計算アルゴリズムと数値実験結果を共に示す．

次に，本研究で考案した高速アルゴリズムが，理論通りの高速性を示すか評価するため，それらの適用に関する数値実験を行った．FMT の適用例として，Disk ファイル I/O を利用した多数桁の分割乗算方式を示す．本分割乗算は，拡張 FMT の適用により FMT を m 分割しても分割しない時と同じ計算量にする方法と，ダイレクト I/O により必要なデータを非連続に入出力して，総 I/O 量を分割数に依存させなくする方法で構成される．分割乗算は，複素 FMT を使用する方法と 2 段階 FMT を使用する方法の，それぞれの具体的アルゴリズムと数値実験結果を示す．ファイル I/O を使用した多数桁の分割乗算は，使用計算機のメモリ容量を越えた桁数の乗算をする場合に必要となる．FFT を使用した多数桁乗算を m 分割すると，計算量は分割しないときの m 倍 [24] になると言われている．また，ファイル I/O を利用し FFT による多数桁乗算を m 分割すると，ファイル I/O の総量も分割しないときの m 倍になると言われている．これに対して，FMT による m 分割乗算のアルゴリズム [3] を使用すると，n 桁計算の計算量は $O(n \cdot \log n \cdot (\log \log n))$ でファイル I/O の総量は $O(n)$ となり，共に分割しない時と同じで，n 桁計算に必要なメモリ量は分割しない時の約 $1/m$ に減少する．16 進 1 兆桁の分割乗算で複素 FMT と 2 段階 FMT を比較すると，計算量とファイル I/O 量のオーダーは同一であるが，2 段階 FMT の方がメモリ及びファイル I/O の使用量をより少なくできる．本分割乗算により，1GB のメモリの計算機でも Disk を使用することにより，数百億桁 (数十 G 桁) の多数桁乗算が可能となる．

また，DRM 法の適用例として，2002 年 11 月 24 日に達成した 10 進表現で 1 兆 2411 億桁，16 進表現で 1 兆 307 億桁の円周率の世界記録達成について記述する．本計算は，DRM 法の有効性を実際の大規模な計算で確認するために実施したもので，1 プロセッサーによる π 計算の基本プログラムは著者が一人で作成した．本記録は，同一メモリ容量（1T バイト）の計算機を使用して，前回の記録約 2061 億桁 [22] を 6 倍更新した．そのため，π の計算公式を算術幾何平均法（AGM 法，ガウス・ルジャンドル公式）から Arctan 公式に変更し，DRM 法と 2 段階 FFT（2 段階 FMT）による多数桁乗算を適用し，約 8 万行の計算プログラムで実施した．この計算では，DRM 法の基底変換の適用性を評価する目的で，従来とは異なり 16 進数の π を計算し，10 進数に変換する方式を採用した．円周率の世界記録計算では，DRM 法と 2 段階 FMT 以外にも誤演

算防止対策等多数の工夫を採用した，これらの工夫点に付いても記述する．2001 年には，Ramanujan 型公式を使用して，世界記録達成に挑戦したが，級数の各項の初期値計算に精度不足が発生する，プログラムミスを入れてしまった．そのため，16 進 3597 億余桁で正計算と検証計算の不一致が発生し，その年の記録達成を断念した．このときの，精度不足を発生させたプログラムミスに付いても記述する．

第 2 章の「高速剰余変換による多数桁乗算」[1] は 2003 年 12 月に情報処理学会論文誌 Vol.44 No.12 に掲載されたものである．第 3 章の「分割有理数化法による級数の多数桁計算」は 2000 年 6 月に情報処理学会論文誌 Vol.41 No.6 に掲載された「級数に基づく多数桁計算の演算量を削減する分割有理数化法」[7] である．第 4 章の「多倍長精度の高速行列乗算」は 2001 年 4 月に京大数理研講究録 1198 [偏微分方程式の数値解法とその周辺 II] に「多倍長精度の値を係数とする行列の高速乗算方式」[5] と題して掲載されたものをベースに作成した．第 5 章の「多数桁の分割乗算」は第 2 章の高速剰余変換で考案した多数桁分割乗算の，Disk を使用した具体的計算方法及び数値実験結果である．第 6 章は 2002 年 11 月 24 日に達成した円周率世界記録の計算における，アルゴリズムの工夫点及び計算概要を記載した．本計算で DRM 法が多数桁級数計算及び基底変換へ適用性が高いことを示した．

第2章　高速剰余変換による多数桁乗算

2.1　はじめに

多数桁乗算の計算法は筆算による方法，中国剰余定理，Karatsuba 法，高速フーリエ変換 (FFT) 及びこれらを組み合わせた方法が知られている．計算桁数が多い場合はこれらの中で FFT が最も計算量が少ない．しかし，FFT を使用した多数桁乗算は FFT 計算で多くのメモリを使用し，分割して使用すると演算量が増加する．このため，大きい桁数の乗算には桁数を適当に分割し，分割した部分から元の桁数の乗算に復元するのに Karatsuba 法 [30] を使用することが多かった．そこで，多数桁乗算で FFT の高速性を生かしながらメモリの使用量を減少させる方法を検討した．その結果，FFT に剰余理論を取り込んだ高速剰余変換 (Fast Modulo Transformation, FMT)[1, 3] が有効なことが判明した．FMT は複素数の他に整数でも利用できる．複素数の上で定義すると FMT と FFT は同じ計算式となる．FMT は剰余理論に基づいて作成されているため，多数桁乗算，即ち畳込み演算に利用すれば中間結果の定義が明白で FFT より見通しが良い．この利点から多数桁乗算への FMT の応用として整数 FMT[12]，巡回乗算，2 段階 FMT，複素 FMT の直接利用及び分割乗算が導出できた．複素 FMT の直接利用による多数桁乗算では，通常の乗算の他に半分の要素数で負巡回乗算もできる．FFT でも正巡回乗算と負巡回乗算 [37] は知られているが，整数 FMT による巡回乗算は ± 1 の巡回だけでなく自然数 α に対して $\pm\alpha$ で巡回する乗算が可能と判明した．また，実用的な巡回 FMT の係数も求めることができた．さらに，多数桁乗算の入力値を m 個に分割し，互いに異なる m 個の自然数 α を用いて，$\pm\alpha$ で巡回する $2m$ 個の分割した乗算から元の多数桁乗算が復元できることも分かった．多数桁乗算に 2 段階 FMT を使用すると，8 バイト整数に 2 進 60 桁 [15] を詰めて 1 兆桁の計算も可能である．更に，$2m$ 個の分割乗算を組み合わせると使用メモリ量を大きく削減できる．FMT と FFT は導出方法は異なるが，多数桁乗算への適用では同じ計算式となるため FFT に整数上での計算を含めると，4 章の FMT の適用方法はそのまま FFT にも適用できる．但し，巡回する多数桁乗算の計算では，FMT が上位要素を巡回させるのに対し，FFT は下位要素を巡回させることに注意を要す．

以下，2.2 節に高速剰余変換 (FMT) の方法，2.3 節に FMT による畳込み演算の方法，即ち多数桁乗算の方法，2.4 節に多数桁乗算への FMT の適用方法を示す．

2.2 高速剰余変換 (FMT)

n 個の素数 $P_1, P_2, \cdots, P_n$ を使用して各剰余から元の値を復元するのが中国剰余定理である．FMT では剰余 (mod) を整数だけでなく記号にも拡張し，記号 E と数 ω, k に対して $f \pmod{E-\omega^k}$ を多項式 f 中の記号 E を ω^k に置き換えることで定義する．更に，ω を FFT の基本原理である 1 の原始 n 乗根とし，素数の代わりに $E-\omega^k$ を利用する．順変換は多項式の剰余を求めることで，逆変換は剰余から元の多項式の係数を求めることで定義する．

2.2.1 基本 FMT 変換

$n-1$ 次の E の多項式 f に対して $E-\omega^k, k=0,1,\cdots,n-1$ の剰余に関する変換である．

順変換

多項式 $f \equiv a_{n-1}E^{n-1}+\cdots+a_1E+a_0$ に対して $f_k \equiv f \pmod{E-\omega^k}$ を求める．各 f_k は f 中の E に ω^k を代入して式 (2.1) が得られる．

$$\begin{aligned} f_k &= a_{n-1}\omega^{(n-1)k}+\cdots+a_1\omega^k+a_0 \\ &= \sum_{l=0}^{n-1} a_l\omega^{lk}, \\ & \qquad k=0,1,\cdots,n-1 \end{aligned} \tag{2.1}$$

逆変換

式 (2.1) の f_k から元の多項式 f の係数 a_j を求める．この計算には下記の ω が原始 n 乗根の条件を利用する．

$$\sum_{k=0}^{n-1}\omega^{(l-j)k} = \begin{cases} n: & l=j \\ 0: & l\neq j \end{cases} \tag{2.2}$$

式 (2.1) 及び式 (2.2) から次式が得られる．

$$\begin{aligned} \sum_{k=0}^{n-1} f_k\omega^{-kj} &= \sum_{k=0}^{n-1}\sum_{l=0}^{n-1} a_l\omega^{kl}\omega^{-kj} \\ &= \sum_{l=0}^{n-1} a_l \sum_{k=0}^{n-1}\omega^{(l-j)k} = na_j \end{aligned} \tag{2.3}$$

従って，係数 a_j を求める逆変換は次式のように計算できる．

$$\begin{aligned} a_j &= (f_{n-1}\omega^{-(n-1)j} + \cdots + f_1\omega^{-j} + f_0)/n \\ &= \sum_{k=0}^{n-1} f_k\omega^{-kj}/n, \\ & \qquad j = 0, 1, \cdots, n-1 \end{aligned} \tag{2.4}$$

2.2.2 拡張FMT変換

E の $n-1$ 次の多項式 f と整数 $m, s(s = n/m)$ 及び有理数 q に対して，ω を原始 s 乗根とするとき，$s-1$ 次の多項式 $f^{(q,s)} \equiv f \pmod{E^s - \omega^{qs}}$ を定義する．拡張FMT変換はこの $s-1$ 次の多項式 $f^{(q,s)}$ に対する $E - \omega^{k+q}, k = 0, 1, \cdots, s-1$ の剰余変換である．

順変換

多項式 $f^{(q,s)} \equiv f \pmod{E^s - \omega^{qs}} \equiv a_{s-1}^{(s)}E^{s-1} + \cdots + a_1^{(s)}E + a_0^{(s)}$ に対して $f_k^{(q,s)} \equiv f^{(q,s)} \pmod{E - \omega^{k+q}}$ を求める．各 $f_k^{(q,s)}$ は $f^{(q,s)}$ 中の E に数 ω^{k+q} を代入して式 (2.5) が得られる．

$$\begin{aligned} f_k^{(q,s)} &= a_{s-1}^{(s)}\omega^{(s-1)(k+q)} + \cdots + a_1^{(s)}\omega^{k+q} + a_0^{(s)} \\ &= \sum_{l=0}^{s-1} a_l^{(s)}\omega^{(k+q)l}, \\ & \qquad k = 0, 1, \cdots, s-1 \end{aligned} \tag{2.5}$$

ただし，$a_l^{(s)} = a_l + a_{l+s}\omega^{qs} + \cdots + a_{l+(m-1)s}\omega^{(m-1)qs}$ である．

逆変換

$f_k^{(q,s)}$ から $s-1$ 次の多項式 $f^{(q,s)}$ の係数 $a_j^{(s)}$ を求める．式 (2.5) 及び式 (2.2) から次式が得られる．

$$\begin{aligned} \sum_{k=0}^{s-1} f_k^{(q,s)}\omega^{-kj} &= \sum_{k=0}^{s-1}\sum_{l=0}^{s-1} a_l^{(s)}\omega^{(k+q)l}\omega^{-kj} \\ &= \sum_{l=0}^{s-1} a_l^{(s)}\omega^{ql}\sum_{k=0}^{s-1}\omega^{(l-j)k} \\ &= s\omega^{qj}a_j^{(s)} \end{aligned} \tag{2.6}$$

従って，係数 $a_j^{(s)}$ を求める逆変換は次式のように計算できる．

$$\begin{aligned} a_j^{(s)} &= (f_{s-1}^{(q,s)}\omega^{-(s-1)j} + \cdots + f_1^{(q,s)}\omega^{-j} + f_0^{(q,s)})\omega^{-qj}/s \\ &= \omega^{-qj}\sum_{k=0}^{s-1} f_k^{(q,s)}\omega^{-kj}/s, \\ & \qquad j = 0, 1, \cdots, s-1 \end{aligned} \tag{2.7}$$

2.3 FMT による畳込み演算

多数桁乗算は適当な桁数で分割し，それぞれ分割した値を多項式の係数とする畳込み演算に帰着できる．多数桁乗算では畳込み演算結果に分割した桁数単位に桁上げする処理を追加すればよい．このため，多数桁乗算と畳込み演算は同じものとして記述する．

2.3.1 畳込み演算への FMT の適用

次式の f, g で示す E の $n-1$ 次多項式の係数の畳込み演算で作成される E の $2n-1$ 次の多項式を h とする．

$$\begin{aligned} f &\equiv a_{n-1}E^{n-1} + \cdots + a_1E + a_0 \\ g &\equiv b_{n-1}E^{n-1} + \cdots + b_1E + b_0 \\ h &\equiv f \cdot g \equiv c_{2n-1}E^{2n-1} + \cdots + c_1E + c_0 \end{aligned} \tag{2.8}$$

このとき，h の係数 c_j は f, g の係数 a_i, b_{j-i} で次式のように表わされる．

$$\begin{aligned} c_j &= \sum_{i=max(0,j-n+1)}^{min(j,n-1)} a_i b_{j-i}, \\ & \qquad j = 0, 1, \cdots, 2n-1 \end{aligned} \tag{2.9}$$

適用方法

原始 $2n$ 乗根 ω を用いて次式のように計算する．
ここで，$k = 0, 1, \cdots, 2n-1$，$j = 0, 1, \cdots, 2n-1$ である．
(1) FMT 順変換で f_k, g_k を計算する．

$$f_k \equiv f \pmod{E - \omega^k}$$

$$
\begin{aligned}
&= a_{n-1}\omega^{(n-1)k} + \cdots + a_1\omega^k + a_0 \\
g_k &\equiv g \pmod{E-\omega^k} \\
&= b_{n-1}\omega^{(n-1)k} + \cdots + b_1\omega^k + b_0
\end{aligned} \tag{2.10}
$$

(2) 各 k 毎に f_k と g_k の乗算を次式で計算する．

$$
h_k = f_k \cdot g_k \tag{2.11}
$$

(3) FMT 逆変換で h_k から多項式 h の係数 c_j を求める．

$$
c_j = (h_{2n-1}\omega^{-(2n-1)j} + \cdots + h_1\omega^{-j} + h_0)/2n \tag{2.12}
$$

畳込み演算になることの証明

式 (2.11) に式 (2.10) 及び式 (2.9) を代入し式 (2.2) の n を $2n$ に置換えた ω が原始 $2n$ 乗根の条件から次式が成立する．

$$
\begin{aligned}
\sum_{k=0}^{2n-1} h_k\omega^{-jk} &= \sum_{k=0}^{2n-1} f_k g_k \omega^{-jk} \\
&= \sum_{k=0}^{2n-1} \left(\sum_{i=0}^{n-1} a_i\omega^{ik}\right)\left(\sum_{l=0}^{n-1} b_l\omega^{lk}\right)\omega^{-jk} \\
&= \sum_{i=0}^{n-1} a_i \sum_{l=0}^{n-1} b_l \sum_{k=0}^{2n-1} \omega^{(i+l-j)k} \\
&= 2n \sum_{i=max(0,j-n+1)}^{min(j,n-1)} a_i b_{j-i} \\
&= 2nc_j, \\
&\qquad\qquad j = 0, 1, \cdots, 2n-1
\end{aligned} \tag{2.13}
$$

この式から式 (2.12) が導かれる．

2.3.2 拡張 FMT 変換と畳込み演算の関係式

式 (2.8) の $n-1$ 次の多項式 f, g を E の $2n-1$ 次ではなく整数 $m, s(s = n/m)$ 及び有理数 q に対して，ω を原始 s 乗根として $s-1$ 次に畳込むことを考える．

$$
h^{(q,s)} \equiv h \pmod{E^s - \omega^{sq}}
$$

$$
\begin{aligned}
&\equiv \quad f \cdot g \pmod{E^s - \omega^{sq}} \\
&\equiv \quad c_{s-1}^{(q,s)} E^{s-1} + \cdots + c_1^{(q,s)} E + c_0^{(q,s)}
\end{aligned}
\tag{2.14}
$$

適用方法

原始 s 乗根 ω を用いて次式のように計算する．
ここで，$k = 0, 1, \cdots, s-1$，$j = 0, 1, \cdots, s-1$ である．
(1) $s-1$ 次の多項式 $f^{(q,s)}, g^{(q,s)}$ に変更する．

$$
\begin{aligned}
f^{(q,s)} &\equiv \quad f \pmod{E^s - \omega^{sq}} \\
&= \quad a_{s-1}^{(s)} E^{s-1} + \cdots + a_1^{(s)} E + a_0^{(s)} \\
g^{(q,s)} &\equiv \quad g \pmod{E^s - \omega^{sq}} \\
&= \quad b_{s-1}^{(s)} E^{s-1} + \cdots + b_1^{(s)} E + b_0^{(s)}
\end{aligned}
\tag{2.15}
$$

ここで，$a_l^{(s)} = a_l + a_{l+s}\omega^{qs} + \cdots + a_{l+(m-1)s}\omega^{(m-1)qs}$ で
$b_l^{(s)} = b_l + b_{l+s}\omega^{qs} + \cdots + b_{l+(m-1)s}\omega^{(m-1)qs}$，　$l = 0, 1, \cdots, s-1$ である．
(2) 拡張 FMT 順変換で $f_k^{(q,s)}, g_k^{(q,s)}$ を計算する．

$$
\begin{aligned}
f_k^{(q,s)} &\equiv \quad f^{(q,s)} \pmod{E - \omega^{k+q}} \\
&= \quad a_{s-1}^{(s)} \omega^{(s-1)(k+q)} + \cdots + a_1^{(s)} \omega^{k+q} + a_0^{(s)} \\
g_k^{(q,s)} &\equiv \quad g^{(q,s)} \pmod{E - \omega^{k+q}} \\
&= \quad b_{s-1}^{(s)} \omega^{(s-1)(k+q)} + \cdots + b_1^{(s)} \omega^{k+q} + b_0^{(s)}
\end{aligned}
\tag{2.16}
$$

(3) 各 k 毎に $f_k^{(q,s)}$ と $g_k^{(q,s)}$ の乗算を次式で計算する．

$$
h_k^{(q,s)} = f_k^{(q,s)} \cdot g_k^{(q,s)} \tag{2.17}
$$

(4) 拡張 FMT 逆変換で $h_k^{(q,s)}$ から多項式 $h^{(q,s)}$ の係数 $c_j^{(q,s)}$ を求める．

$$
\begin{aligned}
c_j^{(q,s)} \quad = \quad & \left(h_{s-1}^{(q,s)} \omega^{-(s-1)j} + \cdots + h_1^{(q,s)} \omega^{-j} + h_0^{(q,s)} \right) \\
& \omega^{-qj} / s
\end{aligned}
\tag{2.18}
$$

畳込み演算との関係式

式 (2.12) の c_j と式 (2.18) の $c_j^{(q,s)}$ の関係は，式 (2.8) と式 (2.14) の定義から次式のようになる．

$$\begin{aligned} c_j^{(q,s)} &= c_j + c_{j+s}\omega^{sq} + \cdots + c_{j+(2m-1)s}\omega^{(2m-1)sq}, \\ & \qquad j = 0, 1, \cdots, s-1 \end{aligned} \tag{2.19}$$

畳込み演算になることの証明

ω が 1 の原始 s 乗根の条件及び式 (2.15)，式 (2.16)，式 (2.17) を式 (2.18) に代入して次式が成立する．式 (2.20) で式 (2.2) の右辺が s になるのは，$i+l-j=0$ 及び $i+l-j=s$ の 2 つの場合がある．

$$\begin{aligned} c_j^{(q,s)} &= \sum_{k=0}^{s-1} h_k^{(q,s)}\omega^{-jk}\omega^{-qj}/s \\ &= \sum_{k=0}^{s-1} f_k^{(q,s)} g_k^{(q,s)}\omega^{-j(k+q)}/s \\ &= \sum_{k=0}^{s-1} \left(\sum_{i=0}^{s-1} a_i^{(s)}\omega^{i(k+q)}\right) \left(\sum_{l=0}^{s-1} b_l^{(s)}\omega^{l(k+q)}\right) \\ &\qquad \omega^{-j(k+q)}/s \\ &= \sum_{i=0}^{s-1} a_i^{(s)} \sum_{l=0}^{s-1} b_l^{(s)}\omega^{(i+l-j)q} \sum_{k=0}^{\mathrm{s}-1} \omega^{(i+l-j)k}/s \\ &= \sum_{i=0}^{j} a_i^{(s)} b_{j-i}^{(s)} + \omega^{sq} \sum_{i=j+1}^{s-1} a_i^{(s)} b_{j+s-i}^{(s)} \\ &= \sum_{i=0}^{j} \sum_{k=0}^{m-1} a_i\omega^{sqk} \sum_{l=0}^{m-1} b_{j-i}\omega^{sql} + \omega^{sq} \sum_{i=j+1}^{s-1} \\ &\qquad \sum_{k=0}^{m-1} a_i\omega^{sqk} \sum_{l=0}^{m-1} b_{j+s-i}\omega^{sql} \\ &= c_j + c_{j+s}\omega^{sq} + \cdots + c_{j+(2m-1)s}\omega^{(2m-1)sq}, \\ & \qquad j = 0, 1, \cdots, s-1 \end{aligned} \tag{2.20}$$

従って，式 (2.20) は式 (2.19) に等しくなる．

表 2.1: P が 2^{24} の近くの原始 n 乗根となる整数 n, ω, P の例

項番	$n = \alpha = 2^{17}$		$n = \beta = 3 \cdot 2^{15}$		$n = \gamma = 3^2 \cdot 2^{14}$	
	ω	P	ω	P	ω	P
1	770	$149\alpha + 1$	26	$173\beta + 1$	175	$114\gamma + 1$
2	201	$153\alpha + 1$	866	$187\beta + 1$	434	$132\gamma + 1$
3	70	$155\alpha + 1$	562	$194\beta + 1$	12	$136\gamma + 1$
4	20	$164\alpha + 1$	637	$204\beta + 1$	604	$147\gamma + 1$
5	1498	$165\alpha + 1$	100	$220\beta + 1$	459	$150\gamma + 1$

2.4 多数桁乗算への FMT の適用方法

2.4.1 整数 FMT

整数 FMT の一つは $\omega = 2, P = \omega^{n/2} + 1$ とすると，ω が法 P のもとで 1 の原始 n 乗根となるのを利用する方法である．ω は 2 以上の任意の整数でも可能である．この方法の問題点は要素数 n が大きくなると，P が指数的に大きくなることである．この問題点を解消するには，適当な大きさの素数 P 上で 1 の原始 n 乗根となる ω を利用すれば良い．計算機で利用しやすい原始 n 乗根となる n, ω, P の例を表 2.1 及び表 2.2 に示す．表 2.1 は倍精度浮動小数点を使用して，P 以下の任意の整数の乗算とその P による剰余が正確に計算できるため，簡単にプログラムが作成できる利点がある．しかし，要素数 n を大きくできない欠点がある．表 2.2 は倍精度浮動小数点で，P 以下の任意の整数の乗算とその剰余はできないがその部分だけ工夫すれば，倍精度浮動小数点で FMT の計算が可能である．また，表 2.1 に比べて要素数 n も大きく取れる．更に，表 2.1 は約 4 倍，表 2.2 は約 16 倍大きな P を選んでも IEEE の倍精度浮動小数点を使用して計算可能であるが，FMT のプログラム作成では多少 P を超える値の乗算が可能なようにしておく方がプログラム作成が容易になる．このため表 2.1 と表 2.2 の値を選んだ．これらの表の $1/2, 1/3, \cdots$ 倍の n に適用するには，P はそのままで ω を法 P のもとで $\omega^2, \omega^3, \cdots$ に置換えればよい．表 2.1，表 2.2 の複数個の P による FMT の畳込み演算結果は，各 P が素数のため中国剰余定理で結果を重ね合せ，各要素の桁数を拡大することができる．

表 2.2: P が 2^{48} の近くの原始 n 乗根となる整数 n, ω, P の例

項番	$n=\alpha=2^{35}$		$n=\beta=3\cdot 2^{32}$		$n=\gamma=3^2\cdot 2^{31}$	
	ω	P	ω	P	ω	P
1	360	$8319\alpha+1$	9375	$22100\beta+1$	8426	$14636\gamma+1$
2	933	$8334\alpha+1$	5549	$22226\beta+1$	226	$14771\gamma+1$
3	1714	$8549\alpha+1$	9520	$22254\beta+1$	4987	$14896\gamma+1$
4	474	$8756\alpha+1$	3991	$22314\beta+1$	9607	$14913\gamma+1$
5	744	$8759\alpha+1$	6955	$22316\beta+1$	7806	$14994\gamma+1$

2.4.2 正巡回と負巡回乗算

拡張 FMT 変換による畳込み演算を使用する．式 (2.19) において $s=n, m=1$ で $q=0$ のとき正巡回 FMT，$q=1/2$ のとき負巡回 FMT と呼び，それぞれに対応する乗算を正巡回乗算及び負巡回乗算と呼ぶ．正巡回 FMT と負巡回 FMT を求める目的は多数桁乗算が 2 分割でき FMT での使用メモリ量を半減することである．更に，M,n を整数とするとき，任意の整数乗算結果の M^n-1 や M^n+1 における剰余が直接計算できるため，この形の合成数の素因数分解の計算にも利用できる．また，負巡回 FMT は 2 段階 FMT の下位 FMT として必要な計算手段である．式 (2.19) に $s=n, m=1$ での q に 0 及び 1/2 を代入すると $\omega^0=1$，$\omega^{n/2}=-1$ で次式のようになる．

$$\begin{aligned} c_j^{(0,n)} &= c_j + c_{j+n} \\ c_j^{(1/2,n)} &= c_j - c_{j+n}, \\ & \qquad j=0,1,\cdots,n-1 \end{aligned} \tag{2.21}$$

正巡回乗算の係数 $c_j^{(0,n)}$ と負巡回乗算の係数 $c_j^{(1/2,n)}$ から次式のように元の乗算の $2n-1$ 次の係数 c_j が計算できる．

$$\begin{aligned} c_j &= (c_j^{(0,n)} + c_j^{(1/2,n)})/2 \quad ; \\ & \qquad \text{元の乗算の下位要素の係数} \\ c_{j+n} &= (c_j^{(0,n)} - c_j^{(1/2,n)})/2 \quad ; \\ & \qquad \text{元の乗算の上位要素の係数,} \\ & \qquad j=0,1,\cdots,n-1 \end{aligned} \tag{2.22}$$

2.4.3　2段階FMTによる多数桁乗算

FFTを利用した多数桁乗算の問題点はFFT利用時に大量のメモリを使用することである．これを解消する方法として2段階FMTが有効である．2段階FMTによる乗算とは，1要素に多数桁をもつN要素の乗算を$P = \omega^{N/2} + 1$を法とする整数FMTでおこない，そこで発生する式(2.17)に対応する各要素ごとの乗算にもFMTを使用する方法である．Pを法とする整数FMTを上位FMT，各要素ごとの乗算に使用するFMTを下位FMTと呼ぶ．ωとして2のべき乗を使用すれば，上位FMTは加減算とシフト演算だけとなる特長がある．ここで重要なのは下位FMTの乗算結果がそのまま法Pの剰余となることである．乗算結果が法Pの剰余となるためには，下位FMTに式(2.23)を満たす負巡回FMTを利用すればよい．ここで下位FMTの要素数をn，1要素に入れるα進数をk桁とし，上位FMTの要素数をN，原始N乗根を$\omega = \alpha^l$とする．

$$l \cdot N = 2n \cdot k \tag{2.23}$$

計算方法

2段階FMTによる多数桁乗算の基底は2進数である必要はなく，10進数でも他の基底でも可能であるが，説明を簡単にするため2進数を仮定して説明する．基底を変更するときは，ωの値を基底数のべき乗にする必要がある．

(1) 記号の定義

ここで使用する記号を下記のように定義する．

N：上位FMTの要素数,　　n：下位FMTの要素数

F_P, F_P^{-1}：上位正巡回FMTの順変換及び逆変換

F_N, F_N^{-1}：上位負巡回FMTの順変換及び逆変換

f_N, f_N^{-1}：下位負巡回FMTの順変換及び逆変換

$\bullet$：通常の乗算(畳み込み演算による全要素乗算)

$\otimes$：項別乗算(対応する要素の乗算)

(2) 上位FMTの計算

上位FMTの要素数をNとしたとき，上位FMTの1要素に2進mN桁を記憶できるようにする．このとき，上位FMTは$\omega = 2^{2m}$で$P = \omega^{N/2} + 1$として構成する．共にN個の要素をもつAとBの多数桁の乗算を下記の様に表す．乗算結果は$2N$個の要素になる．

$A \bullet B = (c_{2N-1}, c_{2N-2}, \cdots, c_N, \cdots, c_1, c_0)$

AとBの多数桁の乗算を$2N$個の要素のFMTではなく，N個の要素の正巡回FMTと，負巡回FMTによる乗算を利用して下記の様に計算する．

正巡回乗算：

$$F_P^{-1}(F_P(A) \otimes F_P(B)) = (d_{N-1}, d_{N-2}, \cdots, d_1, d_0)$$

負巡回乗算:

$$F_N^{-1}(F_N(A) \otimes F_N(B)) = (e_{N-1}, e_{N-2}, \cdots, e_1, e_0)$$

A と B の多数桁乗算の $2N$ 個の要素 c_{j+N}, c_j は下記で計算できる．

$c_{j+N} = (d_j - e_j)/2$

$c_j = (d_j + e_j)/2, \qquad j = 0, 1, \cdots, N-1$

正巡回乗算と負巡回乗算で現れる要素単位の項別乗算 ($\otimes$) は下位 FMT を利用しておこなう．

(3) 下位 FMT の計算

上位 FMT の要素単位に実施される項別乗算に下位 FMT は利用される．そのため，下位 FMT を利用する多数桁の乗算は法 P の上で実行される必要がある．$P = \omega^{N/2} + 1 = 2^{mN} + 1$ のため，下位 FMT の要素数を n，下位 FMT の 1 要素に詰める 2 進桁数を k とすると下記関係が必要となる．

$m \cdot N = n \cdot k$

この条件を満たせば，下位 FMT に実数 FMT を使用するか，整数 FMT を使用するかは自由である．要素数 n でこの条件のもとに，多数桁乗算結果が法 P を満たすためには，負巡回 FMT を下位乗算に使用すればよい．即ち，2 進 k 桁を 1 要素とする a と b の nk 桁同士の乗算は下記の様に n 要素の負巡回 FMT を使用することにより，法 P のもとで nk 桁で求まる．

$$a \cdot b \pmod{P} = f_N^{-1}(f_N(a) \otimes f_N(b))$$

使用メモリ量の比較

具体的なメモリ使用量の比較例として，入力 A,B は 0.5 兆桁で出力 C が 1 兆桁 (10 進換算) となる 3 種類の多数桁乗算を大規模並列計算機で計算するとして比較する．TB はテラバイトの略である．

(1) 2 段階 FMT による 1 兆桁乗算

上位 FMT は正巡回 FMT と負巡回 FMT を重ね合わせる．

上位 FMT は整数 8 バイトに 2 進 60 桁を詰めて計算できる．

FMT(A) 及び計算結果	： 0.5TB
FMT(B)	： 0.5TB
正巡回結果の保存	： 0.25TB
下位 FMT 及び通信ワーク	： 0.05TB
合計	： 1.3TB

(2) 実FFTによる1兆桁乗算

乗算入力の各要素の値は絶対値が最小になるように正，負の符号付で表示する．正，負の値で平均化されるため結果Cの各要素で保持する桁数は理論限界より長くできる．それでも，実数8バイトに2進9桁詰めが限度である．正巡回FFTと負巡回FFTの重ね合わせはしない．

FFT(A)及び計算結果	： 3.0TB (2進9桁)
FFT(B)	： 3.0TB
FFTワーク	： 3.0TB（並列化)
合計	： 9.0TB

(3) 実FFTとKaratsuba法による1兆桁乗算

Karatsuba法の適用回数が増加するとメモリ量は減るが演算量は増加する．1回Karatsuba法を使用すると，FFT領域は半減し，Karatsuba法のワークは増加する．

	FFT	ワーク	合計	演算量比
0回適用：	9.0TB	+ 0.0TB	= 9.0TB	1.0 倍
1回適用：	4.5TB	+ 1.0TB	= 5.5TB	1.5 倍
2回適用：	2.3TB	+ 1.5TB	= 3.8TB	2.3 倍
3回適用：	1.2TB	+ 1.7TB	= 2.9TB	3.4 倍
4回適用：	0.6TB	+ 1.8TB	= 2.4TB	5.1 倍

2.4.4 複素FMTの直接使用による多数桁乗算

複素FFTによる多数桁乗算では，複素共役性を利用して実FFTに変換する必要がある．これに対し拡張FMT変換を利用すると複素FMTで直接多数桁乗算が可能となる．複素FMTで直接多数桁乗算が可能なことの利点は，実FMTを経由しないためプログラムが簡単になることである．特に，並列計算機において複素FMT(又はFFT)の計算結果を実FMT(又はFFT)に変換するのは，先頭と中間の要素を特別処理することで，対応する要素位置が1づれるため結構面倒な工夫をする必要がある．それに対して，複素FMTによる直接多数桁計算では，要素位置は正確に要素数の半分か1/4づれるだけのため並列処理も簡単になる．

式(2.19)にωを複素数として$s = n, m = 1$と$q = 1/4$を適用すると，$\omega^{n/4} = i$から次式のようになる．ここでiは虚数単位である，

$$\begin{aligned} c_j^{(1/4,n)} &= c_j + i \cdot c_{j+n}, \\ & \qquad j = 0, 1, \cdots, n-1 \end{aligned} \tag{2.24}$$

式(2.24)は，入力値の虚部をゼロにした複素FMTによる乗算結果$c^{(1/4,n)}$の実部が元の多数桁乗算の下位要素(c_j)に対応し，虚部が上位要素(c_{j+n})に対応することを示し

ている，
更に，$n-1$ の代りに $n/2-1$ 次で畳込むことを考える．式 (2.19) に $s=n/2, m=2$ と $q=1/4$ を代入すると，$\omega^{n/8}=\omega^{s/4}=i$ から次式のようになる．

$$\begin{aligned} c_j^{(1/4,n/2)} &= c_j + c_{j+n/2}\omega^{n/8} + c_{j+n}\omega^{n/4} \\ &\quad + c_{j+3n/2}\omega^{3n/8} \\ &= (c_j - c_{j+n}) + i\cdot(c_{j+n/2} - c_{j+3n/2}), \\ &\quad j=0,1,\cdots,n/2-1 \end{aligned} \tag{2.25}$$

n 要素の整数入力値に $\pmod{E^{n/2}-i}$ を施すと，入力の上位 $n/2$ 要素を虚部に下位 $n/2$ 要素を実部にした $n/2$ 要素の複素数となる．式 (2.25) は，この $n/2$ 要素の複素 FMT の乗算結果 $c_j^{(1/4,n/2)}$ の実部が元の n 要素の負巡回乗算の下位要素 $(c_j - c_{j+n})$ に，虚部が上位要素 $(c_{j+n/2} - c_{j+3n/2})$ に対応していることを示している．従って，複素 FMT で負巡回乗算が可能となる．

2.4.5 ± 自然数での巡回乗算

α を任意の自然数とするとき，整数 FMT では $\pm\alpha$ で巡回する多数桁乗算の計算が可能である．$\alpha=1$ のときは正及び負巡回乗算になる．これは，$s=n, m=1$ 及び q を有理数として，法 P 上での原始 n 乗根 ω を使用する整数 FMT で式 (2.19) が成立するためである．$\pm\alpha$ で巡回する乗算の第 1 の目的は多数桁の分割乗算に利用する事である．それ以外に，M, n を整数とするとき，任意の整数乗算結果の $M^n-\alpha$ や $M^n+\alpha$ における剰余が直接計算できるため，合成数の因数分解の計算にも利用できる．$\pm\alpha$ で巡回する乗算ができるためには，要素数 n に対して法 P と $\alpha \equiv \omega^{nq} \pmod{P}$ となる自然数の組 P, ω, ω^q 及び有理数 q あればよい．この例として，$n=2^{20}$ で 2 から 10 までの各 α に対応する値を表 2.3 に示す．表 2.3 の n が $2^{19}, 2^{18}, \cdots$ に適用するには α, P, q はそのままで ω, ω^q の値をそれぞれ 2 乗,4 乗,$\cdots$ すればよい．表 2.3 の記号を使用すると，整数 FMT での巡回乗算は $\omega^{n/2}=-1$ のため次式のようになる．ここで，$c_j, c_j^{(q,n)}$ は式 (2.9) 及び式 (2.18) で定義したものである．

$$\begin{aligned} c_j^{(q,n)} &= c_j + \alpha\cdot c_{j+n} \\ c_j^{(q+1/2,n)} &= c_j - \alpha\cdot c_{j+n}, \\ &\quad j=0,1,\cdots,n-1 \end{aligned} \tag{2.26}$$

表 2.3: $n = 2^{20}$ で $\pm\alpha$ で巡回する乗算となる整数FMTの係数の例

α	P	q	ω	ω^q
2	$280049478 \cdot n + 1$	1/93349826	243728030273313	196084934801470
3	$322504032 \cdot n + 1$	1/80626008	278086974322725	45280667942168
4	$283546518 \cdot n + 1$	1/94515506	237787923135329	113109652731064
5	$205984108 \cdot n + 1$	1/205984108	43678380610349	192726477692165
6	$282686616 \cdot n + 1$	1/282686616	20159304809371	288630981189601
7	$222921552 \cdot n + 1$	1/111460776	161161411978886	163047718620128
8	$280049478 \cdot n + 1$	1/93349826	243728030273313	70308039835214
9	$322504032 \cdot n + 1$	1/40313004	93762856436902	259915309872599
10	$314771758 \cdot n + 1$	1/314771758	149271167055472	94022866127931

2.4.6 巡回乗算による多数桁の分割乗算

多数桁の分割乗算の目的は計算メモリ量の削減である．更に重要な用途と思われるのは，Disk 等の外部記憶装置を利用してメモリの何十倍もの桁数の乗算が実用的な時間で可能になる事である．分割乗算を利用すると演算量を増加させないで，Disk への入出力回数を大幅に削減した超多数桁の乗算が可能になる．
式 (2.22) で正巡回乗算と負巡回乗算から元の乗算の係数 c_j が計算できることを示した．これを更に進めて，$2m$ 個の異なる巡回乗算から元の乗算の係数 c_j が計算できる．

式 (2.19) で $s = n/m$ 及び $q = k/2m$ を代入すると次式が得られる．但し，ω は原始 s 乗根で ω_n は原始 n 乗根を示し，$\omega_n = \omega^{1/m}$ である．

$$\begin{aligned} c_j^{(k/2m,n/m)} &= c_j + c_{j+n/m}\omega_n^{kn/2m} + \cdots \\ &\quad + c_{j+(2m-1)n/m}\omega_n^{(2m-1)kn/2m}, \\ &\quad j = 0, 1, \cdots, n/m - 1, \\ &\quad k = 0, 1, \cdots, 2m - 1 \end{aligned} \tag{2.27}$$

この式から逆に $2m$ 分割したときの元の $2n-1$ 次の係数は次式のようになる．

$$\begin{aligned} c_{j+nk/m} &= \sum_{l=0}^{2m-1} c_j^{(l/2m,n/m)}\omega_n^{-lkn/2m}/2m, \\ &\quad j = 0, 1, \cdots, n/m - 1, \\ &\quad k = 0, 1, \cdots, 2m - 1 \end{aligned} \tag{2.28}$$

具体的例として $m=2$ で 4 個に分割して計算する例を次式に示す．ここで，ω_n は原始 n 乗根で，$\omega_n^{-n/2}=-1$ である．

$$\begin{aligned}
c_j &= (c_j^{(0,n/2)}+c_j^{(1/4,n/2)}+c_j^{(1/2,n/2)} \\
&\quad +c_j^{(3/4,n/2)})/4 \\
c_{j+n/2} &= (c_j^{(0,n/2)}+c_j^{(1/4,n/2)}\omega_n^{-n/4}-c_j^{(1/2,n/2)} \\
&\quad -c_j^{(3/4,n/2)}\omega_n^{-n/4})/4 \\
c_{j+n} &= (c_j^{(0,n/2)}-c_j^{(1/4,n/2)}+c_j^{(1/2,n/2)} \\
&\quad -c_j^{(3/4,n/2)})/4 \\
c_{j+3n/2} &= (c_j^{(0,n/2)}-c_j^{(1/4,n/2)}\omega_n^{-n/4}-c_j^{(1/2,n/2)} \\
&\quad +c_j^{(3/4,n/2)}\omega_n^{-n/4})/4, \\
&\qquad j=0,1,\cdots,n/2-1
\end{aligned} \tag{2.29}$$

これは，元の $2n$ 個の要素の多数桁乗算結果が 4 分割されて，それぞれ $n/2$ 個の要素の乗算として求められることを示している．

2.5 まとめ

高速剰余変換 (FMT) を利用し，多数桁乗算において FFT と同じ演算量でメモリ使用量を削減する方法を考案した．整数 FFT と正負の巡回乗算は既に FFT の応用として知られているが，負巡回の性質を利用した 2 段階 FMT，複素 FMT の直接利用による多数桁乗算，自然数 α に対して $\pm\alpha$ 巡回乗算及びそれを使用した分割乗算の提案は他に見当たらない．また倍精度浮動小数点で表示できる法 P のもとで $\pm\alpha$ で巡回する FMT の係数を具体的に提示できた．2 段階 FMT 及び FMT による分割乗算により，多数桁乗算のメモリ使用量が FFT に比較して大幅に削減でき，当初の目的が達成できた．FMT は複素数で定義すると FFT と同じ計算式となる．一方，剰余理論をベースにしている FMT は多数桁乗算の中間結果の定義が明白であり，FFT より見通しがよい点が特長である．本論文の多数桁乗算への適用例を示す，FORTRAN と C のプログラムを http://www.dept.edu.waseda.ac.jp/math/ushiro/research/fmtapply.htm に掲載する．今後の課題として，2 段階 FMT と分割乗算を組み合わせてメモリ容量を大幅に超えた桁数の乗算を，外部記憶装置を使用して計算するパッケージの作成と性能評価がある．

第3章　分割有理数化法による級数の多数桁計算

3.1　はじめに

三角関数，指数関数，対数関数および逆三角関数等の数学関数はTaylor展開で無限級数に展開できる．このような無限級数に展開できる関数を多数桁の精度で計算するためには，適当な項数で打ち切り各項ごとに多数桁計算を行いそれらの和を用いる方法が知られている．

この方法の場合，n桁の乗算の計算量を$M(n)$とするとき，入力値の精度が$O(1)$桁の有理数の場合にはn桁精度の関数値計算に$O(n^2)$の計算量が，また入力値がn桁精度の場合には$O(M(n)\cdot n)$の計算量が必要である．これに対し，n桁精度の関数値計算に分割統治法を適用することで，入力値が$O(1)$桁の有理数の場合には計算量を$O(M(n)\cdot(\log_2 n)^2)$以下に，また入力値がn桁精度で，初等関数のように加法定理が適用できる場合には，計算量を$O(M(n)\cdot(\log_2 n)^3)$以下に削減する方式を提案する．

本論文で提案する方式を分割有理数化法（Divide and Rationalize Method，以下DRM法）と名づける．本DRM法は二つの方法から構成される．第一の方法では入力値の精度が桁数$O(1)$の有理数の場合にn桁の乗算の演算量を$M(n)$とするとき，トーナメント有理数化処理[7, 8, 9]で級数に展開される関数のn桁精度関数値計算の計算量を$O(M(n)\cdot(\log_2 n)^2)$以下に削減する．

第二の方法では加法定理が適用できる関数の多数桁計算において，n桁精度の入力値を上位桁から分母の桁数が$\alpha, 2\alpha, 4\alpha, \cdots, 2^{p-1}\alpha$桁ずつの有理数に分割する．各分割した値に対する関数値計算の計算量を$T(n)$とするとき，n桁精度の関数値の計算を$O(T(n)\cdot\log_2 n)$の計算量[7, 8]で実現する．この二つの方法を結合して，多数桁精度の入力で多数桁の関数値計算の計算量を削減するのがDRM法である．

また，本DRM法は連分数で表現される関数値の多数桁精度の計算及び2進10進変換に代表される多数桁の基底変換[7]にも適用可能である．

本DRM法は多数桁精度を必要とする関数値計算に適用でき，各種関数を含めた多数桁計算システムの構築に有用である．またBrent[25]のアルゴリズムより単純で分かり易い上に，適用範囲も広く自然な並列化が可能な点が特長である．

以下 3.2 節で級数に展開される関数値の n 桁精度の計算で，入力値の精度が $O(1)$ 桁精度（計算機の倍精度で表示できる 10 進 10 数桁以内）の有理数の場合の計算方法について述べる．3.3 節で入力値が $O(1)$ 桁精度の有理数の場合に，級数に展開される関数を多数桁精度（n 桁）でトーナメント有理数化処理を適用して計算するときの計算量を評価し，3.4 節で級数で表現され加法定理が適用できる関数の入力値が m 桁 $(m \leq n)$ で，結果の関数値が n 桁精度の場合の計算について述べる．3.5 節で提案した DRM 法の並列化と計算桁数を増加させた場合における結果の再利用性について，3.6 節で計算桁数 n と項数 $q(n)$ の関係を，3.7 節で n 桁計算の計算量を，3.8 節で連分数の有理数化処理を，3.9 節で基底変換を，3.9 節で sin 及び log 関数への適用結果を，3.11 節で関連研究について述べる．3.12 節はまとめである．

3.2 トーナメント有理数化処理

まず一般的に級数で表現される関数の計算方法を説明し，次いで各関数の具体的計算における係数を示す．

3.2.1 級数関数の有理数化処理

x と y は $0 < |y| < |x|$ なる整数で，関数 $f\left(\frac{y}{x}\right)$ が無限級数

$$f\left(\frac{y}{x}\right) = \sum_{l=0}^{\infty} \frac{a_{0,l}}{b_{0,l}} \left(\prod_{k=0}^{l} \frac{c_{0,k}}{d_{0,k}} \right) \left(\frac{y}{x}\right)^{\gamma+\beta l} \tag{3.1}$$

に展開されるとする．ここで，$a_{0,l}$，$b_{0,l}$，$c_{0,k}$，$d_{0,k}$ は整数とする．また，γ は整数で，β は自然数とする．関数 $f\left(\frac{y}{x}\right)$ の値を小数点以下 B 進 n 桁の精度で求めるには，最終項の絶対値とそれ以下の剰余項の和の最大値の比である $\left(1 - \left|\left(\frac{y}{x}\right)^{\beta}\right|\right)$ 対 1 を考慮して

$$\left| \frac{b_{0,q}}{a_{0,q}} \left(\prod_{k=0}^{q} \frac{d_{0,k}}{c_{0,k}} \right) \left(\frac{x}{y}\right)^{\gamma+\beta q} \right| \left(1 - \left|\left(\frac{y}{x}\right)^{\beta}\right|\right) \geq B^{n+1} \tag{3.2}$$

となる項数 q まで理論的には計算すればよい．さらに，実際の計算では計算誤差を取り除くためガード桁を考慮する必要がある．

q 項からなる関数 $f\left(\frac{y}{x}\right)$ を 2 項ずつまとめて表示すると下記のようになる．

$$f\left(\frac{y}{x}\right) = \sum_{l=0}^{\lceil q/2 \rceil} \left(\frac{y}{x}\right)^{\gamma+2\beta l} \left(\prod_{k=0}^{2l} \frac{c_{0,k}}{d_{0,k}} \right) \left(\frac{a_{0,2l}}{b_{0,2l}} + \frac{a_{0,2l+1}}{b_{0,2l+1}} \cdot \frac{c_{0,2l+1}}{d_{0,2l+1}} \left(\frac{y}{x}\right)^{\beta} \right)$$

ここで $q+1$ 項以降は適当に 0 と仮定する．

これを，以下のようにトーナメント方式を適用し 2 項ずつ通分処理で有理数にする．p 段目の有理数化処理は下記のようになる．

$$
\begin{aligned}
f\left(\frac{y}{x}\right) &= \sum_{l=0}^{L_p}\left(\frac{y}{x}\right)^{\gamma+2^p\beta l}\left(\prod_{k=0}^{2l}\frac{c_{p-1,k}}{d_{p-1,k}}\right)\left(\frac{a_{p-1,2l}}{b_{p-1,2l}}+\frac{a_{p-1,2l+1}}{b_{p-1,2l+1}}\cdot\frac{c_{p-1,2l+1}}{d_{p-1,2l+1}}\left(\frac{y}{x}\right)^{2^{p-1}\beta}\right)\\
&= \sum_{l=0}^{L_p}\left(\frac{y}{x}\right)^{\gamma+2^p\beta l}\left(\prod_{k=0}^{2l}\frac{c_{p-1,k}}{d_{p-1,k}}\right)\times\\
&\qquad\left(\frac{a_{p-1,2l}b_{p-1,2l+1}d_{p-1,2l+1}x^{2^{p-1}\beta}+a_{p-1,2l+1}b_{p-1,2l}c_{p-1,2l+1}y^{2^{p-1}\beta}}{b_{p-1,2l}b_{p-1,2l+1}d_{p-1,2l+1}x^{2^{p-1}\beta}}\right)\\
&\equiv \sum_{l=0}^{L_p}\left(\frac{y}{x}\right)^{\gamma+2^p\beta l}\left(\prod_{k=0}^{l}\frac{c_{p,k}}{d_{p,k}}\right)\frac{a_{p,l}}{b_{p,l}}
\end{aligned}
$$

ここで，$p=1,2,\cdots,\lceil\log_2(q)\rceil$ であり $L_p=\lceil q/2^p\rceil$ である．また $c_{p,k}\equiv c_{p-1,2k}c_{p-1,2k+1}$，$d_{p,k}\equiv d_{p-1,2k}d_{p-1,2k+1}$ であり $a_{p,l},b_{p,l}$ は上式に従って定義する．

3.2.2 具体例

三角関数や指数関数などの初等関数と誤差関数などの特殊関数及び円周率計算公式を例に具体的係数を表 3.1 に示す．

ここで関数 $f\left(\frac{y}{x}\right)$ は式 (3.1) の級数で，例えば $\arcsin\left(\frac{y}{x}\right)$ は次のように表される．

$$
\begin{aligned}
\arcsin\left(\frac{y}{x}\right) &= \sum_{l=0}^{\infty}\frac{a_{0,l}}{b_{0,l}}\left(\prod_{k=0}^{l}\frac{c_{0,k}}{d_{0,k}}\right)\left(\frac{y}{x}\right)^{\gamma+\beta l}\\
&= \sum_{l=0}^{\infty}\frac{1}{2l+1}\left(\prod_{k=1}^{l}\frac{2k-1}{2k}\right)\left(\frac{y}{x}\right)^{2l+1}
\end{aligned}
$$

3.3 入力値の精度が $O(1)$ 桁である関数値計算の計算量

まず使用記号を下記に定義する．関数値計算に必要な級数の項数は 2 のべき乗とはならないが，計算量の算出のため 2 のべき乗としても一般性を失わない．

n	:	計算結果の関数値の桁数（B 進 n 桁）
$q(n)$	:	関数計算に必要な級数の項数 $(q(n)=2^{p(n)})$
$p(n)$	:	トーナメント有理数化処理の段数 $(p(n)=\log_2 q(n))$
$M(n)$	:	n 桁の乗算の計算量

表 3.1: 各関数におけるトーナメント有理数化処理の具体的係数

項番	関数記号	β	γ	$a_{0,l}$	$b_{0,l}$	$c_{0,k}$	$d_{0,k}$
1	$\exp\left(\frac{y}{x}\right)$	1	0	1	1	1	k
2	$\log\left(1+\frac{y}{x}\right)$	1	1	$(-1)^l$	$l+1$	1	1
3	$\sin\left(\frac{y}{x}\right)$	2	1	$(-1)^l$	1	1	$2k(2k+1)$
4	$\cos\left(\frac{y}{x}\right)$	2	0	$(-1)^l$	1	1	$2k(2k-1)$
5	$\arctan\left(\frac{y}{x}\right)$	2	1	$(-1)^l$	$2l+1$	1	1
6	$\arcsin\left(\frac{y}{x}\right)$	2	1	1	$2l+1$	$2k-1$	$2k$
7	$\mathrm{Erf}\left(\frac{y}{x}\right)$	2	1	$(-1)^l$	$2l+1$	1	k
8	$\mathrm{Si}\left(\frac{y}{x}\right)$	2	1	$(-1)^l$	$2l+1$	1	$2k(2k+1)$
9	$J_0\left(\frac{y}{x}\right)$	2	0	$(-1)^l$	1	1	$(2k)^2$
10	$\frac{(2u)^{3/2}}{12\pi}$	0	0	$v+wl$	$(-1)^l$	$(6k-5)\cdot(6k-3)\cdot$ $(6k-1)$	$(uk)^3$

注) $u=320160$, $v=13591409$, $w=545140134$ であり, $l=0,1,\cdots,\quad k=0,1,\cdots,l$ で $l-1$ や $k-1$ のように 0 以下の値となるときは 1 とする. 項番 10 の π の計算式は文献 [26] に示されている.

ここで，$q(n)$，$p(n)$ は n の関数で $q(n)$ と n の関係は次のようになる（3.6 節参照）．

(a) 級数の係数が指数的に減少する場合

$$O(q(n) \cdot \log_B q(n)) = O(n) \tag{3.3}$$

(b) 級数の係数が逆数的に減少する場合（係数が減少しない場合を含む）

$$O(q(n)) = O(n) \tag{3.4}$$

一方，有理数化処理の最終段の分子と分母の桁数は，入力値の精度が $O(1)$ 桁のため $O(\log_B(q(n)!))$，すなわち $O(q(n) \cdot \log_B q(n))$ となる．また有理数化処理では 2 項ずつ有理数を通分処理するため，一段前の処理では分子と分母の桁数は半減し項数は 2 倍となる．これから，級数に展開される関数の n 桁精度の関数値計算の計算量 $T(n)$ は次のようになる．

$$T(n) = O\left(\sum_{l=0}^{p(n)-1} 2^l \cdot M(2^{-l} \cdot q(n) \cdot \log_B q(n))\right)$$

現時点で知られている乗算の最良の計算量が $M(n) = O(n \cdot \log_2 n \cdot \log_2 \log_2 n)$[27] であることを考えると

$$M(n) \geq 2M\left(\frac{n}{2}\right) \geq 2^l M\left(\frac{n}{2^l}\right) \qquad l = 1, 2, \cdots n$$

が成立する．なお，この関係式は筆算による $M(n) = O(n^2)$ の場合や，Karatsuba 法に基づく $M(n) = O(n^{\log_2 3})$ の場合においても成立することに注意しておく．

l がある値より大きいとき $\dfrac{2^l}{\log_B q(n)} \geq 1$ が成立するため，次式が成立する．

$$\begin{aligned} 2^l \cdot M(\log_B q(n)) &\leq 2^l \cdot \frac{\log_B q(n)}{2^l} \cdot M\left(\log_B q(n) \cdot \frac{2^l}{\log_B q(n)}\right) \\ &= \log_B q(n) \cdot M(2^l) \qquad l = k, k+1, \cdots, p(n)-1 \end{aligned}$$

したがって，桁数 n がある値（数百桁）以上の多数桁の関数値計算では，級数に展開される関数の n 桁精度の関数値計算の計算量 $T(n)$ は次のようになる．

$$
\begin{aligned}
T(n) &= O\left(\sum_{l=0}^{p(n)-1} 2^l \cdot M(2^{-l} \cdot q(n) \cdot \log_B q(n))\right) \\
T(n) &\leq O\left(\sum_{l=0}^{p(n)-1} \log_B q(n) \cdot M(q(n))\right) \qquad (3.5)
\end{aligned}
$$

$$
\begin{aligned}
O\left(\sum_{l=0}^{p(n)-1} \log_B q(n) \cdot M(q(n))\right) &= O\left(M(q(n)) \cdot \log_B q(n) \cdot p(n)\right) \\
&= O\left(M(q(n)) \cdot \log_B q(n) \cdot \log_2 q(n)\right) \qquad (3.6)
\end{aligned}
$$

式 (3.3) と式 (3.4) における $q(n)$，n の関係を式 (3.5)，(3.6) に適用して計算量 $T(n)$ を評価すると次のようになる．

(a) 級数の係数が指数的に減少する場合：$O(q(n) \cdot \log_B q(n)) = O(n)$

$$
\begin{aligned}
T(n) &\leq O(M(q(n)) \cdot \log_B q(n) \cdot \log_2 q(n)) \\
O(M(q(n)) \cdot \log_B q(n) \cdot \log_2 q(n)) &\leq O(M(q(n) \cdot \log_B q(n)) \cdot \log_2 q(n)) \\
O(M(q(n) \cdot \log_B q(n)) \cdot \log_2 q(n)) &= O(M(n) \cdot \log_2 q(n))
\end{aligned}
$$

$\log_B q(n) > 1$ のため，式 (3.3) から $O(q(n)) \leq O(n)$ となる．したがって，次式が成立する．

$$
\begin{aligned}
O(M(n) \cdot \log_2 q(n)) &\leq O(M(n) \cdot \log_2 n) \\
T(n) &\leq O(M(n) \cdot \log_2 n)
\end{aligned}
$$

(b) 級数の係数が逆数的に減少する場合：$O(q(n)) = O(n)$ （係数が減少しない場合を含む）

$$
\begin{aligned}
T(n) &\leq O(M(q(n)) \cdot \log_B q(n) \cdot \log_2 q(n)) \\
O(M(q(n)) \cdot \log_B q(n) \cdot \log_2 q(n)) &= O(M(n) \cdot \log_B n \cdot \log_2 n) \\
O(M(n) \cdot \log_B n \cdot \log_2 n) &= O(M(n) \cdot (\log_2 n)^2)
\end{aligned}
$$

よって $T(n) \leq O(M(n) \cdot (\log_2 n)^2)$.

すなわち，級数に展開される関数の n 桁精度の関数値計算の計算量 $T(n)$ は，入力値が $O(1)$ 桁精度の有理数の場合には $O(M(n)\cdot(\log_2 n)^2)$ 以下になる．

3.4 入力値が多数桁精度 (m 桁) の場合の計算方法

級数で表現され加法定理が適用できる関数を考えよう．この場合には，入力値が基本区間外（例えば $exp(X)$ では $|X|\geq 1$）でも，適当な区間還元方式で区間内の実数の関数計算に帰着できるため，ここでは入力値は級数が収束する区間内の値とする．

まず m 桁精度の入力値 X が加法定理を使用して p 個（$O(\log_2 m)$ 個）に分割できる例として，関数 $\exp(X)$ と関数 $\sin(X)$ の計算を示す．表 3.2 に示す記号を使用すると $\exp(X)$ および $\sin(X)$ は次のように表される．

$$\begin{aligned}\exp(X) &= \exp\left(\frac{x_1}{B^{\alpha}}+\frac{x_2}{B^{2\alpha}}+\frac{x_3}{B^{4\alpha}}+\frac{x_4}{B^{8\alpha}}+\cdots+\frac{x_p}{B^{2^{p-1}\alpha}}\right)\\ &= \exp(a_1+a_2+a_3+a_4+\cdots+a_p)\\ &= \exp(a_1)\cdot\exp(a_2)\cdot\exp(a_3)\cdot\exp(a_4)\cdots\exp(a_p)\end{aligned}$$

$$\begin{aligned}\sin(X) &= \sin\left(\frac{x_1}{B^{\alpha}}+\frac{x_2}{B^{2\alpha}}+\frac{x_3}{B^{4\alpha}}+\frac{x_4}{B^{8\alpha}}+\cdots+\frac{x_p}{B^{2^{p-1}\alpha}}\right)\\ &= \sin(a_1+a_2+a_3+a_4+\cdots+a_p)\\ &= \sin(a_1)\cdot\cos(a_2+a_3+\cdots+a_p)+\cos(a_1)\cdot\sin(a_2+a_3+\cdots+a_p)\\ &= \sin(a_1)\cdot(\cos(a_2)\cdot\cos(a_3+\cdots+a_p)-\sin(a_2)\cdot\sin(a_3+\cdots+a_p))\\ &\quad+\cos(a_1)\cdot(\sin(a_2)\cdot\cos(a_3+\cdots+a_p)+\cos(a_2)\cdot\sin(a_3+\cdots+a_p))\end{aligned}$$

$$\begin{aligned}\sin(a_3+\cdots+a_p) &= \sin(a_3)\cdot\cos(a_4+\cdots+a_p)+\cos(a_3)\cdot\sin(a_4+\cdots+a_p)\\ \cos(a_3+\cdots+a_p) &= \cos(a_3)\cdot\cos(a_4+\cdots+a_p)-\sin(a_3)\cdot\sin(a_4+\cdots+a_p)\\ \sin(a_4+\cdots+a_p) &= \sin(a_4)\cdot\cos(a_5+\cdots+a_p)+\cos(a_4)\cdot\sin(a_5+\cdots+a_p)\\ \cos(a_4+\cdots+a_p) &= \cos(a_4)\cdot\cos(a_5+\cdots+a_p)-\sin(a_4)\cdot\sin(a_5+\cdots+a_p)\\ &\vdots\\ \sin(a_{p-1}+a_p) &= \sin(a_{p-1})\cdot\cos(a_p)+\cos(a_{p-1})\cdot\sin(a_p)\\ \cos(a_{p-1}+a_p) &= \cos(a_{p-1})\cdot\cos(a_p)-\sin(a_{p-1})\cdot\sin(a_p)\end{aligned}$$

次に，m 桁精度の入力値 X を加法定理を使用して p 個（$O(\log_2 m)$ 個）の有理数に分割できず，q 個（$O(\log_2 n)$ 個）に分割する例として関数 $\log(1+X)$ を示す．

表 3.2 に示す記号を使用すると $\log(1+X)$ は次のように表される．

$$\begin{aligned}\log(1+X) &= \log\left(\left(1+\frac{x_1}{B^{\alpha}}\right)\cdot\left(1+\frac{x_2}{B^{2\alpha}}\right)\cdot\left(1+\frac{x_3}{B^{4\alpha}}\right)\cdots\left(1+\frac{x_q}{B^{2^{q-1}\alpha}}\right)\right)\\ &= \log\left((1+a_1)\cdot(1+a_2)\cdot(1+a_3)\cdots(1+a_q)\right)\\ &= \log(1+a_1)+\log(1+a_2)+\log(1+a_3)+\cdots+\log(1+a_q)\end{aligned}$$

一般的に，m 桁の実数 X を上位桁から分母の桁数が $\alpha, 2\alpha, 4\alpha, \cdots, 2^{p-1}\alpha$ 桁 $(m \le 2^{p-1}\alpha)$ の p 個の有理数，または $\alpha, 2\alpha, 4\alpha, \cdots, 2^{q-1}\alpha$ 桁 $(n \le 2^{q-1}\alpha)$ の q 個の有理数に分ける．指数関数や三角関数では単純に実数 X を上位桁から分割できるため，p 個すなわち $O(\log_2 m)$ 個の有理数に分割できる．一方，対数関数や逆三角関数では，除算やさらに複雑な演算で実数 X を $\alpha, 2\alpha, 4\alpha, \cdots$ 桁の有理数に上位桁から順次割り戻すため，入力値 X が m 桁でも，関数値の精度と同等な n 桁の精度が必要となり，q 個すなわち $O(\log_2 n)$ 個の有理数に分割する必要がある．

このとき，上位桁から分割に対応する値（有理数）を表 3.2 表す．ここでの計算は B 進法とする．

表 3.2: 入力値 X の上位桁からの有理数による分割

小数点以下桁数	分母の桁数	値（有理数）
$1 \sim \alpha$	α	$a_1 \equiv \dfrac{x_1}{B^{\alpha}}$
$\alpha+1 \sim 2\alpha$	2α	$a_2 \equiv \dfrac{x_2}{B^{2\alpha}}$
$2\alpha+1 \sim 4\alpha$	4α	$a_3 \equiv \dfrac{x_3}{B^{4\alpha}}$
$\vdots$	$\vdots$	$\vdots$
$2^{S-2}\alpha+1 \sim 2^{S-1}\alpha$	$2^{S-1}\alpha$	$a_S \equiv \dfrac{x_S}{B^{2^{S-1}\alpha}}$

ここで，$S=p$ または $S=q$ であり，$\sin(X)$ では $X=\sum_{k=1}^{p} a_k$ で $\log(1+X)$ では $1+X=\prod_{k=1}^{q}(1+a_k)$ となるように各 a_k，すなわち各 x_k を定める．

一方，3.7 節より上位桁から B 進 α 桁と，$2^k\alpha$ 桁それぞれに分割した場合の各々の関数値の n 桁精度計算の計算量をそれぞれ $T_0(n)$，$T_k(n)$ とするとき $T_0(n) \ge T_k(n)$ である．

すなわち，分割した各入力値における関数値の n 桁精度計算の計算量 $T_k(n)$ は入力値が $O(1)$ 桁の有理数の n 桁精度の関数値の計算量と同等かそれ以下となる．

したがって，加法定理が適用でき入力値が m 桁精度の関数の n 桁精度の関数値の計算量は，入力値が $O(1)$ 桁の有理数の n 桁精度の関数値の計算量の $\log_2 m$ または $\log_2 n$ 倍かそれ以下となることが分かる．

また入力値が $O(1)$ 桁の有理数で，$O(n)$ の項数の計算が必要な級数関数で表現される n 桁精度関数値の計算量 $T(n)$ は，$O(M(n)\cdot(\log_2 n)^2)$ 以下となることを 3.3 節で説明した．

このように加法定理が適用でき，かつ級数で表現される関数は入力値が n 桁精度で，n 桁精度の関数値を計算する計算量は $O(M(n)\cdot(\log_2 n)^3)$ 以下となる．

3.5 並列化と結果の再利用

本 DRM 法は入力値 X を上位桁から複数個の有理数に分割し，加法定理で各分割に対応する関数値から最終的に求める関数値を計算する部分と，トーナメント有理数化処理で各分割ごとの入力値に対応する関数値を計算する部分から構成される．

3.2 節で説明したトーナメント有理数化処理では，各段でのトーナメント有理数化の計算は完全に独立であり並列化できる．さらに，加法定理を使用して入力値を分割した各関数は互いに独立であり, 分割関数ごとに並列計算可能である．

計算結果の再利用性に関しては，トーナメント有理数化処理は有理数で表現しているため，最終段の除算による実数化を除き，正確な値で表現されている．そのため，計算桁数を増加する場合は，増加する前に計算した項数分の級数の和として作成した有理数はそのまま使用でき，追加する桁数に対応する部分以降の項の級数の有理数化処理を追加実行すればよい．

また，入力値 X の桁数 n が増加した場合も，上位桁から複数個の有理数に分割して各分割ごとに関数値を計算しているため，それまでにトーナメント有理数化処理で計算した有理数は有効に再利用可能である．

3.6 計算桁数 n と級数の項数 $q(n)$ の関係

級数に展開される関数の関数値を B 進 n 桁精度で計算するときに必要な項数 $q(n)$ と n の関係式を示す．

(a) 級数の係数が指数的に減少する場合

これは指数関数，三角関数等の場合である．

入力値を X とすると，計算項数 $q(n)$ と計算桁数 n の関係式は次のようになる．

$$O\left((q(n)!)\cdot X^{-q(n)}\right)=O(B^n)$$

n が大きい場合，$q(n)!$ は Stirling の公式により $\sqrt{2\pi}\cdot q(n)^{q(n)+\frac{1}{2}}\cdot e^{-q(n)}$ と近似される．したがって $q(n)$ と n の関係は次のようになる．

$$O\left(\sqrt{2\pi}\cdot q(n)^{q(n)+\frac{1}{2}}\cdot e^{-q(n)}\cdot X^{-q(n)}\right)=O\left(q(n)^{q(n)}\cdot(e\cdot X)^{-q(n)}\right)=O(B^n)$$

これから次式となる．

$$O\left(q(n)\cdot(\log_B q(n)-\log_B(e\cdot X))\right)=O(n)$$

X と e は定数であるため，下記が成立する．

$$O\left(q(n)\cdot\log_B q(n)\right)=O(n)$$

(b) 級数の係数が逆数的に減少する場合

これは対数関数，逆三角関数等の場合である．ここでは係数が減少しない最悪のケースも同じであり，評価は最悪のケースで行う．

入力値を X（対数関数の場合は $1+X$）とすると，計算項数 $q(n)$ と計算桁数 n の関係式は次のようになる．

$$O\left(X^{-q(n)}\right)=O(B^n)$$

これから次式となる．

$$O\left(-q(n)\cdot\log_B X\right)=O(n)$$

X は 1 より小さい定数なので次式が成立する．

$$O\left(q(n)\right)=O(n)$$

3.7 級数関数 $f\left(\frac{1}{B^{2^k\alpha}}\right)$ の n 桁精度計算の計算量

加法定理を使用するため，上位桁から B 進 α 桁と，$2^k\alpha$ 桁のそれぞれに分割した場合の各々の n 桁精度計算の計算量を比較する．

級数関数 $f\left(\frac{1}{B^{\alpha}}\right)$ の計算項数を q，計算量を $T_0(n)$ とし，$f\left(\frac{1}{B^{2^k\alpha}}\right)$ の計算項数を r，計算量を $T_k(n)$ とする．

評価を単純化するため係数が指数的に減少する場合として $f(X)=\sum_{l=0}^{\infty}\frac{X^l}{l!}$ を，係数が減少しない場合として $f(X)=\sum_{l=0}^{\infty}X^l$ を考える．

(a) 級数の係数が指数的に減少する場合

入力値の 1 項当りの桁数は $f\left(\frac{1}{B^{\alpha}}\right)$ の計算で $\alpha+\log_B q$ となり，$f\left(\frac{1}{B^{2^k\alpha}}\right)$ の計算では $2^k\alpha+\log_B r$ となるため次式が得られる．

$$\begin{aligned}
T_0(n) &= O\left(\sum_{l=1}^{\lceil\log_2 q\rceil}2^{-l}q\cdot M(2^l\cdot(\alpha+\log_B q))\right)\\
&= O\left(\sum_{l=0}^{\lceil\log_2 q\rceil-1}2^{l}\cdot M(2^{-l}q\cdot(\alpha+\log_B q))\right)\\
T_k(n) &= O\left(\sum_{l=0}^{\lceil\log_2 r\rceil-1}2^{l}\cdot M(2^{-l}r\cdot(2^k\alpha+\log_B r))\right)
\end{aligned}$$

関数値計算の入力値が α 桁と $2^k\alpha$ 桁の各々の計算項数 q と r 及び計算桁数 n との関係は下記のようになる．

$$O\left((q!)\cdot(B^{\alpha})^q\right) = O\left((r!)\cdot(B^{2^k\alpha})^r\right)=O\left(B^n\right)$$

これに $q!$，$r!$ に Stirling の近似公式を適用すると次式が得られる．

$$\begin{aligned}
O((q\cdot B^{\alpha})^q) &= O((r\cdot B^{2^k\alpha})^r)\\
O(q\cdot(\alpha+\log_B q)) &= O(r\cdot(2^k\alpha+\log_B r))\\
O\left(M(2^{-l}q\cdot(\alpha+\log_B q))\right) &= O\left(M(2^{-l}r\cdot(2^k\alpha+\log_B r))\right)\\
O\left(\sum_{l=0}^{\lceil\log_2 q\rceil-1}2^{l}\cdot M(2^{-l}q\cdot(\alpha+\log_B q))\right) &\geq O\left(\sum_{l=0}^{\lceil\log_2 r\rceil-1}2^{l}\cdot M(2^{-l}r\cdot(2^k\alpha+\log_B r))\right)
\end{aligned}$$

したがって，$T_0(n)\geq T_k(n)$ となる．

(b) 級数の係数が逆数的に減少する場合（係数が減少しない場合を含む）

関数値計算のの入力値が α 桁と $2^k\alpha$ 桁の各々の計算項数 q と r 及び計算桁数 n との関係は下記のようになる．

$$O\left((B^{\alpha})^{q}\right) = O\left((B^{\alpha 2^k})^{r}\right) = O\left(B^{n}\right)$$

この式より $O(q) = O(2^k r)$ となる．

一方，$T_0(n)$ と $T_k(n)$ は次のようになる．

$$\begin{aligned} T_0(n) &= O\left(\sum_{l=0}^{\lceil \log_2 q \rceil - 1} 2^l \cdot M(2^{-l} q\alpha)\right) \\ T_k(n) &= O\left(\sum_{l=0}^{\lceil \log_2 r \rceil - 1} 2^l \cdot M(2^{-l} r 2^k \alpha)\right) \end{aligned}$$

したがって，$q > r$ を考慮すると $T_0(n) \geq T_k(n)$ となる．

3.8 連分数の有理数化処理

関数値 f が q 項までの連分数

$$f = \cfrac{a_{0,1}}{b_{0,1} + \cfrac{a_{0,2}}{b_{0,2} + \cfrac{a_{0,3}}{b_{0,3} + \cdots \cfrac{a_{0,q-2}}{b_{0,q-2} + \cfrac{a_{0,q-1}}{b_{0,q-1} + \cfrac{a_{0,q}}{b_{0,q}}}}}}}$$

に展開されるとする．これを以下のように，トーナメント方式を適用して 2 項ずつ通分処理で有理数にする．1 段目の有理数化処理は下記のようになる．

$$\frac{a_{1,l}}{b_{1,l}} \equiv \cfrac{a_{0,2l-1}}{b_{0,2l-1} + \cfrac{a_{0,2l}}{b_{0,2l} + \cfrac{a_{1,l+1}}{b_{1,l+1}}}} = \frac{T_{1,l} a_{1,l+1} + U_{1,l} b_{1,l+1}}{V_{1,l} a_{1,l+1} + W_{1,l} b_{1,l+1}}$$

$$T_{1,l} \equiv a_{0,2l-1}, \quad U_{1,l} \equiv a_{0,2l-1} b_{0,2l}$$

$$V_{1,l} \equiv b_{0,2l-1}, \quad W_{1,l} \equiv a_{0,2l} + b_{0,2l-1} b_{0,2l}$$

ここで $l=1,2,\cdots,\lceil q/2\rceil$，$a_{1,\lceil q/2\rceil+1}=0$，$b_{1,\lceil q/2\rceil+1}=1$ で p が奇数なら $a_{0,p+1}$，$b_{0,p+1}$ は共に 0 と仮定する．

また，p 段目の有理数化処理は 2 回漸化式を使用して下記のようになる．

$$\begin{aligned}
\frac{a_{p,l}}{b_{p,l}} &\equiv \frac{a_{p-1,2l-1}}{b_{p-1,2l-1}}=\frac{T_{p-1,2l-1}a_{p-1,2l}+U_{p-1,2l-1}b_{p-1,2l}}{V_{p-1,2l-1}a_{p-1,2l}+W_{p-1,2l-1}b_{p-1,2l}}\\
&= \frac{T_{p-1,2l-1}(T_{p-1,2l}a_{p-1,2l+1}+U_{p-1,2l}b_{p-1,2l+1})+U_{p-1,2l-1}(V_{p-1,2l}a_{p-1,2l+1}+W_{p-1,2l}b_{p-1,2l+1})}{V_{p-1,2l-1}(T_{p-1,2l}a_{p-1,2l+1}+U_{p-1,2l}b_{p-1,2l+1})+W_{p-1,2l-1}(V_{p-1,2l}a_{p-1,2l+1}+W_{p-1,2l}b_{p-1,2l+1})}\\
&\equiv \frac{T_{p,l}a_{p-1,2l+1}+U_{p,l}b_{p-1,2l+1}}{V_{p,l}a_{p-1,2l+1}+W_{p,l}b_{p-1,2l+1}}=\frac{T_{p,l}a_{p,l+1}+U_{p,l}b_{p,l+1}}{V_{p,l}a_{p,l+1}+W_{p,l}b_{p,l+1}}\\
T_{p,l} &\equiv T_{p-1,2l-1}T_{p-1,2l}+U_{p-1,2l-1}V_{p-1,2l},\quad U_{p,l}\equiv T_{p-1,2l-1}U_{p-1,2l}+U_{p-1,2l-1}W_{p-1,2l}\\
V_{p,l} &\equiv V_{p-1,2l-1}T_{p-1,2l}+W_{p-1,2l-1}V_{p-1,2l},\quad W_{p,l}\equiv V_{p-1,2l-1}U_{p-1,2l}+W_{p-1,2l-1}W_{p-1,2l}
\end{aligned}$$

ここで，$p=2,3,\cdots,\lceil\log_2 q\rceil$，$l=1,2,\cdots,\lceil q/2^p\rceil$

トーナメント有理数化の p 段では $T_{p,l}$，$U_{p,l}$，$V_{p,l}, W_{p,l}$ と $a_{p,\lceil q/2^p\rceil}$，$b_{p,\lceil q/2^p\rceil}$ を計算する．最終段の $\dfrac{a_{\lceil\log_2 q\rceil,1}}{b_{\lceil\log_2 q\rceil,1}}$ で結果の有理数が求まる．

3.9 多数桁の基底変換

多数桁の基底変換は，DRM 法を適用して各段で変換桁数を倍増させることを繰り返して実行できる．p 進数から q 進数に基底変換するには，与えられた自然数 X を

$$X=\sum_{k=0}^{m-1}A_{k,0}p^k,\quad m=\lceil\log_p X\rceil,\quad 0\le A_{k,0}<p$$

なる表現から

$$X=\sum_{k=0}^{n-1}b_{k,0}q^k,\quad n=\lceil\log_q X\rceil,\quad 0\le b_{k,0}<q$$

に変換すればよい．

適当な整数を用いると α を $p^\alpha<q<p^{2\alpha}$ 又は $p<q^\alpha<p^2$ とできるため，該当する p^α, q^α を p, q と以下再表示し，項数 m も 2 のベキ乗 $(m=2^r)$ と仮定しても一般性は失われない．

多数桁の基底変換は下記の処理を j が 1 から r まで繰り返すことで実行できる．

$$X = \sum_{k=0}^{2^{r-j+1}-1}A_{k,j-1}p^{2^{j-1}k}=\sum_{k=0}^{2^{r-j}-1}\left(A_{2k,j-1}+A_{2k+1,j-1}p^{2^{j-1}}\right)p^{2^j k}$$

$$\equiv \sum_{k=0}^{2^{r-j}-1} A_{k,j} p^{2^j k} \equiv \sum_{k=0}^{2^{r-j}-1} \left(B_{2k,j} + B_{2k+1,j} q^{2^{j-1}} \right) p^{2^j k}$$
$$\equiv \sum_{k=0}^{2^{r-j}-1} \left(\sum_{i=0}^{2^j-1} b_{i+2^{r-j}k,r-j} q^i \right) p^{2^j k}$$

ここで，j 段では下記の計算をする．

$$W_{k,j} = A_{2k,j-1} + A_{2k+1,j-1} p^{2^{j-1}}$$
$$B_{2k+1,j} = \lfloor \frac{W_{k,j}}{q^{2^{j-1}}} \rfloor, \quad B_{2k,j} = W_{k,j} - B_{2k+1,j} q^{2^{j-1}}, \quad k = 0, 1, 2, \cdots, 2^{r-j} - 1$$

3.10 DRM 法の sin 関数と log 関数への適用結果

DRM 法を sin 関数と log 関数に適用した場合の結果を図 3.1 に示す．図 3.1 は DRM 法を適用して 10 進 n 桁の関数値を計算したときに使用した乗算の合計計算量を示す．10 進 n 桁の乗算の計算量 $M(n)$ とのオーダーの比較をするため，縦軸は $M(n)$ の計算量を $n \log_2 n$ とする単位で示す．比較のため $M1(n) = O(M(n) \log_2 n)$ と $M2(n) = O(M(n)(\log_2 n)^2)$ も点線で示す．$\sin(0.5)$ 及び $\log(1.5)$ はそれぞれ $O(1)$ 桁の入力値 0.5 と 1.5 を与えて，横軸に示す 10 進 n 桁の関数値を計算することを示す．一方，$\sin(0.5\ldots)$ 及び $\log(1.5\ldots)$ はそれぞれ n 桁の入力値 0.5... と 1.5... を与えて同一桁の関数値を計算することを示す．

これより $\sin(0.5)$ の計算量のオーダーははは $O(M(n) \log_2 n)$ とほぼ等しく，$\log(1.5)$ 及び入力値が n 桁の $\sin(0.5\ldots)$ と $\log(1.5\ldots)$ の計算量のオーダーはいずれも $O(M(n)(\log_2 n)^2)$ より少し小さいことが分かる．したがって，計算量の理論的オーダーをいずれも満足していることが分かる．また，入力値が n 桁の $\log(1.5\ldots)$ に関しては理論的オーダー $(O(M(n)(\log_2 n)^3))$ が過大評価であることも分かる．

3.11 関連研究

提案した DRM 法を構成する二つの方法のうち，最初のトーナメント有理数化処理に関するものは文献 [9, 31, 36] に関連研究がある．一方，入力値を有理数に分割して加法定理を使用しトーナメント有理数化処理と結合する方式の提案は他に見当たらない．なお一般的な分割統治法が適応できる計算で，トーナメント方式による計算量の $O(n \log_2 n)$ への削減と並列化適応性については 1981 年の佐々木他の論文 [35] に示されている．

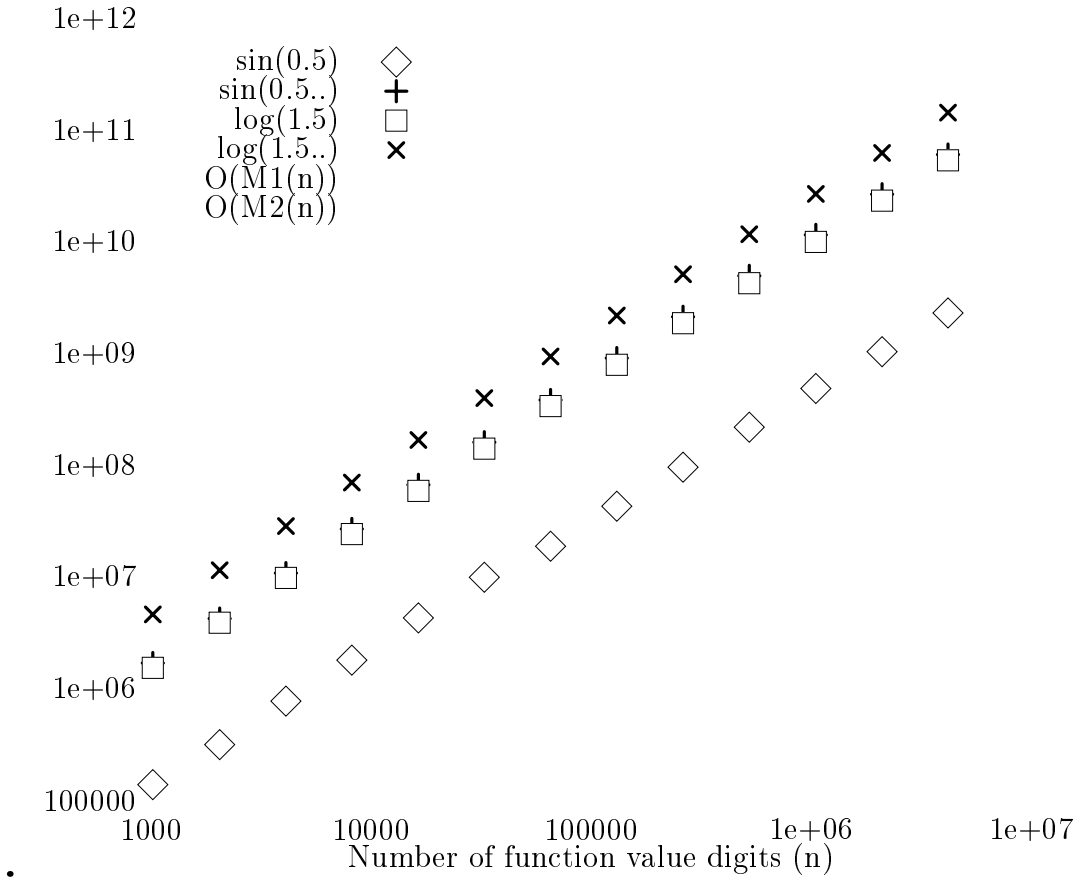

図 3.1: DRM 法の sin 関数と log 関数への適用結果

Haible 他の論文 [31] 及び右田他の論文 [36] は入力値の精度が $O(1)$ 桁に限定されており，後他の論文 [7, 8] のように加法定理と結合して多数桁関数システムとして閉じることは考慮されていない．

3.12 まとめ

n 桁の乗算の計算量を $M(n)$ とするとき三角関数や指数関数および誤差関数等の級数で表現される関数を n 桁の精度で求める際は，提案した DRM 法を使用して計算量が削減できることを示した．

提案した DRM 法は二つの方法で構成される．一つは入力値の精度が $O(1)$ 桁の有理数の場合に，級数に展開される関数で n 桁精度の関数値計算における計算量を $O\left(M(n)\cdot(\log_2 n)^2\right)$ 以下とする方法である．もう一つは加法定理が適用できる関数の多数桁計算で，入力値を上位桁から分母の桁数が $\alpha, 2\alpha, 4\alpha, \cdots, 2^{p-1}\alpha$ 桁ずつの有理数に分割し，各分割した関数値の計算量を $T(n)$ とするとき，入力が n 桁精度で，n 桁精度の関数値の計算量は $O\left(T(n)\cdot\log_2 n\right)$ とする方法である．この二つの方法を結合して，入力値が n 桁精度の場合に n 桁精度の関数値計算の計算量を $O\left(M(n)\cdot(\log_2 n)^3\right)$ 以下に削減するのが DRM 法である．

本 DRM 法は連分数で表現される関数値の多数桁計算にも，級数で表現される関数と同様に適用可能である．また，2 進 10 進変換に代表される多数桁の基底変換にも適用可能である．

本 DRM 法は $O(1)$ 桁の入力に対する多数桁精度を必要とする関数値計算で，Brent のアルゴリズム [25] と同一の計算量を持つが，Brent のアルゴリズムには示されていない誤差関数のように特殊関数でも級数で展開される関数に適用可能である．そのため適用可能な関数の範囲が広く計算方法が単純で，多くの関数値計算まで含めた多数桁計算システムの構築に有用である．また，自然な並列化が可能であり，計算桁数を増加するとき計算済みの有理数が再利用可能な点，さらに Brent のアルゴリズムのように複数の複雑な関係式を使用して計算するのではなく，単純に級数展開の公式をそのまま適用できるため分かり易い点が特長である．

今後の課題として，DRM 法に基づく高速高精度な数学関数計算パッケージの作成と性能評価がある．

第4章　多倍長精度の高速行列乗算

4.1　はじめに

4 倍精度以上の多数桁を係数とする連立一次方程式や固有値問題の高速計算を目的に，その基本演算となる多倍長精度（比較的短い多数桁）の行列乗算を提案する．多数桁の乗算には入力値を一定の桁数に分割して筆算方式（定義式）で計算する方法以外に，中国剰余定理 (CRT，百五演算) または高速フーリエ変換 (FFT) を使用する方法が知られている．しかし，これらの方法が筆算方式の計算より高速となるには桁数が数百桁以上と長い必要がある．一方，多倍長精度の係数をもつ n 次元の行列乗算に中国剰余定理や FFT を多数桁乗算用に拡張した高速剰余変換 (FMT) を適用する場合は，それらの線形性を利用して中国剰余定理や FMT の適用回数を $O(n^3)$ から $O(n^2)$ に削減することができる．今回の提案はそれらの線形性を利用して，多倍長精度の係数をもつ n 次元の行列乗算に，$O(n^2)$ 回の中国剰余定理または FFT を適用する高速乗算方式である．中国剰余定理は桁数が数十桁と比較的短い時に有利であり，FFT は桁数が数百桁と長いときに有利である．

4.2　多倍長精度の行列乗算

4.2.1　多倍長精度行列乗算の考え方

行列乗算 $C = A \cdot B$ の計算を多倍長精度で行うものとする．A, B, C はそれぞれ $n \times l$，$l \times m$，$n \times m$ 次元の行列に適用可能であるが，以下の説明ではすべて $n \times n$ 次元の行列で，各要素は固定小数点の多倍長精度とする．入力行列 A, B の各要素は m 個の数値に分割して保存され，出力行列 C の各要素は入力の 2 倍強の桁数である $2m + 1$ 個の数値に分割保存されているものとする．また，分割保存された個々の数値の乗算は倍精度浮動小数点演算で正確に実行できるものとする．

$n \times n$ 次元の行列 A, B, C の各要素を

$$A = (a_{ij}), \quad B = (b_{ij}), \quad C = (c_{ij}), \qquad i, j = 1, 2, \cdots, n$$

とし，多数桁の値を持つ入力値の各要素は m 個に分割され

$$a_{ij} = a_{ij}^{(m-1)} \cdot E^{m-1} + \cdots + a_{ij}^{(1)} \cdot E + a_{ij}^{(0)}$$
$$b_{ij} = b_{ij}^{(m-1)} \cdot E^{m-1} + \cdots + b_{ij}^{(1)} \cdot E + b_{ij}^{(0)}$$

と表示され，出力値の各要素は $2m+1$ 個に分割され

$$c_{ij} = c_{ij}^{(2m)} \cdot E^{2m} + \cdots + c_{ij}^{(1)} \cdot E + c_{ij}^{(0)}$$

と表示されるものとする．

ここで，E は分割した数値の基本単位で IEEE の倍精度浮動小数点数では，2 進の場合は $E = 2^{20}$，10 進の場合は $E = 10^6$ とする．実 FFT を適用する場合は，丸め誤差を考慮して整数値に正確に戻す必要があるため，$E = 2^{14}$ 又は $E = 10^4$ とする．

4.2.2 筆算方式

行列乗算 $C = A \cdot B$ の各要素の計算を下記で行なう．

$$c_{ij} = \sum_{l=1}^{n} a_{il} \cdot b_{lj}, \qquad i, j = 1, 2, \cdots, n \tag{4.1}$$

ここで，各要素ごとに多数桁計算を下記で行なう．

$$(a \cdot b)^{(k)} = \sum_{p=max(0,k-m)}^{min(k,m)} a^{(p)} \cdot b^{(k-p)}, \qquad k = 0, 1, \cdots, 2(m-1) \tag{4.2}$$

式 (4.1) を式 (4.2) に代入して，行列 C の各要素 $c_{ij}(i, j = 1, 2, \cdots, n)$ の各分割表示される値 $c_{ij}^{(k)}$ を下記で計算する．

$$\begin{aligned} c_{ij}^{(2m)} &= 0, \qquad c_{ij}^{(2m-1)} = 0, \\ c_{ij}^{(k)} &= \sum_{l=1}^{n} \left(\sum_{p=max(0,k-m)}^{min(k,m)} a_{il}^{(p)} \cdot b_{lj}^{(k-p)} \right), \qquad k = 0, 1, \cdots, 2(m-1) \end{aligned} \tag{4.3}$$

ここで，$c_{ij}^{(k)}$ は

$$|c_{ij}^{(k)}| \leq nm \cdot E^2, \qquad k = 0, 1, \cdots, 2(m-1)$$

となる．計算された $c_{ij}^{(k)}$ は倍精度浮動小数点で正確に表現できる必要がある．
また E を単位として $c_{ij}^{(k)}$ は 2 単位桁上げとなり，$c_{ij}^{(2m-1)}$，$c_{ij}^{(2m)}$ も桁上げで恒常的なゼロではなくなる．更に，符号は各要素で一括管理し，c_{ij} の成分 $c_{ij}^{(k)}$ は

$$0 \leq c_{ij}^{(k)} < E$$

となるように $c_{ij}^{(k)}/E$ の整数部は $c_{ij}^{(k)}$ に桁上げ処理をする．

4.2.3 中国剰余定理による方式

計算手順

下記のような手順で行列乗算 $C = A \cdot B$ の各要素 $c_{ij}, (i, j = 1, 2, \cdots, n)$ を計算する．

(1) 互いに素な整数 p_k を用意
結果 c_{ij} が表示可能なように E に近い互いに素な整数 p_k を $2m+1$ 個用意する．この時，下記の条件が必要であるが，E に近い互いに素な整数を $2m+1$ 個用意すれば，通常 $\sigma mn < E$ のため満足する．結果の符号の判定を安全に判定するため σ は 4 以上が必要である．

$$|c_{ij}| < 2mnE^{2m} < E^{2m+1}/\sigma \simeq \left(\prod_{k=1}^{2m+1} p_k \right) /\sigma \tag{4.4}$$

(2) 各整数ごとに剰余計算
入力行列の各要素 a_{ij}, b_{ij} にたいする各整数 p_k に対する剰余を計算する．

$$\begin{aligned} \alpha_{ij}^{(k)} &= a_{ij} \pmod{p_k} \\ \beta_{ij}^{(k)} &= b_{ij} \pmod{p_k}, \\ & \qquad i, j = 1, 2, \cdots, n, \quad k = 1, 2, \cdots, 2m+1 \end{aligned} \tag{4.5}$$

(3) 各剰余ごとに行列乗算

$$\begin{aligned} \theta_{ij}^{(k)} &= \sum_{l=1}^{n} \alpha_{il}^{(k)} \cdot \beta_{lj}^{(k)}, \\ & \qquad i, j = 1, 2, \cdots, n, \quad k = 1, 2, \cdots, 2m+1 \end{aligned} \tag{4.6}$$

(4) 中国剰余定理で結果を計算
中国剰余定理（CRT）で各剰余から結果 c_{ij} を計算する．記号 $CRT(a_1, \cdots, a_m)$ は $a_1, \cdots, a_m$ に中国剰余定理を使用して，多数桁に戻す計算を示す．

$$
\begin{aligned}
c_{ij} &= CRT(\theta_{ij}^{(1)}, \theta_{ij}^{(2)}, \cdots, \theta_{ij}^{(2m+1)}), \\
&\qquad i, j = 1, 2, \cdots, n
\end{aligned} \tag{4.7}
$$

c_{ij} は下記の形で求める．

$$
c_{ij} = c_{ij}^{(2m)} \cdot E^{2m} + \cdots + c_{ij}^{(1)} \cdot E + c_{ij}^{(0)} \tag{4.8}
$$

但し，$c_{ij}^{(k)}$ は下記の様にする．

$$
0 \leq c_{ij}^{(k)} < E, \quad k = 0, 1, \cdots, 2m
$$

(5) 結果の正規化

正規化とは符号の決定と桁上げ処理である．ここでは，結果の符号の判定方法を述べる．

中国剰余定理により求めた値は，各整数の積 $(H = \prod_{k=1}^{(2m+1)} p_k)$ を法とするものである．このため，中間結果は正で表現し本来の結果が負の場合の判定を必要とする．符号の判定は最上位の桁で行ない，定数 h を定めておいて $c_{ij}^{(2m)} \leq h$ なら結果の符号は正と判定し，それ以外なら負と判定する．符号が負の場合は c_{ij} は $c_{ij} = H - c_{ij}$ で求める．

中国剰余定理での計算

各 $c_{ij}, (i, j = 1, 2, \cdots, n)$ を $c_{ij} = c_{ij}^{(2m)} \cdot E^{2m} + \cdots + c_{ij}^{(1)} \cdot E + c_{ij}^{(0)}$ の形で表示し，以下の計算を行なう．

(1) 前処理

$c_{ij} = c_{ij}^{(0)} = \theta_{ij}^{(1)}$

(2) 主計算

下記を $k = 2$ から $2m + 1$ まで繰り返す．

$$
\begin{aligned}
v &= c_{ij} \qquad (\mathrm{mod}\ p_k) \\
w &= (\theta_{ij}^{(k)} - v) \cdot t_k \qquad (\mathrm{mod}\ p_k) \\
c_{ij} &= c_{ij} + w \cdot Q_k
\end{aligned} \tag{4.9}
$$

但し，多倍長精度の定数 Q_k と定数 t_k は前もって下記で求めておく．

$$
\begin{aligned}
Q_k &= \prod_{l=1}^{k-1} p_l \\
t_k &= 1/Q_k \qquad (\mathrm{mod}\ p_k), \qquad k = 2, 3, \cdots, 2m + 1
\end{aligned}
$$

計算のための注意事項

(1) $\alpha_{ij}^{(k)} = a_{ij} \pmod{p_k}$ 等の剰余の計算
$d_l^{(k)} = E^{(l)} \pmod{p_k}$ を先に計算しておき，a_{ij} が $a_{ij}^{(k)}$ で表示されるのを利用して次のように計算する．

$$\begin{aligned} \alpha_{ij}^{(k)} &= \sum_{l=0}^{m-1} a_{ij}^{(l)} \cdot d_l^{(k)} \pmod{p_k}, \\ & \qquad k = 1, 2, \cdots, 2m+1, \quad i, j = 1, 2, \cdots, n \end{aligned}$$

(2) 符号の決定に使用する基準値 h の算出
$H = \prod_{k=1}^{2m+1} p_k$ を計算し，下記のように表示する．

$$\begin{aligned} H &= \prod_{k=1}^{2m+1} p_k \\ &= h^{(2m)} \cdot E^{2m} + h^{(2m-1)} \cdot E^{2m-1} + \cdots + h^{(1)} \cdot E + h^{(0)} \end{aligned} \tag{4.10}$$

一方，式 (4.4) から

$$-H/\sigma < c_{ij} < H/\sigma, \qquad \sigma \geq 4$$

となる．このため $h = h^{(2m)}/2$ としておけば，式 (4.9) で求めた c_{ij} の桁上げ後の先頭要素 $c_{ij}^{(2m)}$ と h を比較して，c_{ij} の正負が判定できる．

4.2.4 実 FFT 変換による方式

FMT(高速剰余変換) と FFT(高速フーリエ変換) は多数桁の乗算に関しては同じ計算式になるので，ここでは多数桁乗算で一般的に知られている FFT で説明する．下記のような手順で行列乗算 $C = A \cdot B$ の各要素 $c_{ij}, (i, j = 1, 2, \cdots, n)$ を計算する．

入力の順実 FFT 変換

行列乗算の多数桁入力値 a_{ij}，b_{ij} をそれぞれ m 個に分割し，式 (4.11) に示す．

$$\begin{aligned} a_{ij} &= a_{ij}^{(m-1)} \cdot E^{m-1} + a_{ij}^{(m-2)} \cdot E^{m-2} + \cdots + a_{ij}^{(1)} \cdot E + a_{ij}^{(0)} \\ b_{ij} &= b_{ij}^{(m-1)} \cdot E^{m-1} + b_{ij}^{(m-2)} \cdot E^{m-2} + \cdots + b_{ij}^{(1)} \cdot E + b_{ij}^{(0)}, \\ & \qquad i, j = 1, 2, \cdots, n \end{aligned} \tag{4.11}$$

a_{ij}，b_{ij} 共に後半の要素をゼロにした，それぞれ $2m$ 個の要素を持つ式 (4.12) のようなベクトル $\tilde{a}_{ij}$，$\tilde{b}_{ij}$ を作成する．

$$\begin{aligned}\tilde{a}_{ij} &= (a_{ij}^{(m-1)}, a_{ij}^{(m-2)}, \cdots, a_{ij}^{(0)}, 0, \cdots, 0) \\ \tilde{b}_{ij} &= (b_{ij}^{(m-1)}, b_{ij}^{(m-2)}, \cdots, b_{ij}^{(0)}, 0, \cdots, 0), \qquad i,j = 1,2,\cdots,n \end{aligned} \tag{4.12}$$

$\tilde{a}_{ij}$，$\tilde{b}_{ij}$ に対して，式 (4.13) のような順実 FFT 変換を行う．

$$\begin{aligned}\bar{a}_{ij} &= FFT(\tilde{a}_{ij}) \\ \bar{b}_{ij} &= FFT(\tilde{b}_{ij}), \qquad i,j = 1,2,\cdots,n \end{aligned} \tag{4.13}$$

項別内積計算

a_{ij}，b_{ij} の順実 FFT 結果 $\bar{a}_{ij}$，$\bar{b}_{ij}$ に対して，下記の様に項別に内積計算する．

$$f_{ij} = \sum_{l=1}^{n} \bar{a}_{il} \otimes \bar{b}_{lj}, \qquad i,j = 1,2,\cdots,n$$

ここで，$\otimes$ はベクトルの対応する要素ごとの計算である． ベクトル $\bar{a}_{ij}$，$\bar{b}_{ij}$，f_{ij} を下記のように表す．

$$\begin{aligned}\bar{a}_{ij} &= (\bar{a}_{ij}^{(2m-1)}, \bar{a}_{ij}^{(2m-2)}, \cdots, \bar{a}_{ij}^{(0)}) \\ \bar{b}_{ij} &= (\bar{b}_{ij}^{(2m-1)}, \bar{b}_{ij}^{(2m-2)}, \cdots, \bar{b}_{ij}^{(0)}) \\ f_{ij} &= (f_{ij}^{(2m-1)}, f_{ij}^{(2m-2)}, \cdots, f_{ij}^{(0)}), \qquad i,j = 1,2,\cdots,n \end{aligned}$$

具体的な要素ごとの内積計算を式 (4.14) に示す．

$$\begin{aligned} f_{ij}^{(2m-1)} &= \sum_{l=1}^{n} \bar{a}_{il}^{(2m-1)} \cdot \bar{b}_{lj}^{(2m-1)}, \qquad f_{ij}^{(m-1)} = \sum_{l=1}^{n} \bar{a}_{il}^{(m-1)} \cdot \bar{b}_{ij}^{(m-1)} \\ f_{ij}^{(k+m)} &= \sum_{l=1}^{(n)} \left(\bar{a}_{il}^{(k+m)} \cdot \bar{b}_{lj}^{(k+m)} - \bar{a}_{il}^{(k)} \cdot \bar{b}_{lj}^{(k)} \right) \\ f_{ij}^{(k)} &= \sum_{l=1}^{(n)} \left(\bar{a}_{il}^{(k+m)} \cdot \bar{b}_{lj}^{(k)} + \bar{a}_{il}^{(k)} \cdot \bar{b}_{lj}^{(k+m)} \right), \\ & \qquad i,j = 1,2,\cdots,n, \quad k = 0,1,\cdots,m-2 \end{aligned} \tag{4.14}$$

逆実 FFT 変換

正規化されていない乗算結果 $\hat{c}_{ij}$ を f_{ij} から下記の逆実 FFT 変換で求める．

$$\hat{c}_{ij} = FFT^{-1}(f_{ij}), \qquad i,j = 1,2,\cdots,n$$

結果の正規化

正規化とは逆実 FFT 変換で得られた誤差を含む乗算結果を整数に戻すこと，所定の桁数に桁上げすることである．正規化前の各 c_{ij} を

$$\begin{aligned}\hat{c}_{ij} &= \hat{c}_{ij}^{(2m-1)} \cdot E^{2m-1} + \hat{c}_{ij}^{(2m-2)} \cdot E^{2m-2} + \cdots + \hat{c}_{ij}^{(1)} \cdot E + \hat{c}_{ij}^{(0)}, \\ & \qquad i,j = 1,2,\cdots,n\end{aligned}$$

と表示したとき，各 $c_{ij}^{(k)}$ は整数のはずであるが誤差のため小数点以下の値が発生する．各 $c_{ij}^{(k)}$ の整数化とは，誤差を含んで実数となっている値をその実数に最も近い整数に戻すことである．桁上げ処理は，$-n \cdot m \cdot E^2 < \hat{c}_{ij}^{(k)} < n \cdot m \cdot E^2$ である $\hat{c}_{ij}^{(k)}$ に対して E を超えた値を桁上げして，c_{ij} を下記のように表示することである．

$$\begin{aligned}c_{ij} &= c_{ij}^{(2m)} \cdot E^{2m} + c_{ij}^{(2m-1)} \cdot E^{2m-1} + \cdots + c_{ij}^{(1)} \cdot E + c_{ij}^{(0)}, \\ 0 &\le c_{ij}^{(k)} < E, \qquad k = 0,1,\cdots,2m, \quad i,j = 1,2,\cdots,n\end{aligned}$$

ただし，通常はゼロでない先頭の要素に正負の符号を持たす．

4.3 各方式による計算量

10 進 6 桁を単位として，入力の桁数がその m 倍の桁数の行列乗算を行う場合に m 倍精度と言う．筆算方式（定義方式）や中国剰余定理の場合は 1 ワードに 10 進 6 桁詰めて計算が可能であるが，FFT の場合は丸め誤差のため 1 ワードに 10 進 4 桁詰めて計算する．このため，FFT の場合は同じ桁数の計算をするのに 1.5 倍のワード数が必要とする．

4.3.1 計算量算出のための計算条件

$n \times n$ 次元行列で，入力は m ワード（FFT では $1.5m$ ワード），結果は $2m+1$ ワード（FFT では $3m+1$ ワード）で表現される多倍長精度の整数とする．ワード単位に分割した値は，32 ビット整数で保存し，演算及び演算途中の値の保存はその乗算，剰余，内積計算が正確に行えるように倍精度浮動小数点とする．このとき，下記のような計算処理に対して下記の様に計算量を定義する．筆算方式及び中国剰余定理では，1 ワードに 10 進 6 桁（$E = 10^6$）つめとし，FFT では 1 ワードに 10 進 4 桁（$E = 10^4$）つめとする．

(a) 要素単位の加減乗除 ： 1
(b) n 要素の内積計算 ： $2n$

(c)　　$(\mathrm{mod}\ p_k)$ 等の剰余計算： 4
(d)　実 FFT 計算（m 倍精度計算に $3m$ 要素が必要)：　$7.5m \cdot \log_2(3m)$
(e)　結果の桁上げ及び符合の判定：　他に比較して少ないので省略

4.3.2　各方式での計算量

筆算方式での計算量

m ワードに分割した多倍長精度の乗算が n^3 回必要である．1 回 m 倍長精度同士の乗算の計算量は $2m^2$ である．桁上げのため $2mn^2$ 回の　$(\mathrm{mod}\ E)$ 及び加算の処理が必要であるが，この部分を省略すると筆算方式による m 倍精度の値を係数にもつ $n \times n$ 次元行列の乗算の計算量 T_0 は次のようになる．

$$T_0 = 2m^2n^3$$

中国剰余定理での計算量

中国剰余定理による m 倍精度の値を係数にもつ $n \times n$ 次元行列の乗算の計算量を下記に示す．また，記号 $CRT(a_1, \cdots, a_m)$ は $a_1, \cdots, a_m$ に中国剰余定理を使用して，多数桁に戻す計算を示す．

(1)　$a_{ij}, b_{ij}\ \ (\mathrm{mod}\ p_k)$ の計算
計算量及び回数の内訳は書きのようになる．
　　$(\mathrm{mod}\ p_k)$ の 1 回の計算量 ： $2m$
　要素 a_{ij}, b_{ij} への適用回数　：$2n^2$
　適用する素数 p_k の個数　　：$2m+1$
従って，行列の全要素 $a_{ij}, b_{ij}\ \ (\mathrm{mod}\ p_k)$ の計算量 T_{11} は下記のようになる．

$$T_{11} = 4m(2m+1)n^2$$

(2)　$\theta_{ij}^{(k)} = \sum_{l=1}^{n} \alpha_{il}^{(k)} \cdot \beta_{lj}^{(k)}$ の計算
計算量及び回数の内訳は書きのようになる．
　1 回の内積計算の計算量 ： $2n$
　$\theta_{ij}^{(k)}$ の i, j の数 (行列要素) ： n^2
　$\theta_{ij}^{(k)}$ の k 数 (素数 p_k の数) ：$2m+1$
従って，全 $\theta_{ij}^{(k)}$ の計算量 T_{12} は次のようになる．

$$T_{12} = 2(2m+1)n^3$$

(3)　$c_{ij} = CRT(\theta_{ij}^{(1)}, \theta_{ij}^{(2)}, \cdots, \theta_{ij}^{(2m+1)})$ の計算

中国剰余定理 (CRT) を 1 回適用する計算量は式 (4.9) の v と c_{ij} の計算が主力となり，それぞれの計算量は下記の様になる．

$$v : \sum_{l=1}^{2m} 2l = 2m(2m+1)$$
$$c_{ij} : \sum_{l=1}^{2m} 6l = 6m(2m+1)$$

一方，CRT の適用回数は n^2 回である．従って，CRT による c_{ij} の計算の計算量 T_{13} は次のようになる．

$$T_{13} = 8m(2m+1)n^2$$

(4)　合計計算量

中国剰余定理による合計計算量は下記のようになる．

$$\begin{aligned} T_1 &= T_{11} + T_{12} + T_{13} = 4m(2m+1)n^2 + 2(2m+1)n^3 + 8m(2m+1)n^2 \\ &= 2(2m+1)(6m+n)n^2 \end{aligned}$$

FFT での計算量

中国剰余定理による m 倍精度の値を係数にもつ $n \times n$ 次元行列の乗算の計算量を下記に示す．FFT を使用した計算では誤差を考慮する必要があるため筆算方式や中国剰余定理の場合と異なり，1 要素に 10 進 6 桁ではなく 10 進 4 桁つめとする．そのため，入力が m 倍長精度の正確な計算に要素数 $3m$ の実 FFT を使用する．

(1)　実 FFT 計算 (要素数 $3m$)

複素 FFT の 1 要素当りの 1 ステップの計算量は 5 であり，実 FFT はその半分のため 2.5 となる．一方，多倍長精度を係数とする $n \times n$ 次元の行列乗算 $C = A \cdot B$ への FFT の適用回数は，行列 A, B に対する順変換と結果の行列 C に対する逆変換で，それぞれ n^2 回の合計 $3n^2$ 回となる．従って，要素数 $3m$ の実 FFT の計算量と行列乗算における使用回数は下記となる．

1 回の実 FFT の計算量：　$7.5m \cdot \log_2(3m)$

実 FFT の総適用回数 ：　$3n^2$

従って，実 FFT 計算の合計計算量 T_{21} は四捨五入し整数化して下記の様になる．

$$T_{21} = 23mn^2 \cdot \log_2(3m)$$

(2)　実 FFT 結果の内積計算

実 FFT の要素単位の乗算は共役複素数の乗算になるため複素数乗算の半分になる．このため，$3m$ 要素の内積計算の計算量は $12m$ で，行列要素全体の計算量 T_{22} は下記の様になる．

$$T_{22} = 12mn^3$$

(3)　合計計算量

FFT による合計計算量 T_2 は下記のようになる．

$$T_2 = T_{21} + T_{22} = 23mn^2 \cdot \log_2(3m) + 12mn^3 = mn^2(12n + 23\log_2(3m))$$

4.4　数値実験結果

多倍長精度を係数とする行列乗算 $C = A \cdot B$ の計算量及び計算時間の評価を行う．提案方式を評価するために，$n \times n$ 次元行列で倍精度浮動小数点に 10 進 6 桁を入れ，入力は m 倍精度で出力は $2m + 1$ 倍精度としたときの，計算量及び計算時間の比較評価を行う．1 倍精度は 10 進 6 桁とする．FFT 方式の場合は倍精度浮動小数点に 10 進 4 桁入れて計算するため，3 要素を 2 倍精度と換算して評価する．性能評価には下記の計算機を使用した．

(a) パソコン (Duron)

計算機： ADM Duron 850Mhz (850MFLOPS)

OS： Microsoft Windows Me

FORTRAN コンパイラー： Visual Fortran Standard Edition 5.0A

コンパイルオプション： Full Optimizations

図 4.1，図 4.2，図 4.3 に倍精度浮動小数点演算を使用した，同一次元の行列乗算における倍精度での計算時間を 1 とした比を示す．図 4.1 は 50 次元の行列で，図 4.2，図 4.3 は 100 次元及び 200 次元の行列に対する m 倍精度を入力値とする計算時間比を示す．◆印は筆算方式を，■印は中国剰余定理を，▲印は FFT 方式での計算方式を示す．図 4.4 は 200 次元の行列における理論計算量の比を示す．この場合も同一次元の行列乗算における倍精度での計算時間を 1 とした比を示す．図 4.5 に筆算方式の計算時間を 1 とした場合の，中国剰余定理と FFT 方式での効果比を示す．本図は効果比のため値が大きいほど高速なことを示す．これらの図から，中国剰余定理を使用した方式は 4 倍精度計算ですでに，筆算方式より高速で倍精度浮動小数点演算での行列乗算の約 10 倍の計算時間で実行できることが分かる．また，8 倍，12 倍及び 16 倍精度の計算は 4 倍精度での計算に比較して，それぞれ 2 倍，3 倍及び 4 倍で実行でき，桁数が短いうちはほぼ桁数に比例した計算時間となることが分かる．FFT 方式は，中国剰余定理に比較し，桁数が大きくなると有利で，その分かれ目は行列の次元数に比例して大きくなる．

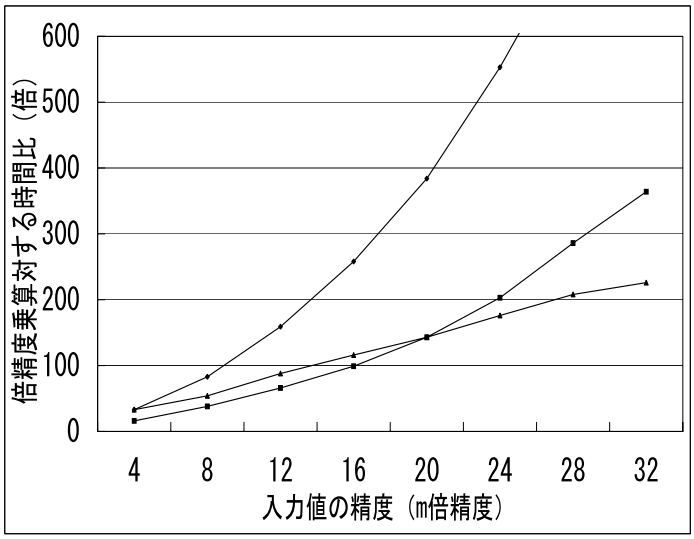

図 4.1: 倍精度計算との計算時間の比較（50 次元）

4.5 まとめ

多倍長精度の値を係数とする行列乗算に対して，2 方式の高速化方式を提案した．一つは中国剰余定理を利用する方法で，もう一つは FFT を利用する方式である．中国剰余定理による方法は，入力値が 10 進 6 桁を単位とする 4 倍精度ですでに，筆算方式より高速となり，比較的桁数が短いときに有功な方式である．一方，FFT 方式は長い桁数の行列乗算に有効な方式である．

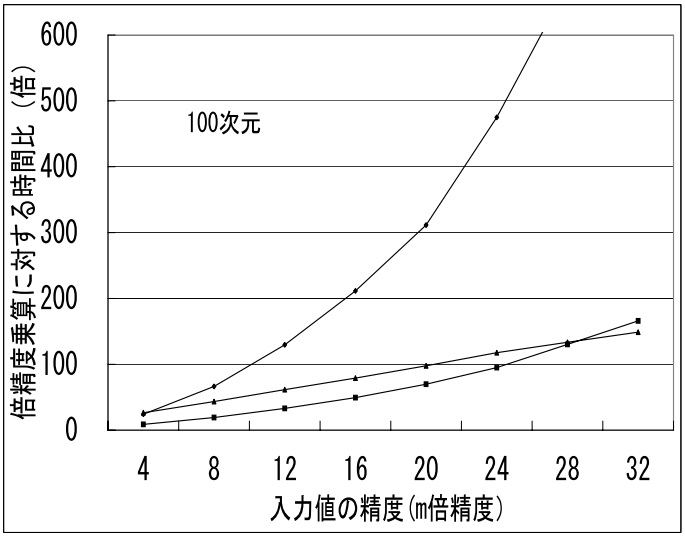

図 4.2: 倍精度計算との計算時間の比較（100 次元）

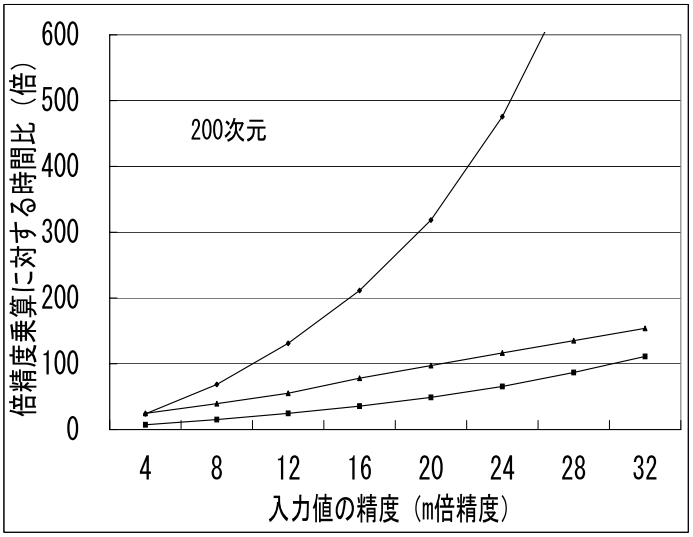

図 4.3: 倍精度計算との計算時間の比較（200 次元）

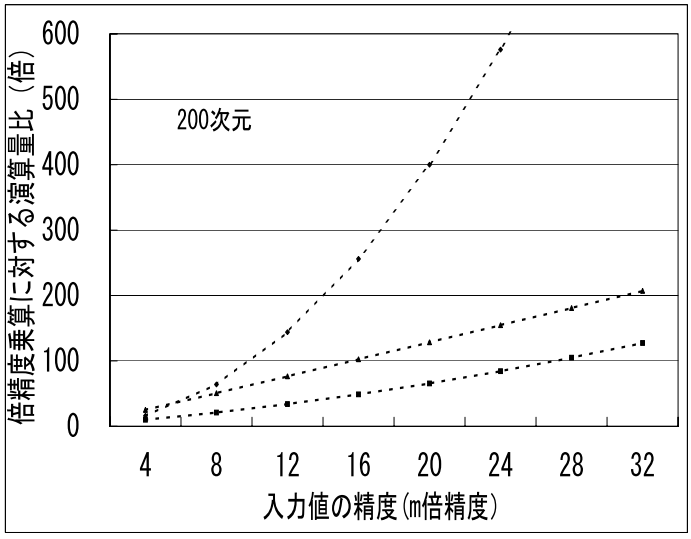

図 4.4: 倍精度計算との理論計算量の比較（200 次元）

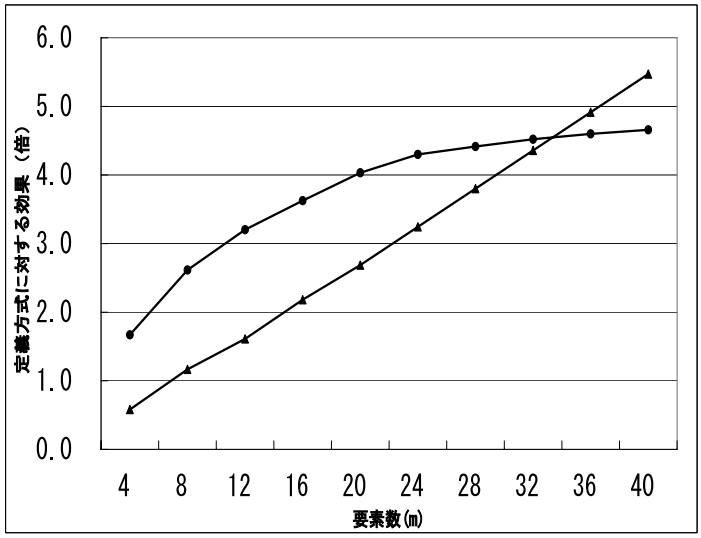

図 4.5: 筆算方式との計算時間の比較（100 次元）

第5章　多数桁の分割乗算

5.1　はじめに

n 桁の多数桁乗算は高速フーリエ変換 (FFT) を使用して，$O(n \log n(\log\log n))$ の計算量 [27] で計算できることが知られている．また，FFT による多数桁の乗算はメモリ量を多く使用するため，$Karatsuba$ 法 [30] や 2 段階 FMT[1] によるメモリ削減方法が利用されている．しかし，利用計算機のメモリ量を超えた多数桁の乗算を行うには，ファイル I/O を使用した分割乗算が必要になる．これまで，m 分割して FFT を適用すると計算量は分割前の m 倍 [17] になり，ファイル I/O を使用すると I/O 量は $O(m \cdot n)$ となリ，分割数 m が大きくなると効率が大幅に低下していた．これに対し,「高速剰余変換による多数桁乗算」[1] に，n 桁の m 分割した多数桁乗算の計算量を，分割しないときの計算量と同等にする分割乗算方式の原理が示されている．それは，拡張 FMT（Fast Modulo Transformation，高速剰余変換）の機能を利用し，FMT 変換を 2 段階に分けて行う方法である．本論文では，この分割乗算方式を具体的に示すと共に，直接編成のファイル I/O を利用して I/O 量を分割数 m に依存しない $O(n)$ にする方法を示す．本分割乗算の具体的計算方法として，複素 FMT の直接利用及び 2 段階 FMT への適用方法を示す．どちらの場合も n 桁の多数桁乗算で計算量は $O(n \cdot \log n \cdot (\log\log n))$ に，ファイル I/O 量は $O(n)$ になり，分割数 m に依存しない．更に，1 兆桁程度の超多数桁の乗算では，複素 FMT 方式では倍精度浮動小数点（64 ビット）に 10 進 2 桁詰めが限度で，2 段階 FMT では 64 ビット整数に 16 進 60 桁 (10 進 18 桁) 詰めの計算が可能であり，メモリ及びファイル使用量の面から 2 段階 FMT がより効率的である．本分割乗算の理論の裏付けをするため，ワークステーション (Itanium2) とパソコン (Duron) を使用し複素 FMT 及び 2 段階 FMT による分割乗算を実施した．FMT と FFT は多数桁乗算に適用すると同じ計算式となる，そのため FFT に整数を法とするものも含めると，ここで記載した FMT による分割乗算は，そのまま FFT でも適用可能である．本分割乗算を行うと，メモリ量に依存しないで超多数桁の乗算が可能になる．2002 年に達成した 10 進で 1.2 兆桁の円周率世界記録にはまだ本機能がなく苦しい経験をした．2001 年に開発したプログラムでは最大構成に近い 128 ノード (2T バイト) を使用しないと 1.2 兆桁の計算ができず，半分の 64 ノード (1T バイト) で実行可能とするため適用公式から変更したプログラムが必要になった．そのため，世界記録の達成が 1 年遅

れ 2002 年 11 月にづれ込んだ．次回の円周率世界記録では，本分割乗算機能を組み込み，使用ノード数 (メモリ量) に依存しないプログラムを開発する予定である．

5.2 高速剰余変換

「高速剰余変換による多数桁乗算」[1] に高速剰余変換 (FMT) の原理と多数桁の分割乗算の原理を示しているが，ここでは分割乗算に必要な拡張 FMT と分割乗算の計算式を記載する．n 個の素数 $P_1, P_2, \cdots, P_n$ を使用して各剰余から元の値を復元するのが中国剰余定理である．FMT では剰余 (mod) を整数だけでなく記号にも拡張し，記号 E と数 ω_n，k に対して $f \pmod{E-\omega_n^k}$ を多項式 f 中の記号 E を ω_n^k に置き換えることで定義する．更に，ω_n を FFT の基本原理である 1 の原始 n 乗根とし，素数の代わりに $E-\omega_n^k$ を利用する．順変換は多項式の剰余を求めることで，逆変換は剰余から元の多項式の係数を求めることで定義する．

5.2.1 基本 FMT 変換

$n-1$ 次の E の多項式 f に対して $E-\omega_n^k, k=0,1,\cdots,n-1$ の剰余に関する変換である．

順変換

多項式 $f \equiv a_{n-1}E^{n-1}+\cdots+a_1E+a_0$ に対して $f_k \equiv f \pmod{E-\omega_n^k}$ を求める．各 f_k は f 中の E に ω_n^k を代入して式 (5.1) が得られる．

$$\begin{aligned} f_k &= a_{n-1}\omega_n^{(n-1)k}+\cdots+a_1\omega_n^k+a_0 = \sum_{l=0}^{n-1} a_l\omega_n^{lk}, \\ & k=0,1,\cdots,n-1 \end{aligned} \tag{5.1}$$

逆変換

式 (5.1) の f_k から元の多項式 f の係数 a_j を求める．この計算には下記の ω_n が原始 n 乗根の条件を利用する．

$$\sum_{k=0}^{n-1}\omega_n^{(l-j)k} = \begin{cases} n: & l=j \\ 0: & l\neq j \end{cases} \tag{5.2}$$

式 (5.1) 及び式 (5.2) から次式が得られる．

$$\sum_{k=0}^{n-1} f_k \omega_n^{-kj} = \sum_{k=0}^{n-1}\sum_{l=0}^{n-1} a_l \omega_n^{kl}\omega_n^{-kj} = \sum_{l=0}^{n-1} a_l \sum_{k=0}^{n-1} \omega_n^{(l-j)k} = n a_j \tag{5.3}$$

従って，係数 a_j を求める逆変換は次式のように計算できる．

$$a_j = (f_{n-1}\omega_n^{-(n-1)j} + \cdots + f_1\omega_n^{-j} + f_0)/n = \sum_{k=0}^{n-1} f_k \omega_n^{-kj}/n, \quad j = 0, 1, \cdots, n-1 \tag{5.4}$$

5.2.2 拡張FMT変換

E の $n-1$ 次の多項式 f と整数 $m, s(s = n/m)$ 及び有理数 q に対して，ω_s を原始 s 乗根とするとき，$s-1$ 次の多項式 $f^{(q,s)} \equiv f \pmod{E^s - \omega_s^{qs}}$ を定義する．拡張FMT変換はこの $s-1$ 次の多項式 $f^{(q,s)}$ に対する $E - \omega_s^{k+q}, k = 0, 1, \cdots, s-1$ の剰余変換である．

順変換

多項式 $f^{(q,s)} \equiv f \pmod{E^s - \omega_s^{qs}} \equiv a_{s-1}^{(q,s)}E^{s-1} + \cdots + a_1^{(q,s)}E + a_0^{(q,s)}$ に対して $f_k^{(q,s)} \equiv f^{(q,s)} \pmod{E - \omega_s^{k+q}}$ を求める．各 $f_k^{(q,s)}$ は $f^{(q,s)}$ 中の E に数 ω_s^{k+q} を代入して式 (5.1) が得られる．

$$f_k^{(q,s)} = a_{s-1}^{(q,s)}\omega_s^{(s-1)(k+q)} + \cdots + a_1^{(q,s)}\omega_s^{k+q} + a_0^{(q,s)} = \sum_{l=0}^{s-1} a_l^{(q,s)}\omega_s^{(k+q)l}, \quad k = 0, 1, \cdots, s-1 \tag{5.5}$$

ただし，$a_l^{(q,s)} = a_l + a_{l+s}\omega_s^{qs} + \cdots + a_{l+(m-1)s}\omega_s^{(m-1)qs}$ である．

逆変換

$f_k^{(q,s)}$ から $s-1$ 次の多項式 $f^{(q,s)}$ の係数 $a_j^{(q,s)}$ を求める．式 (5.5) 及び ω_s が 1 の原始 s 乗根から次式が得られる．

$$\begin{aligned}\sum_{k=0}^{s-1} f_k^{(q,s)}\omega_s^{-kj} &= \sum_{k=0}^{s-1}\sum_{l=0}^{s-1} a_l^{(q,s)}\omega_s^{(k+q)l}\omega_s^{-kj} \\ &= \sum_{l=0}^{s-1} a_l^{(q,s)}\omega_s^{ql}\sum_{k=0}^{s-1}\omega_s^{(l-j)k} = s\omega_s^{qj}a_j^{(q,s)}\end{aligned} \tag{5.6}$$

従って，係数 $a_j^{(q,s)}$ を求める逆変換は次式のように計算できる．

$$\begin{aligned} a_j^{(q,s)} &= (f_{s-1}^{(q,s)}\omega_s^{-(s-1)j} + \cdots + f_1^{(q,s)}\omega_s^{-j} + f_0^{(q,s)})\omega_s^{-qj}/s \\ &= \omega_s^{-qj}\sum_{k=0}^{s-1} f_k^{(q,s)}\omega_s^{-kj}/s, \qquad j = 0, 1, \cdots, s-1 \end{aligned} \tag{5.7}$$

5.2.3 拡張FMT変換と畳込み演算の関係式

次式の f, g で示す E の $n-1$ 次多項式の係数の畳込み演算で作成される E の $2n-1$ 次の多項式を h とする．

$$\begin{aligned} f &\equiv a_{n-1}E^{n-1} + \cdots + a_1E + a_0 \\ g &\equiv b_{n-1}E^{n-1} + \cdots + b_1E + b_0 \\ h &\equiv f \cdot g \equiv c_{2n-1}E^{2n-1} + \cdots + c_1E + c_0 \end{aligned} \tag{5.8}$$

このとき，h の係数 c_j は f, g の係数 a_i, b_{j-i} で次式のように表わされる．

$$c_j = \sum_{i=max(0,j-n+1)}^{min(j,n-1)} a_i b_{j-i}, \qquad j = 0, 1, \cdots, 2n-1 \tag{5.9}$$

式 (5.8) の $n-1$ 次の多項式 f, g を E の $2n-1$ 次ではなく整数 $m, s(s = n/m)$ 及び有理数 q に対して，ω_s を原始 s 乗根として $s-1$ 次に畳込むことを考える．

$$\begin{aligned} h^{(q,s)} &\equiv h \pmod{E^s - \omega_s^{sq}} \equiv f \cdot g \pmod{E^s - \omega_s^{sq}} \\ &\equiv c_{s-1}^{(q,s)}E^{s-1} + \cdots + c_1^{(q,s)}E + c_0^{(q,s)} \end{aligned} \tag{5.10}$$

原始 s 乗根 ω_s を用いて次式のように計算する．
ここで，$k = 0, 1, \cdots, s-1$，$j = 0, 1, \cdots, s-1$ である．

(1) $s-1$ 次の多項式 $f^{(q,s)}, g^{(q,s)}$ に変更する．

$$\begin{aligned} f^{(q,s)} &\equiv f \pmod{E^s - \omega_s^{sq}} = a_{s-1}^{(q,s)}E^{(s-1)} + \cdots + a_1^{(q,s)}E + a_0^{(q,s)} \\ g^{(q,s)} &\equiv g \pmod{E^s - \omega_s^{sq}} = b_{s-1}^{(q,s)}E^{(s-1)} + \cdots + b_1^{(q,s)}E + b_0^{(q,s)} \end{aligned} \tag{5.11}$$

ここで，$a_l^{(q,s)} = a_l + a_{l+s}\omega_s^{qs} + \cdots + a_{l+(m-1)s}\omega_s^{(m-1)qs}$ で
$b_l^{(q,s)} = b_l + b_{l+s}\omega_s^{qs} + \cdots + b_{l+(m-1)s}\omega_s^{(m-1)qs}$，　$l = 0, 1, \cdots, s-1$ である．

(2) 拡張FMT順変換で $f_k^{(q,s)}, g_k^{(q,s)}$ を計算する．

$$\begin{aligned} f_k^{(q,s)} &\equiv f^{(q,s)} \pmod{E - \omega_s^{k+q}} \\ &= a_{s-1}^{(q,s)}\omega_s^{(s-1)(k+q)} + \cdots + a_1^{(q,s)}\omega_s^{k+q} + a_0^{(q,s)} \\ g_k^{(q,s)} &\equiv g^{(q,s)} \pmod{E - \omega_s^{k+q}} \\ &= b_{s-1}^{(q,s)}\omega_s^{(s-1)(k+q)} + \cdots + b_1^{(q,s)}\omega_s^{k+q} + b_0^{(q,s)} \end{aligned} \tag{5.12}$$

(3) 各 k 毎に $f_k^{(q,s)}$ と $g_k^{(q,s)}$ の乗算を次式で計算する．

$$h_k^{(q,s)} = f_k^{(q,s)} \cdot g_k^{(q,s)} \tag{5.13}$$

(4) 拡張 FMT 逆変換で $h_k^{(q,s)}$ から多項式 $h^{(q,s)}$ の係数 $c_j^{(q,s)}$ を求める．

$$c_j^{(q,s)} = \left(h_{s-1}^{(q,s)}\omega_s^{-(s-1)j} + \cdots + h_1^{(q,s)}\omega_s^{-j} + h_0^{(q,s)}\right)\omega_s^{-qj}/s \tag{5.14}$$

式 (5.9) の c_j と式 (5.14) の $c_j^{(q,s)}$ の関係は，式 (5.8) と式 (5.10) の定義から次式のようになる．

$$\begin{aligned} c_j^{(q,s)} &= c_j + c_{j+s}\omega_s^{sq} + \cdots + c_{j+(2m-1)s}\omega_s^{(2m-1)sq}, \\ & \qquad j = 0, 1, \cdots, s-1 \end{aligned} \tag{5.15}$$

5.2.4 FMT の計算方法と FFT の関係

基本 FMT 変換と FFT との関係

n 要素の FFT は ω_n を $e^{-2\pi i/n}$ とした変換であり，FMT 変換は ω_n を 1 の原始 n 乗根にした変換である．このため，ω_n を複素数とすれば両者は同一で下記の計算式となる．l 段目の順変換を $n = 2^M$ の例で示すと，FMT と FFT の計算式は共に下記のようになる．

$$\begin{aligned} F_{j,k,0}^{(l+1)} &= F_{j,0,k}^{(l)} + F_{j,1,k}^{(l)} \cdot \omega_n^{jkt}, \\ F_{j,k,1}^{(l+1)} &= F_{j,0,k}^{(l)} - F_{j,1,k}^{(l)} \cdot \omega_n^{jkt}, \\ & \qquad j = 0, 1, \cdots, t-1, \quad k = 0, 1, \cdots, 2^l - 1, \\ & \qquad l = 0, 1, \cdots, M-1, \quad t = 2^{M-l-1} \end{aligned} \tag{5.16}$$

ここで，$F_{j,0,0}^{(0)}, F_{j,1,0}^{(0)}, (j = 0, 1, \cdots, n/2-1)$ は入力値で式 (5.1) の記号を使用すると，$F_{j,0,0}^{(0)} = a_j, F_{j,1,0}^{(0)} = a_{j+n/2}$ となる．また，$F_{j,0,0}^{(0)} = a_{n-j-1}, F_{j,1,0}^{(0)} = a_{n/2-j-1}$ と対応させることも可能であり，この時は，式 (5.16) をそれに合わせて変更する必要がある．逆変換は ω_n を ω_n^{-1} に変更し，最終段で $1/n$ を各要素に乗算する．

拡張 FMT 変換と基本 FMT 変換の関係

式 (5.1) の基本 FMT 変換 f_k と，式 (5.5) の拡張 FMT 変換で要素数 s を n に代えた $f_k^{(q,n)}$ の関係を調べる．要素数を共に n にしたので，ω_n は 1 の原始 n 乗根となる．ここで，q は有理数である．まず，順変換における関係は下記のようになる．

$$
\begin{aligned}
f_k &= \sum_{l=0}^{n-1} a_l \omega_n^{lk}, \\
f_k^{(q,n)} &= \sum_{l=0}^{n-1} a_l \omega_n^{(k+q)l} = \sum_{l=0}^{n-1} (a_l \omega_n^{ql}) \omega_n^{lk}, \\
&\qquad k = 0, 1, \cdots, n-1
\end{aligned} \tag{5.17}
$$

即ち，拡張順FMT変換は入力の各要素 a_l に対して，ω_n^{ql} を乗算した $a_l \omega_n^{ql}$ に基本順FMT変換をしたものに対応する．次に，逆変換における関係は下記のようになる．

$$
\begin{aligned}
a_j^{(q,n)} &= \omega_n^{-qj} \sum_{k=0}^{n-1} f_k \omega_n^{-kj}/n = \omega_n^{-qj} a_j, \\
&\qquad j = 0, 1, \cdots, n-1
\end{aligned} \tag{5.18}
$$

即ち，拡張逆FMT変換は基本逆変換結果の各要素 a_j に対して，ω_n^{-qj} を乗算した $a_j \omega_n^{-qj}$ で計算される．

5.3 多数桁分割乗算の原理

多数桁の分割乗算の目的は，計算量を増加させないでファイルI/Oを使用し，使用メモリ量を削減することである．FFT変換又は基本FMT変換を m 分割してファイルに保存し，分割単位で乗算を行うと，計算量は m 倍に増加し，ファイルI/O量も $O(m \cdot n)$ となる．そこで，ここでは拡張FMTを利用して $O(n)$ の入出力量で，計算量も分割前と同等の多数桁乗算の計算原理を示す．

5.3.1 多数桁の分割乗算原理

式 (5.15) より，要素数 $s = n/m$ の $2m$ 個の異なる q における畳み込み演算結果 $c_j^{(q,s)}$ から，式 (5.9) の元の $2n$ 要素の畳み込み演算結果 c_j が得られることが分る．式 (5.15) で $s = n/m$ 及び $q = k/2m, (k = 0, 1, \cdots, 2m-1)$ を代入すると次式が得られる．ここで，ω_{2m} は原始 $2m$ 乗根である．

$$
\begin{aligned}
c_j^{(k/2m,s)} &= c_j + c_{j+s} \omega_{2m}^{ks} + \cdots + c_{j+(2m-1)s} \omega_{2m}^{(2m-1)ks}, \\
&\qquad j = 0, 1, \cdots, s-1, \quad k = 0, 1, \cdots, 2m-1
\end{aligned} \tag{5.19}
$$

この式から逆に $2m$ 分割したときの元の $2n-1$ 次の係数は次式のようになる．

$$\begin{aligned} c_{j+ks} &= \sum_{l=0}^{2m-1} c_j^{(l/2m,s)} \omega_{2m}^{-lks}/2m, \\ & \qquad j=0,1,\cdots,s-1, \quad k=0,1,\cdots,2m-1 \end{aligned} \tag{5.20}$$

5.3.2 分割乗算の計算方法

式 (5.11) から式 (5.15) 及び式 (5.19) を使用して，式 (5.8) の $f \cdot g$ の具体的な計算方法を示す．ここで ω_{2n}，ω_s，ω_{2m} はそれぞれ 1 の原始 $2n$，s 及び $2m$ 乗根である．分割数は $2m$ で $s=n/m$ とする．また，係数の表示を見やすくするため $a_j^{(k/2m,s)}$ 等と表示すべき右肩の $(k/2m,s)$ を (k) と省略して $a_j^{(k)}$ と表示する．

(1) 入力 $a_i, b_i, (i=0,1,\cdots,n-1)$ を $2m$ 個に分割する計算

各 j に対して下記の $2m$ 要素の基本 FMT 順変換を行なう．

$$\begin{aligned} a_j^{(k)} &= \sum_{l=0}^{m-1} a_{l+mj}\omega_{2m}^{kl}, \qquad b_j^{(k)} = \sum_{l=0}^{m-1} b_{l+mj}\omega_{2m}^{kl}, \\ & \qquad j=0,1,\cdots,s-1, \quad k=0,1,\cdots,2m-1 \end{aligned} \tag{5.21}$$

この変換の特徴は乗算する ω_{2m} のベキ数が添え字 j に依存しないことである．

(2) $a_j^{(k)}$ と $b_j^{(k)}$ を入力し拡張 FMT 順変換

$\omega_{2n}^{2m}=\omega_s$ の関係を利用すると，拡張 FMT 順変換は下記の様になる．

$$\begin{aligned} f_j^{(k)} &= \sum_{l=0}^{s-1} a_l^{(k)}\omega_{2n}^{(2mj+k)l} = \sum_{l=0}^{s-1}(a_l^{(k)}\omega_{2n}^{kl})\omega_s^{jl}, \\ g_j^{(k)} &= \sum_{l=0}^{s-1} b_l^{(k)}\omega_{2n}^{(2mj+k)l} = \sum_{l=0}^{s-1}(b_l^{(k)}\omega_{2n}^{kl})\omega_s^{jl}, \\ & \qquad j=0,1,\cdots,s-1, \quad k=0,1,\cdots,2m-1 \end{aligned} \tag{5.22}$$

この拡張 FMT 順変換は各入力 $a_j^{(k)}, b_j^{(k)}, (j=0,1,\cdots,s-1)$ に ω_{2n}^{kl} を乗算した，$2m$ 組の s 要素の基本 FMT 順変換に対応する．

(3) 各 k 及び j 毎に $f_j^{(k)}$ と $g_j^{(k)}$ の乗算で $h_j^{(k)}$ を計算

各 j,k に対して下記のように項別の乗算をする．

$$\begin{aligned} h_j^{(k)} &= f_j^{(k)} \cdot g_j^{(k)}, \\ & \qquad j=0,1,\cdots,s-1, \quad k=0,1,\cdots,2m-1 \end{aligned} \tag{5.23}$$

(4) 拡張 FMT 逆変換で $h_j^{(k)}$ から多項式 $h^{(k)}$ の係数 $c_j^{(k)}$ を求める．
$\omega_{2n}^{-2m} = \omega_s^{-1}$ の関係を利用すると，拡張 FMT 逆変換は下記の様になる．

$$c_j^{(k)} = \sum_{l=0}^{s-1} h_l^{(k)} \omega_{2n}^{-(2ml+k)j}/s = (\sum_{l=0}^{s-1} h_l^{(k)} \omega_s^{-jl}) \omega_{2n}^{-kj}/s, \quad j = 0, 1, \cdots, s-1, \quad k = 0, 1, \cdots, 2m-1 \tag{5.24}$$

この拡張 FMT 逆変換は基本 FMT 逆変換の結果の各要素に ω_{2n}^{-kj} を乗算したことに等しい．

(5) $c_j^{(k)}$ から元の乗算結果の係数 $c_i, (i = 0, 1, \cdots, 2n-1)$ を復元
各 j に対して下記の $2m$ 要素の基本 FMT 順変換を行なう．

$$c_{j+ks} = \sum_{l=0}^{2m-1} c_j^{(l)} \omega_{2m}^{-kl}/(2m), \quad j = 0, 1, \cdots, s-1, \quad k = 0, 1, \cdots, 2m-1 \tag{5.25}$$

この変換の特徴は乗算する ω_{2m} のベキ数が添え字 j に依存しないことである．

5.3.3 ファイル I/O を利用した計算方法

ファイル I/O を使用した多数桁の分割乗算の計算方法を示す．整数 FMT による分割乗算の場合はこの計算方法が直接利用できる．$C = A \cdot B$ の多数桁乗算において，入力 A はファイル f_1 に，入力 B はファイル f_2 に記憶し，結果 C はファイル f_3 に求める．多数桁の乗算結果 C の各要素は畳込み演算の結果で，多数桁として正規化するには桁上げ処理が必要である．入力 A 及び B の要素数を n とし，結果 C の要素数を $2n$ とする．計算用のワークファイルとして，ファイル f_3 と f_4 を使用する．分割数を $2m$ とし，n は $n = n_1 n_2 m$ と分解できるとする．このとき，各ファイルのレコード長は n_1 個の要素で構成する．一方，レコード数はファイル f_1 と f_2 が $n_2 m$ で，ファイル f_3 と f_4 は $2n_2 m$ の直接探査ファイルである．計算用配列として $n_1 \cdot max(n_2, 2m)$ のサイズの A 及び B を使用する．そのため，同じ分割数 $2m$ では $n_2 = 2m$ のときにメモリ使用量が最小となる．計算用配列 A, B は共に 1 次元配列を使用する．変数 A, B はメモリ上の配列を意味し，$A_j = A_j \omega_p^j$ は A_j と ω_p^j を乗算た結果を A_j に入れることを意味する．$read(f_i, key = j)$ はファイル f_i から j 番目のレコードを，その後に記載された配列に読み込むことを意味する．$write(f_i, key = j)$ は同様に書き込むことを意味する．ω_p は 1 の原始 p 乗根とする．記載した FMT の計算式は，原理計算式であり高速化のために式 (5.16) の様に計算する必要がある．

(1) 入力 A, B を $2m$ 個に分割する計算
入力データ A, B をファイル f_1, f_2 から非連続に読み取り，前半にゼロを付けて基本

FMT 変換を行い，結果をファイル f_3, f_4 に非連続に書き込む．下記に入力データ A の処理を示す．下記の計算を $j = 0, 1, \cdots, n_2 - 1$ で n_2 回行う．ここで，ω_{2m} は 1 の原始 $2m$ 乗根でである．

$$
\begin{aligned}
&read(f_1, key = j + n_2 k + 1)\ A_{(k+m)n_1+i}, \quad A_{kn_1+i} = 0, \\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, m - 1 \\
&A_{kn_1+i} = \sum_{l=0}^{2m-1} A_{ln_1+i}\omega_{2m}^{kl}, \quad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, 2m - 1 \\
&write(f_3, key = j + n_2 k + 1)\ A_{kn_1+i}, \\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, 2m - 1
\end{aligned} \tag{5.26}
$$

入力データ B の処理は，上記に対してファイル f_1 及び f_3 をそれぞれ f_2 及び f_4 に変更する．

(2) 拡張 FMT 変換による各分割単位の乗算

データ A, B を変換したファイル f_3, f_4 を連続に入力し，拡張 FMT 変換を利用した分割単位の乗算を行い，結果をファイル f_3 に連続に出力する．下記にその処理を示す．下記の計算を $k = 0, 1, \cdots, 2m - 1$ まで $2m$ 回行う．ここで，$s = n_1 n_2$ で，ω_{2n}，ω_s はそれぞれ 1 の原始 $2n$ 乗根及び s 乗根である．

$$
\begin{aligned}
&read(f_3, key = j + n_2 k + 1)\ A_{n_1 j+i}, \quad i = 0, 1, \cdots, n_1 - 1, \\
&read(f_4, key = j + n_2 k + 1)\ B_{n_1 j+i}, \quad i = 0, 1, \cdots, n_1 - 1, \\
&\qquad j = 0, 1, \cdots, n_2 - 1 \\
&A_j = A_j\omega_{2n}^{jk}, \quad B_j = B_j\omega_{2n}^{jk}, \quad j = 0, 1, \cdots, s - 1 \\
&A_j = \sum_{l=0}^{s-1} A_l\omega_s^{jl}, \quad j = 0, 1, \cdots, s - 1 \\
&B_j = \sum_{l=0}^{s-1} B_l\omega_s^{jl}, \quad j = 0, 1, \cdots, s - 1 \\
&A_j = A_j \cdot B_j, \quad j = 0, 1, \cdots, s - 1 \\
&A_j = \sum_{l=0}^{s-1} A_l\omega_s^{-jl}, \quad j = 0, 1, \cdots, s - 1 \\
&A_j = A_j\omega_{2n}^{-jk}, \quad j = 0, 1, \cdots, s - 1 \\
&write(f_3, key = j + n_2 k + 1)\ A_{n_1 j+i}, \quad i = 0, 1, \cdots, n_1 - 1, \\
&\qquad j = 0, 1, \cdots, n_2 - 1
\end{aligned} \tag{5.27}
$$

(3) $2m$ 個の分割結果から多数桁乗算の復元

$2m$ 個の分割乗算結果のあるファイル f_3 を非連続に読み取り，多数桁乗算結果をファイル f_3 に上書きする．下記の計算を $j = 0, 1, \cdots, n_2 - 1$ で n_2 回行う．ここで，ω_{2m} は 1 の原始 $2m$ 乗根である．

$$
\begin{aligned}
&read(f_3, key = j + n_2 k + 1)\ \ A_{kn_1+i}, \\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, 2m - 1 \\
&A_{kn_1+i} = \sum_{l=0}^{2m-1} A_{ln_1+i}\omega_{2m}^{-kl}/n, \quad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, 2m - 1 \\
&write(f_3, key = j + n_2 k + 1)\ \ A_{kn_1+i}, \\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, 2m - 1
\end{aligned}
\tag{5.28}
$$

5.4 複素 FMT による分割乗算

複素 FMT を使用して，複素結果の実部が多数桁の下位部に，虚部が多数桁の上位部になる変換を分割乗算に適用して，多数桁の乗算を行う．複素 FMT 変換を使用するため，要素数は実変換の半分，入力の要素数と同じになる．式 (5.15) に $q = 1/4$ 及び $s = n, m = 1$ を代入すると，ω_n は 1 の原始 n 乗根で，$\omega_n^{n/4} = i$ となるため次式が得られる．ここで，i は虚数単位である．

$$
c_j^{(1/4,n)} \quad = \quad c_j + i \cdot c_{j+n}, \qquad j = 0, 1, \cdots, n - 1 \tag{5.29}
$$

本式により，整数値 a_j, b_j を入力し，$\omega_n^{j/4}$ を乗算して複素数 $a_j\omega_n^{j/4}, b_j\omega_n^{j/4}$ に変換し，複素数による分割乗算を行い，結果の実部を元の多数桁の下位部に虚部を上位部に対応させると，多数桁の分割乗算となる．$C = A \cdot B$ の多数桁乗算において，入力 A はファイル f_1 に，入力 B はファイル f_2 に記憶し，結果 C はファイル f_3 に記憶する．多数桁の乗算結果 C の各要素は畳込み演算の結果で，多数桁として正規化するには桁上げ処理と丸め誤差を除いて整数化する必要がある．入力 A 及び B の要素数を n とし，結果 C の要素数は $2n$ でである．計算用のワークファイルとして，ファイル f_3 と f_4 を使用する．分割数を m とし，n は $n = n_1 n_2 m$ と分解できるとする．このとき，各ファイルのレコード長は n_1 個の要素で構成する．入力と出力は実数 (整数) で，途中は複素数で計算する．一方，レコード数は $n_2 m$ 及び $2n_2 m$ の直接探査ファイルである．計算用配列として $n_1 \cdot max(n_2, m)$ のサイズの複素配列 C 及び Z を使用する．また，I/O 用の配列として n_1 のサイズの実配列 A を使用する．そのため，同じ分割数 m では $n_2 = m$ のときにメモリ使用量が最小となる．配列 A, C, Z は共に 1 次元配列を使用する．変数

A, C, Z はメモリ上の配列を意味し，$C_j = A_j\omega_n^j$ は実数 A_j と複素数 ω_n^j を乗算した複素結果を C_j に入れることを意味する．$read(f_i, key = j)$ はファイル f_i から j 番目のレコードを，その後に記載された配列に読み込むことを意味する．$write(f_i, key = j)$ は同様に書き込むことを意味する．ω_n は 1 の原始 n 乗根で複素数，即ち i を虚数単位として $\omega_n = e^{-2\pi i/n}$ である．記載した FMT の計算式は，原理計算式であり高速化のために式 (5.16) の様に計算する必要がある．

(1) 入力 A, B を m 個に分割する計算

入力データ A, B をファイル f_1, f_2 から非連続に読み取り，$j = 0, 1, \cdots, n-1$ 番目の要素に $\omega_n^{j/4}$ を乗算し，複素数 C に変換して基本複素 FMT 変換を行い結果をファイル f_3, f_4 に非連続に書き込む．下記に入力データ A の処理を示す．下記の計算を $j = 0, 1, \cdots, n_2-1$ で n_2 回行う．ここで，ω_m は 1 の原始 m 乗根で $\omega_m = \omega_n^{n_1 n_2}$ である．

$$
\begin{aligned}
&read(f_1, key = j + n_2k + 1)\ A_i, \qquad C_{kn_1+i} = A_i\omega_n^{((j+n_2k)n_1+i)/4},\\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, m - 1\\
&C_{kn_1+i} = \sum_{l=0}^{m-1} C_{ln_1+i}\omega_m^{kl}, \quad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, m - 1\\
&write(f_3, key = j + n_2k + 1)\ C_{kn_1+i},\\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, m - 1
\end{aligned}
\tag{5.30}
$$

入力データ B の処理は，上記に対してファイル f_1 及び f_3 をそれぞれ f_2 及び f_4 に変更する．

(2) 拡張 FMT 変換による各分割単位の乗算

データ A, B を変換したファイル f_3, f_4 を連続に入力し，拡張 FMT 変換を利用した分割単位の乗算を行い，結果をファイル f_4 に連続に出力する．下記にその処理を示す．下記の計算を $k = 0, 1, \cdots, m-1$ まで m 回行う．ここで，$s = n_1n_2$ で，ω_s は 1 の原始 s 乗根となる複素数で $\omega_s = \omega_n^m$ である．

$$
\begin{aligned}
&read(f_3, key = j + n_2k + 1)\ C_{n_1j+i}, \quad i = 0, 1, \cdots, n_1 - 1,\\
&read(f_4, key = j + n_2k + 1)\ Z_{n_1j+i}, \quad i = 0, 1, \cdots, n_1 - 1,\\
&\qquad j = 0, 1, \cdots, n_2 - 1\\
&C_j = C_j\omega_n^{jk}, \quad Z_j = Z_j\omega_n^{jk}, \quad j = 0, 1, \cdots, s - 1\\
&C_j = \sum_{l=0}^{s-1} C_l\omega_s^{jl}, \quad j = 0, 1, \cdots, s - 1\\
&Z_j = \sum_{l=0}^{s-1} Z_l\omega_s^{jl}, \quad j = 0, 1, \cdots, s - 1
\end{aligned}
$$

$$
\begin{aligned}
&C_j = C_j \cdot Z_j, \quad j = 0, 1, \cdots, s-1 \\
&C_j = \sum_{l=0}^{s-1} C_l \omega_s^{-jl}, \quad j = 0, 1, \cdots, s-1 \\
&C_j = C_j \omega_n^{-jk}, \quad j = 0, 1, \cdots, s-1 \\
&write(f_4, key = j + n_2 k + 1)\ C_{n_1 j+i}, \quad i = 0, 1, \cdots, n_1 - 1, \\
&\qquad j = 0, 1, \cdots, n_2 - 1
\end{aligned} \tag{5.31}
$$

(3) m 個の複素分割結果から多数桁乗算の復元

m 個の複素分割乗算結果のあるファイル f_4 を非連続に読み取り，基本複素 FMT 逆変換を行った後，$j = 0, 1, \cdots, n-1$ 番目の要素に $\omega_n^{-j/4}$ を乗算し，虚部を多数桁乗算結果の上位桁，実部を下位桁としファイル f_3 に書き込む．下記の計算を $j = 0, 1, \cdots, n_2 - 1$ で n_2 回行う．ここで，ω_m は 1 の原始 m 乗根となる複素数である．$real(\ )$ 及び $imag(\ )$ はそれぞれ $(\ )$ 内の複素数の実部及び虚部を意味する．

$$
\begin{aligned}
&read(f_4, key = j + n_2 k + 1)\ C_{kn_1+i}, \\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, m-1 \\
&C_{kn_1+i} = \sum_{l=0}^{m-1} C_{ln_1+i} \omega_m^{-kl}/n, \quad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, m-1 \\
&C_{kn_1+i} = C_{kn_1+i} \omega_n^{-((j+n_2 k)n_1+i)/4} \\
&A_i = real(C_{kn_1+i}), \qquad write(f_3, key = j + n_2 k + 1)\ A_i, \\
&A_i = imag(C_{kn_1+i}), \qquad write(f_3, key = j + n_2(m+k) + 1)\ A_i, \\
&\qquad i = 0, 1, \cdots, n_1 - 1, \quad k = 0, 1, \cdots, m-1
\end{aligned} \tag{5.32}
$$

5.5 2段階FMTによる分割乗算

2 段階 FMT による乗算とは，1 要素に多数桁を持つ $2N$ 要素の乗算を $P = \omega_{2N}^N + 1$ を法とする整数 FMT で行い，そこで発生する各要素ごとの負巡回乗算にも FMT を使用する方法である．P を法とする整数 FMT を上位 FMT，各要素ごとの乗算に使用する FMT を下位 FMT と呼ぶ．分割して計算するため，上位 FMT は m 個に分割する．下位 FMT は種々の計算方法があるが，ここでは $2n$ 要素の実数の負巡回乗算が可能な n 要素の複素拡張 FMT を使用する．また，上位 FMT と下位 FMT の要素数をそれぞれ $2N$ 及び n(実数にすると $2n$ 要素) とする．上位 FMT は整数演算のため，32 ビット整数または 64 ビット整数を使用し，下位 FMT は複素数演算のため倍精度浮動小数点 (8 バイト) を使用する．下位 FMT の 1 要素 (倍精度浮動小数点) に α 進の l 桁を詰め，上位 FMT の整数 1 個に h 倍の α 進 lh 桁詰める．上位 FFT の基底 ω_{2N} は整数 Ng 個

で構成され，ω_{2N} は整数 P を法とする 1 の原始 $2N$ 乗根で，$\omega_{2N} = \alpha^{glh}$ とする．また，上位 FMT の各要素の先頭 2 個の整数 (入力及び最終結果では 1 個の整数) は桁上げ (下位乗算の結果が P 以内になることを保証) のため空けておく必要がある．このとき，上位 FMT の 1 要素は Ng 個の整数で構成され，$n{=}Ngh/2$ の関係がある．この乗算では，m 分割して α 進 $(Ng/2-1)Nlh$ 桁 ($N^2g/2$ 個の整数) を与え，$(Ng-1)Nlh$ 桁 (N^2g 個の整数) の結果を得る．上位 FMT は $2N^2g$ 個の整数で行う．全体を m 分割して乗算するため，上位 FMT 変換は m 分割する変換と分割後の変換の 2 段階に分ける．記号 F は上位 FMT 用の整数配列を，c 及び z は下位 FMT 用の複素数配列を示す．また，式中での等号は右辺の計算値を左辺の配列に代入することを示す．記号 F で示す整数配列はそれぞれ α 進 lh 桁を超えないように，計算結果を桁上げ処理をして正規化する必要がある．FMT 変換の計算式は拡張 FMT 変換の場合も，入力及び結果に適当な変換をして基本 FMT の計算式として示した．記載した FMT の計算式は，原理計算式であり高速化のために式 (5.16) の様に計算する必要がある．

5.5.1 上位順 FMT 変換

上位 FMT の要素数 $2N$ を m 分割し，$n_2 = 2N/m$ とする．上位 FMT は m 個に分割する FMT と，分割後の FMT の 2 段に分ける．m 個に分割する処理は m 要素の基本 FMT を n_2 回行い，分割後の処理は n_2 要素の拡張 FMT を m 回行う．

(1) 入力 A, B を m 個に分割する計算

F は Nmg 個の整数配列で，g 個の整数配列が下付き添え字の 1 に対応する．ω_m は 1 の原始 m 乗根で $\omega_m = \omega_{2N}^{n_2} = \alpha^{n_2glh}$ である．入力データ A の処理は下記の計算を $j = 0, 1, \cdots, n_2-1$ で n_2 回行う．

$$
\begin{aligned}
&read(f_1, key = j + n_2k + 1)\ F_{(k+1/2)N+i}, \qquad i = 0, 1, \cdots, N/2-1,\\
&F_{(kN+i)} = 0, \qquad i = 0, 1, \cdots, N/2-1,\\
&F_{(k+m/2)N+i} = 0, \qquad i = 0, 1, \cdots, N-1,\\
&\qquad k = 0, 1, \cdots, m/2-1\\
&F_{kN+i} = \sum_{t=0}^{m-1} F_{tN+i}\omega_m^{kt},\\
&\qquad i = 0, 1, \cdots, N-1, \quad k = 0, 1, \cdots, m-1\\
&write(f_3, key = j + n_2k + 1)\ F_{kN+i}\\
&\qquad i = 0, 1, \cdots, N-1, \quad k = 0, 1, \cdots, m-1
\end{aligned}
\tag{5.33}
$$

入力データ B の処理は，上記に対してファイル f_1 及び f_3 をそれぞれ f_2 及び f_4 に変更する．F_i と ω_m^k の乗算は，F_i の内容を $F_{i+kn_2 \pmod N}$ の位置に移せば良い．

(2) 分割単位の上位順FMT変換

データ A のFMTは下記の計算を $k=0,1,\cdots,m-1$ まで m 回行う．この変換には拡張FMTを使用し，拡張FMTは入力 F_i に ω_{2N}^{ik} を乗算して要素数 n_2 の基本FMT を実行することに対応する．ω_{n_2} は1の原始 n_2 乗根で $\omega_{n_2}=\omega_{2N}^{m}=\alpha^{mglh}$ である．

$$
\begin{aligned}
&read(f_3, key=j+n_2k+1)\ F_{jN+i}, \qquad i=0,1,\cdots,N-1,\\
&\qquad j=0,1,\cdots,n_2-1\\
&F_i=F_i\omega_{2N}^{ik}, \qquad i=0,1,\cdots,n_2N-1\\
&F_{jN+i}=\sum_{t=0}^{n_2-1}F_{tN+i}\omega_{n_2}^{jt}, \quad i=0,1,\cdots,N-1, \quad j=0,1,\cdots,n_2-1\\
&write(f_3, key=j+n_2k+1)\ F_{jN+i},\\
&\qquad i=0,1,\cdots,N-1, \quad j=0,1,\cdots,n_2-1
\end{aligned}
\tag{5.34}
$$

データ B のFMTは上記に対してファイル f_3 を f_4 に変更する．F_i と ω_{2N}^{k} 及び $\omega_{n_2}^{j}$ の乗算は，F_i の内容をそれぞれ $F_{i+k \pmod N}$ 及び $F_{i+jm \pmod N}$ の位置に移せば良い．

5.5.2 複素FMTによる下位乗算

項別乗算は上位FMTの要素単位に下記(1)〜(5)を $k=0,1,\cdots,2N-1$ と $2N$ 回行う．ここで，ω_n は1の原始 n 乗根となる複素数 $\omega_n=e^{2\pi i/n}$ で，i は虚数単位を示す．また，$n=Ngh/2$ の関係がある．

(1) 整数をファイルから読込み複素数化

上位順FMTが完了したデータAとデータBをそれぞれ，ファイル f_3 とファイル f_4 から読込み F_j を実部に $F_{N/2+j}$ を虚部にした複素数に変換する．このとき，各 F_j は g 個の整数で構成されており，これを α 進 l 桁の gh 個の値に分割して複素数化する．記号 $cmp(a,b)$ は a を実部に，b を虚部に持つ複素数を示し，$\lfloor x \rfloor$ は実数 x の値を切り捨て整数化することを示す．

$$
\begin{aligned}
&read(f_3, key=k+1)\ (F_j, \quad j=0,1,\cdots,N/2-1)\\
&c_{jh+i}=cmp(\lfloor F_j/\alpha^{li}\rfloor-\lfloor F_j/\alpha^{l(i+1)}\rfloor\alpha^l, \lfloor F_{N/2+j}/\alpha^{li}\rfloor-\lfloor F_{N/2+j}/\alpha^{l(i+1)}\rfloor\alpha^l),\\
&\qquad i=0,1,\cdots,gh-1, \qquad j=0,1,\cdots,N/2-1\\
&read(f_4, key=k+1)\ (F_j, \quad j=0,1,\cdots,N/2-1)\\
&z_{jh+i}=cmp(\lfloor F_j/\alpha^{li}\rfloor-\lfloor F_j/\alpha^{l(i+1)}\rfloor\alpha^l, \lfloor F_{N/2+j}/\alpha^{li}\rfloor-\lfloor F_{N/2+j}/\alpha^{l(i+1)}\rfloor\alpha^l),\\
&\qquad i=0,1,\cdots,gh-1, \qquad j=0,1,\cdots,N/2-1
\end{aligned}
\tag{5.35}
$$

(2) 複素順FMT変換

n要素の拡張複素FMTで$2n$要素の実数に対応する負巡回乗算を行う．$j=0,1,\cdots,n-1$番目のc_j及びz_jの要素に共に$\omega_n^{j/4}$を乗算して，複素順FMT変換を行う．

$$
\begin{aligned}
&c_j = c_j\omega_n^{j/4}, \qquad z_j = z_j\omega_n^{j/4}, \qquad j=0,1,\cdots,n-1\\
&c_j = \sum_{i=0}^{n-1} c_i\omega_n^{ji}, \qquad z_j = \sum_{i=0}^{n-1} z_i\omega_n^{ji}, \qquad j=0,1,\cdots,n-1
\end{aligned}
\tag{5.36}
$$

(3) 項別乗算

2つの複素順FMT結果の各要素単位の複素乗算をc_jに求める．

$$
c_j = c_j \cdot z_j, \qquad j=0,1,\cdots,n-1 \tag{5.37}
$$

(4) 複素逆FMT変換

複素逆FMT変換を行った結果のc_jに対して，$\omega_n^{-j/4}$を乗算する．

$$
\begin{aligned}
&c_j = \sum_{i=0}^{n-1} c_i\omega_n^{-ji}/n, \qquad j=0,1,\cdots,n-1\\
&c_j = c_j\omega_n^{-j/4}, \qquad j=0,1,\cdots,n-1
\end{aligned}
\tag{5.38}
$$

(5) 複素数を整数化しファイルへ書き出し

n要素の拡張複素FMTによる乗算結果が，$2n$要素の実数に対応する負巡回乗算になることを利用する．乗算結果は本来整数であるが，計算で丸め誤差が発生しており，誤差を取り除くため四捨五入して整数に変換する．このとき，各F_jをg個の整数のα進ghl桁にするため1個がl桁の$c_{i,j}$をgh個重ね合せる．記号$real(\ )$は複素数の実部を，$imag(\ )$は虚部を取り出すことを，[ ]は記号内の実数を四捨五入して整数に変換することを意味する．

$$
\begin{aligned}
&F_j = \sum_{i=0}^{gh-1} [real(c_{i,j})]\alpha^{li}, \qquad F_{N/2+j} = \sum_{i=0}^{gh-1} [imag(c_{i,j})]\alpha^{li},\\
&\qquad j=0,1,\cdots,N/2-1\\
&write(f_3, key=k+1)\ \ (F_j, \quad j=0,1,\cdots,N-1)
\end{aligned}
\tag{5.39}
$$

5.5.3 上位逆 FMT 計算

上位順 FMT の逆変換により $2N^2g$ 個の整数に乗算結果を求め，正規化して N^2g の整数に α 進 $(Ng-1)Nlh$ の乗算結果を得る．

(1) 分割単位の上位逆 FMT 計算

逆 FMT 変換は下記の計算を $k=0,1,\cdots,m-1$ まで m 回行う．ω_{n_2} は 1 の原始 n_2 乗根で $\omega_{n_2}=\omega_{2N}^{m}=\alpha^{mglh}$ である．

$$
\begin{aligned}
&read(f_3, key=j+n_2k+1)\ F_{jN+i}, \qquad i=0,1,\cdots,N-1,\\
&\qquad j=0,1,\cdots,n_2-1\\
&F_{jN+i}=\sum_{t=0}^{n_2-1} F_{tN+i}\omega_{n_2}^{-jt}, \quad i=0,1,\cdots,N-1, \quad j=0,1,\cdots,n_2-1\\
&F_i=F_i\omega_{2N}^{-ik}, \qquad i=0,1,\cdots,n_2N-1\\
&write(f_3, key=j+n_2k+1)\ F_{jN+i},\\
&\qquad i=0,1,\cdots,N-1, \quad j=0,1,\cdots,n_2-1
\end{aligned}
\tag{5.40}
$$

(2) m 個の分割乗算結果から多数桁乗算の復元

下記の計算を $j=0,1,\cdots,n2-1$ で n_2 回行う．ω_m は 1 の原始 m 乗根である．

$$
\begin{aligned}
&read(f_3, key=j+n_2k+1)\ F_{kN+i}\\
&\qquad i=0,1,\cdots,N-1, \quad k=0,1,\cdots,m-1\\
&F_{kN+i}=\sum_{t=0}^{m-1} F_{tN+i}\omega_m^{-kt}/(2N)\\
&\qquad i=0,1,\cdots,N-1, \quad k=0,1,\cdots,m-1\\
&write(f_4, key=j+n_2k+1)\ F_{kN+i}\\
&\qquad i=0,1,\cdots,N-1, \quad k=0,1,\cdots,m-1
\end{aligned}
\tag{5.41}
$$

(3) 結果の正規化

N^2g 個の整数に求められている乗算結果から，同一桁を示す重なりの部分を正規化して，重なりの無い $N^2g/2$ 個の整数にする．乗算結果の桁数は $(Ng-1)Nlh$ である．ここで，配列 V と W はそれぞれ $Ng/2$ 及び Ng 個の整数配列であり，添え字はそのまま整数 1 個に対応する．結果の正規化のため，データは逆方向から処理する．各整数内の桁上げ処理も必要であるが，その部分の処理は省略して記載した．多数桁の乗算結果はファイル f_3 に得られる．

$$
\begin{aligned}
&read(f_4, key = 2N)\ W_i, \quad i = 0, 1, \cdots, Ng - 1 \\
&V_0 = 0, \qquad (V_i = W_i, \quad i = 1, \cdots, Ng/2 - 1) \\
&[\ read(f_4, key = k)\ W_{i+1}, \quad i = 0, 1, \cdots, Ng - 1, \\
&\quad V_i = V_i + W_{i+gN/2}, \quad i = 0, 1, \cdots, Ng/2 - 1, \\
&\quad write(f_3, k + 1)\ V_i, \quad i = 0, 1, \cdots, Ng/2 - 1 \\
&\quad V_0 = 0, \qquad (V_i = W_{i+1}, \quad i = 1, \cdots, Ng/2 - 1) \\
&\qquad , k = 2N - 1, \cdots, 2, 1\ ] \\
&V_i = V_i + W_{i+Ng/2}, \quad i = 0, 1, \cdots, Ng/2 - 1 \\
&write(f_3, key = 1)\ V_i, \quad i = 0, 1, \cdots, Ng/2 - 1
\end{aligned} \tag{5.42}
$$

5.6 数値実験結果

分割乗算の計算時間が計算桁数及び分割数にどのように依存するかを調べるため，複素 FMT 及び 2 段階 FMT を使用した分割乗算で数値実験した．複素 FMT による分割乗算は倍精度浮動小数点に 10 進 3 桁の数を入れて計算した．2 段階 FMT による分割乗算の場合は，32 ビット整数に 16 進 7 桁 (2 進 28 桁) 及び 64 ビット整数に 16 進 15 桁 (2 進 60 桁) 入れて計算した．2 段階 FMT の下位 FMT の計算は倍精度浮動小数点に，32 ビット整数の場合は 2 分割し 2 進 14 桁を入れ，64 ビット整数の場合は 4 分割し 2 進 15 桁を入れて計算した．数値実験に使用した計算機は下記の 2 種類である．1 回の測定では計算時間のぶれが大きいため，ワークステーションでは 2 回計測し，パソコンでは 5 回計測してその平均値を実測値とした．

(a) ワークステーション (Itanium2)
計算機： Intel Itanium2 1.7Ghz (6.8GFLOPS)
OS： Red Hat Linux 7.2
FORTRAN コンパイラー： Intel(R) Fortran, Version 8.0
コンパイルオプション： ifort -fsat
整数： 64 ビット整数

(b) パソコン (Duron)
計算機： ADM Duron 850Mhz (850MFLOPS)
OS： Microsoft Windows Me
FORTRAN コンパイラー： Visual Fortran Standard Edition 5.0A
コンパイルオプション： Full Optimizations
整数： 32 ビット整数

5.6.1 複素FMTによる分割乗算

本複素FMTによる多数桁乗算では，計算桁数によらず倍精度浮動小数点に10進3桁を詰めて計算する．そのため，n桁の多数桁乗算の計算量は$O(n\log(n))$となる．以下図中で示す桁数は結果の桁数で，入力桁数は結果の桁数の半分である．Itanium2による複素FMT分割乗算の計算桁数と計算時間の関係を図5.1に示す．結果の計算桁数を10進$12M$桁から2倍づつ増やし，$3072M$桁まで計算した．それぞれの分割数は32又は64であり，Mはメガの単位で$M = 2^{20} = 1,048,576$である．図5.1で測定点を結んだ線は直線となり，計算桁数をnとすると計算時間がほぼ$O(n\log(n))$になっていることが分る．更に，計算時間が分割数mに依存しないことを示すため，2種類の計算機で一定桁数の計算を分割数を変えて実験した．結果が10進$384M$桁となる分割乗算をItanium2で実測したものを図5.2に，$48M$桁の分割乗算をDuronで実測したものを図5.3に示す．分割数は，Itanium2が2から128まで，Duronが2から64まで2倍づつ変化させたものである．両図から，計算時間は分割数に依存しないことが分る．

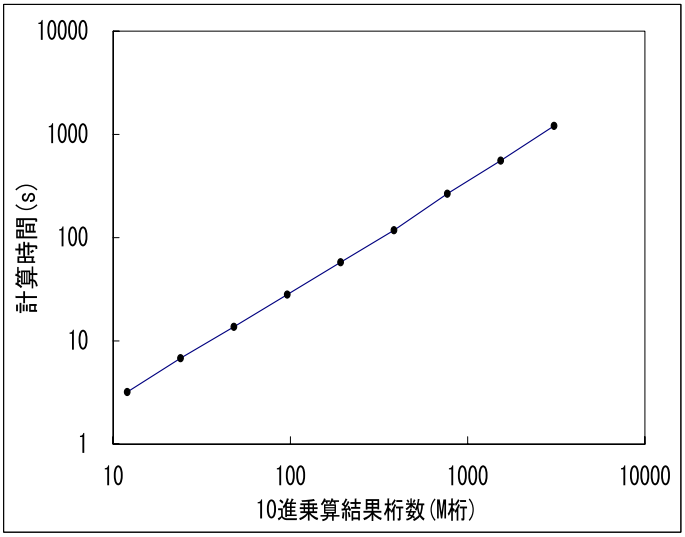

図 5.1: 複素FMT分割乗算の桁数による計算時間 (Itanium2)

5.6.2 2段階FMTによる分割乗算

2段階FMT計算は，上位FMT計算と下位FMT計算の2段階に分け，更に上位FMTは分割数mに分ける処理と分割した後の処理の2ステップに分けて計算する．上位FMTの計算では整数を使用し，Itanium2では64ビット整数に16進15桁を入れ，Duronで

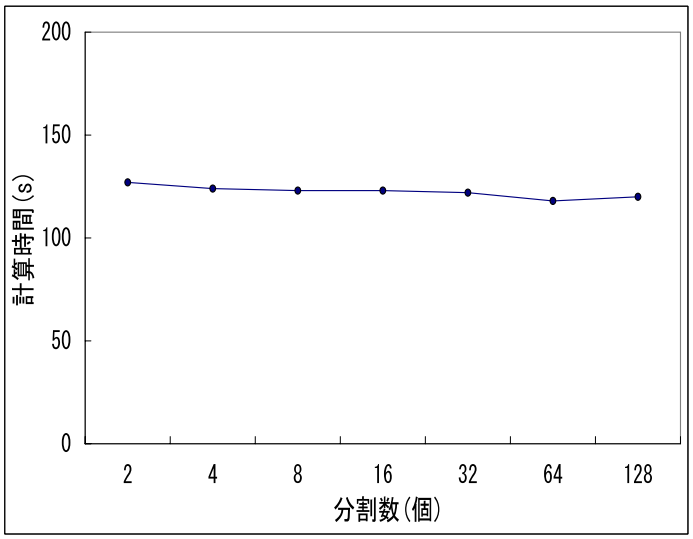

図 5.2: 複素 FMT 乗算の分割数と計算時間 (Itanium2,384M 桁)

は 32 ビット整数に 16 進 7 桁を入れて計算した．下位 FMT は拡張複素 FMT を使用して，負巡回多数桁乗算をする．複素 FMT は倍精度浮動小数点を使用し，Itanium2 では 64 ビット整数を 4 分割し 2 進 15 桁を詰め，Duron では 32 ビット整数を 2 分割し 2 進 14 桁を詰めて計算した．数値実験では計算桁数によらず，要素あたり詰める桁数を一定としたため，n 桁の多数桁乗算の計算量は $O(n\log(n))$ となる．以下図中で示す桁数は結果の桁数で，入力桁数は結果の桁数の半分である．Itanium2 による 2 段階 FMT 分割乗算の計算桁数と計算時間の関係を図 5.4 に示す．結果の桁数を 16 進で $30M$ 桁から 2 倍づつ増やし，$7680M$ 桁まで計算した．それぞれの分割数は 32 又は 64 であり，M はメガの単位で $M = 2^{20} = 1,048,576$ である．図 5.4 で測定点を結んだ線は直線となり，計算桁数を n とすると計算時間がほぼ $O(n\log(n))$ になっていることが分る．更に，計算時間が分割数 m に依存しないことを示すため，2 種類の計算機で一定桁数の計算を分割数を変えて測定した．結果が 16 進 $480M$ 桁となる分割乗算を Itanium2 で実測したものを図 5.5 に，$112M$ 桁の分割乗算を Duron で実測したものを図 5.6 に示す．分割数は，Itanium2 が 2 から 128 まで，Duron が 2 から 64 まで 2 倍づつ変化させたものである．両図から，計算時間は分割数に依存しないことが分る．

5.7 まとめ

高速剰余変換（FMT）を適用し使用メモリを削減するため，ファイル I/O を利用した多数桁の乗算を高速に行う方法を考案した．本多数桁の分割乗算で n 桁の多数桁乗

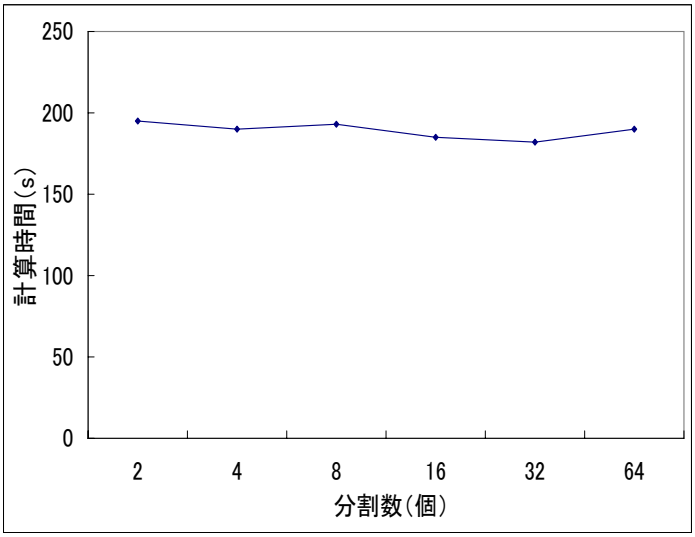

図 5.3: 複素 FMT 乗算の分割数と計算時間 (Dulon,48M 桁)

算を m 分割して行う時，使用メモリ量は分割前のほぼ $1/m$ に減少し，計算量とファイル I/O 量の両方とも分割数に依存せず，それぞれ $O(n \log n(\log\log n))$ 及び $O(n)$ になることを理論的に示した．理論を裏づけするため，ワークステーションとパソコンを使用した数値実験を行った．数値実験では，複素 FMT 及び 2 段階 FMT を利用した分割乗算を対象にして，計算桁数及び分割数を変化させた場合の測定を行い評価した．その結果，計算時間は分割数に依存することなく，理論評価式 (計算量) にほぼ従うことが分かった．今後，本方式を 2002 年に達成した円周率世界記録のプログラムに追加し，次期円周率計算の世界記録はメモリ量 (使用ノード数) が少ない場合でも実行可能なようにする．

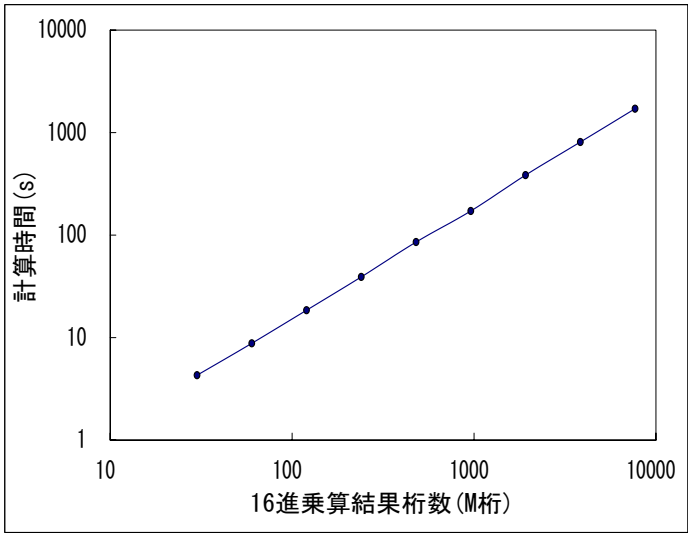

図 5.4: 2 段階 FMT 分割乗算の桁数による計算時間 (Itanium2)

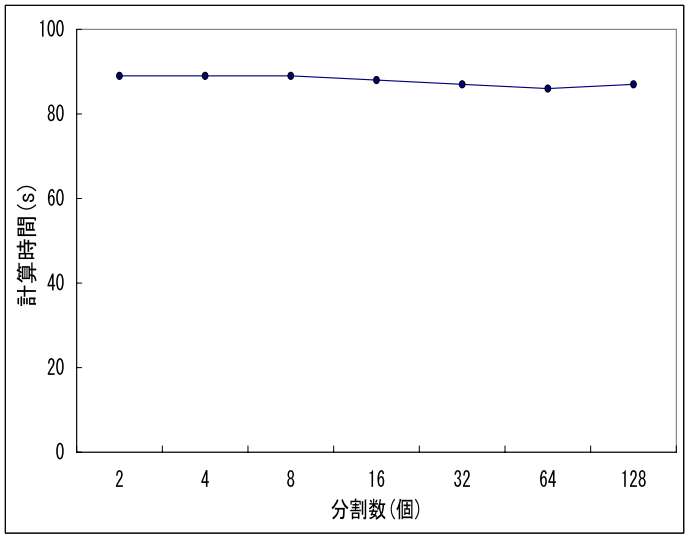

図 5.5: 2 段階 FMT 分割乗算の分割数と計算時間 (Itanium2,480M 桁)

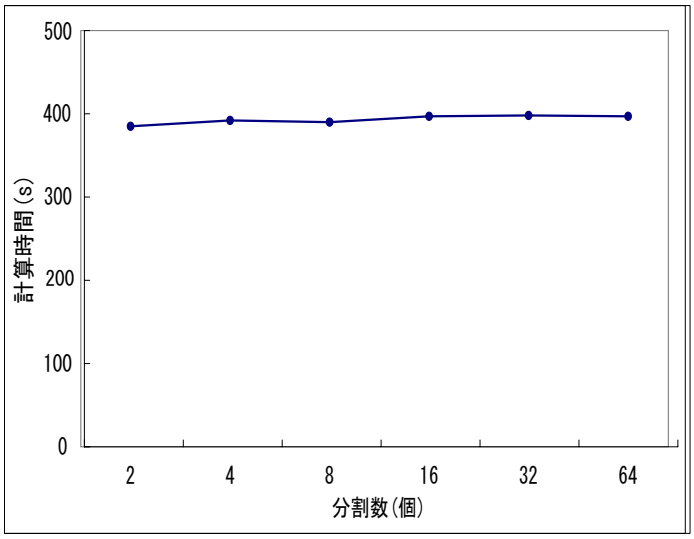

図 5.6: 2 段階 FMT 分割乗算の分割数と計算時間 (Dulon,112M 桁)

第6章　1.2兆桁π計算の世界記録

6.1　はじめに

多数桁の円周率計算は古代から数学者によって計算されてきた．1949 年に計算機で約 2 千桁を計算し，約半世紀の間に表 6.1 に示すように 6 億倍と飛躍的に計算桁数が伸びている．1981 年までの最初の 32 年間はすべて Arctan 公式での記録で，それ以後 1999 年の 2061 億桁の記録までは，算術幾何平均法 (AGM 法，ガウス・ルジャンドル法)[25] [32] か Chudnovsky の公式 [26] を使用している．その理由は，計算桁数を n とすると Arctan 公式の計算量が $O(n^2)$ [24] となるのに対して，AGM では $O(n(\log n)^2)$[24] に削減されるためである．この間，Chudonovsky 兄弟が Chudonovsky の発見した級数展開法で計算しているが，その詳細は公表されていない．今回，1983 年の 1000 万桁以来，約 20 年ぶりに Arctan 公式での世界記録を達成した．これは従来，Arctan 公式の計算量が $O(n^2)$ であったのを，分割有理数化法 (DRM) を考案 [9] して AGM 法とほぼ同等な $O(n(\log n)^{2\sim3})$[7] に削減できたためである．今回は最初から 1 兆桁以上の記録達成を目標とした．1 兆桁計算で最も問題になるのは，計算機の記憶容量である．このため，工夫により 1 兆桁同士の乗算回数を，AGM 法より削減可能な級数展開法を採用することにした．2001 年に，計算量の最も少ない Chudonovsky の公式で正計算をし，Ramanujan の公式で検証計算をした．この時は 3597 億 1280 万 9450 桁以降で不一致が発生した．2002 年に，計算量が Chudonovsky の公式の約 3 倍となるが，なじみの深い Arctan 公式にプログラムを変更して再挑戦した．まず，2002 年 10 月 4 日に正計算と検証計算の結果，16 進数で 1 兆 307 億桁が正確に計算できたことを確認した．次いで，同年 11 月 24 日に，10 進数で 1 兆 2411 億桁に正確に変換できたことを，16 進数から 10 進数への変換及び再変換で確認した．従来までは 10 進数の π の世界記録計算においては，10 進の π の値を直接計算していたが，今回は DRM 法の基底変換への適用性の検証のため，16 進で π を計算し 10 進へ変換する方法を採用した．また，2001 年の 1 兆桁計算は SR8000/MPP の 128 ノード（2TB のメモリ）を使用したが，2002 年の計算は半分の 64 ノード（1TB のメモリ）で実行する必要が発生したため，Arctan 公式に変更するなどの更なるメモリ削減対策を行なった．

表 6.1: 計算機による円周率計算の記録

NO.	記録保持者	使用計算機	計算 年月	計算桁数	計算時間 検証時間	計算公式 検証公式
1	Reitwiesner etc.	ENIAC	1949	2,037	～70h	M(M)
2	Nicholson etc	NORC	1954	3,092	13m(-)	M(M)
3	Felton	Pegasus	1957	7,480	33h	K(G)
4	Genuys	IBM 704	1958/1	10,000	1h40m	M(M)
5	Guilloud	IBM 704	1959	16,167	4h18m	M(M)
6	Shanks, Wrench	IBM 7090	1961	100,265	8h43(4h22)	S(G)
7	Guilloud, Filliatre	IBM 7030	1966	250,000	41h55m 24h35m	G S
8	Guilloud, Dichampt	CDC 6600	1967	500,000	28h10 16h35	G S
9	Guilloud, Bouyer	CDC 6600	1973	1,001,250	23h18m 13h40m	S G
10	三好, 金田	FACOM M-200	1981	2,000,036	137h18 143h18	K M
11	田村, 金田	HITAC M-280H	1982	4,194,288	2h21 6h52	G-L G-L
12	田村, 金田	HITAC M-280H	1982	16,777,206	30h 6h36	G-L G-L
13	後, 金田	HITAC S-810/20	1983/10	10,000,000	24h 30h	G G-L
14	Gosper	Symbolics	1985/10	17,526,200	-(28h)	Ram(B4)
15	Bailey	CRAY-2	1986/1	29,360,111	28h(40h)	B4(B2)
16	金田, 田村	HITAC S-810/20	1986/9	33,554,414	6h36 23h	G-L G-L
17	金田 田村	HITAC S-810/20	1986/10	67,108,839	23h 35h15m	G-L G-L
18	金田, 田村 他	NEC SX-2	1987/1	134,214,700	35h15 48h2m	G-L B4
19	金田, 田村	HITAC S-820/80	1988/1	204,326,551	5h57m 7h30m	G-L B4
20	Chudnovskys	IBM-3090	1989/6	535,339,270	-	Chu
21	金田, 田村	HITAC S-820/80	1989/7	536,870,898	67h13m 80h39m	G-L B4
22	Chudnovskys	IBM-3090	1989/8	1,011,196,691	-	Chu(Chu)
23	金田, 田村	HITAC S-820/80	1989/11	1,073,740,799	74h30m 85h57m	G-L B4
24	Chudnovskys	-	1991/8	2,260,000,000	-	Chu(Chu)
25	Chudnovskys	-	1994/5	4,044,000,000	-	Chu(Chu)
26	高橋 金田	HITAC S-3800/480	1995/10	6,442,450,000	116h38m 131h40m	B4 G-L
27	Chudnovskys	-	1996/3	8,000,000,000?	-	Chu(Chu)
28	高橋, 金田	HITACHI SR2201	1997/4	17,179,869,142	5h11m 5h26m	G-L B4
29	高橋, 金田	HITACHI SR2201	1997/7	51,539,600,000	29h3m 37h8m	B4 G-L
30	高橋, 金田	HITACHI SR8000	1999/9	206,158,430,000	37h21 46h7m	G-L B4
31	金田, 後 , 黒田 他	HITACHI SR8000/MPP	2002/11	1,241,100,000,000	423h20h 181h5m	T S

公式欄の G,K,M,S,T はそれぞれ arctan 級数による Gauss,Klingenstierna,Machin,Stormer, 高野の公式である.
G-L,B2,B4 はそれぞれ AGM による Gauss-Lugendre,Borwein の 2 次及び Borwein の 4 次の公式である。
Ram,Chu はそれぞれ Ramanujan 級数展開における Ramanujan 及び Chudnovsky の公式である。

6.2 πの1兆桁計算方針

6.2.1 1兆桁挑戦の契機

1997年にDRM法の原理を考案[9]し，1998年に京都大学数理解析研究所で発表した．このとき，πの級数展開式にDRM法を適用すると，従来世界記録更新に利用されていた算術幾何平均法(AGM法)と同等の計算量で，メモリ使用量が大幅に削減できることが分かった．これにより，使用メモリ量が1TBの計算機で1兆桁π計算の可能性が出てきた．その結果，1兆桁π計算を達成するためのアルゴリズムの検討とプログラム作成を開始した．2001年春から，東大金田康正教授をプロジェクトリーダとする，東京大学と日立製作所の共同プロジェクト体制で，プログラムの並列化とチューニングを実施し記録達成を目指した．

6.2.2 主な検討事項

最初は使用メモリ量を2TBとし，16進1兆桁(10進1.2兆桁)の計算を目標にした．しかし，2001年にSR8000/MPPの128ノード(メモリ2TB)での正計算に失敗し，128ノード使用は周囲の状況から困難となった．そのため，2002年の目標は半分の64ノード(メモリ1TB)で，同じ16進1兆桁(10進1.2兆桁)計算をする方針に変更した．以下に目標を達成するための検討事項と結果の概要を示す．検討結果で採用した手法の詳細は別に記載する．ここではどのような項目を検討し，どれを最終的に採用したかを記載する．

π計算の全体使用領域削減

1. π計算公式及び基底数
   DRM法の考案によりπの級数展開がAGM法と同程度の計算量で可能になった．また，級数展開はAGM法に比較して，1兆桁同士の乗算回数を大幅に削減できるため，メモリの使用量を削減できる利点があり，級数展開を使用することにした．π計算の基底は10進表示で行う方法と，16進(2進)表示で行う方法の2種類が考えられる．従来の記録は10進表示で計算されていたが，今回はメモリ効率を考慮し，16進表示のπを計算し，10進数表示に変換する方法を採用することにした．これは，DRM法で10進16進の基底変換が可能であることを，大規模計算で実証するためでもあった．級数展開として，2001年は速度を重視しChudnovsky及びRamanujanの公式を採用した．2002年はメモリ使用量の削減が必要なことに加え，公式の認知度及びπ計算が計算機の耐久性試験となること

などを考慮し，Arctan 公式を採用した．

2. 級数の分割計算
   有理数で級数展開された値を DRM 法で計算する場合，分子分母ともに必要な桁数で打ち切り，全項数を纏めて計算するのが計算量は最小となる．ただし，今回は 1 兆桁を目標とするため，多量にメモリを使用する 1 兆桁同士の乗算の回数を削減する必要がある．そこで，級数を分割し，求める桁数の $1/p$ の桁数まで DRM 法で通分して，通分した分数は通常の方法で計算することにした．1TB の計算機で 1 兆桁を計算するケースを想定し，$p = 8$ とすることにした．その理由は，$p = 8$ では分割しないケース ($P = 1$) に比較して計算量が約 1.5 倍の増加であるが，$p = 16$ では約 2.3 に増加するためである．$p = 7$ とし $p = 8$ と同じ桁数 (16 進数で 1 兆 307 億/8 桁) まで通分すると，16 進数で 9018 億桁，10 進数で 1 兆 859 桁が計算できるが 16 進数で目標の 1 兆桁を越さないため，今回は使用しないことにした．

3. Disk の活用
   Arctan 公式で円周率を計算する場合は，複数個の Arctan 公式を使用する．1TB のメモリ量で 1 兆桁の計算をすると，1 兆桁の Arctan 結果を保存するだけでメモリ量の半分の 0.5TB の記憶容量が必要である．このため，途中計算の Arctan 結果は Disk に保存する必要がある．更に，DRM 法で Arctan 級数を分割して計算すると，その途中結果も Disk に保存する必要がある．Arctan 結果の Disk への保存に対して，途中結果の保存は桁違いのファイル I/O 量となる．このため，ファイル I/O の高速化対策が必要である．また，途中結果が Disk に保存されているため，その時点での中断及び再開が容易である．

多数桁の記憶方法と多数桁乗算方法

従来より 6 倍の計算桁数に対するメモリ効率を向上させるためには，DRM 法の利用による公式の変更と級数の分割計算だけでは不十分で，多数桁乗算を考慮した多数桁の記憶方法を十分検討する必要がある．多数桁の記憶方法はプログラムの原型を PC で開発することを考慮し，PC でも１ノードのスーパーコンでも共に動作する方式とした．また，ノード間並列化により変更するプログラムの範囲をできるだけ少なくすることにした．

1. 多数桁の記憶方法
   多数桁の保存は利用効率を考え，スーパーコンでは 64 ビット整数 (8 バイト整数，I*8) を使用し，PC では 32 ビット整数 (4 バイト整数，I*4)) を使用して保存する

ことにした．I*8 又は I*4 に記憶する値は 2 進数 (16 進数) 又は 10 進数の α 桁とした．桁数 α はプログラム中では可変とし，多数桁乗算のためにそれらの値を PC では 2 分割し，スーパーコンでは 4 分割できることを仮定した．I*4 では 2 進数で 28 桁又は 10 進数で 8 桁，I*8 には 2 進数で 60 桁又は 10 進数で 16 桁を記憶した．多数桁の記憶方式は，DRM 法による有理数計算に使用する，固定小数点表示と，実数として使用する浮動小数点表示の両方をサポートした．浮動小数点表示の仮数部は先頭の位置に小数点を置き，表現方法は固定小数点表示と同一にすることにした．固定小数点及び浮動小数点とも，桁数と並列化を考慮して 3 種類を用意した．

2. 多数桁の乗算方法
   桁数が大きい場合の多数桁の乗算方式として下記の 4 種類を比較検討した．
   (a) 実 FFT による多数桁の乗算．
   (b) 実 FFT と Karatsuba 法による多数桁の乗算．
   (c) 複数個の整数 FFT と中国剰余定理による多数桁乗算．
   (d) 2 段階 FFT（上位は整数 FFT，下位は実数 FFT）による多数桁乗算．
   入力が共に 10 進の 0.5 兆桁で出力が 1 兆桁の乗算の使用メモリ量で比較すると，(a),(b),(c),(d) はそれぞれ 9.0 兆バイト，2.4 兆バイト，1.9 兆バイト及び 1.3 兆バイトとなる．ここで，(b) は Karatsuba 法を 4 回適用したもので，(c) は 8 個の整数 FFT を利用した場合である．(b) 及び (c) はそれ以上適用回数及び利用個数を増加してもメモリ使用量はほとんど減少しない．また，Karatsuba 法は 1 回適用するごとに計算量が約 1.5 倍増加する欠点がある．一方，2 段階 FFT は下位の実数 FFT で計算する桁数が 1 兆桁の平方根の 100 万桁程度で済むこと，及び並列化は上位の整数 FFT にだけ実施すれば良い利点がある．これらを，考慮して 2 段階 FFT を桁数の多い多数桁の乗算方式として採用した．

結果の検証方法

1. 16 進数の π の検証
   異なる 2 公式による計算結果を比較する．16 進表現で，2 公式の計算結果の最後の数十桁を除いて一致していれば両者の計算結果は正しいと判定する．両公式とも Arctan 公式の場合は，高速化のため Arctan の引数が一致しているものは，正計算で計算したものを検証計算でも利用する．そのため，両公式で同じ引数の Arctan に掛ける係数は必ず異なるものを採用する必要がある．今回の計算では正計算で計算した Arctan(1/57) と Arctan(1/239) の値を検証計算でも利用した．これらの係数は正計算ではそれぞれ 32 及び-5 で検証計算では 44 及び 7 で異なる

値である．

2. 10 進数の π の検証
   求めた 16 進数の円周率の値を 10 進数に DRM 法で変換し，10 進数に変換した値から再度 16 進数に DRM 法で再変換する．再変換した 16 進数が入力した 16 進数と入力した桁まで一致したら，10 進数への変換は正しいと判定する．

6.2.3 プログラム作成方針

プログラムの作成は下記の 4 段階で作成した．最初の 2 段階は著者が単独で作成し，後半の 2 段階は東京大学金田教授をプロジェクトリーダとする東京大学と日立製作所の合計 10 名のプロジェクトチームで作成した．

1. DRM 法の π 計算原理確認プログラム
   DRM 法を利用して，Arctan 公式及び Ramanujan 公式による π の多数桁の計算プログラム．DRM 法での級数計算プログラムが正しく動作することの確認が目的である．メモリの使用量は考慮しないで，A-Type データ形式だけをサポートする．

2. PC で動作する π 世界記録用原型プログラム
   32 ビット整数を使用し，2 段階 FFT による乗算及び B-Type データと Disk を使用した計算をサポートし，π 世界記録用原型プログラムと位置づけ，主計算に向けてのパラメータの調整を目的とした．16 進 10 進及び 10 進 16 進の基底変換プログラムも同様である．

3. SR8000 の 1 ノード用プログラム
   64 ビット整数を使用し，ノード内並列化とマシン依存のチューニングを実施した．

4. SR8000 の並列ノードプログラム
   C-Type データのサポートとノード間並列化及びチューニングを実施した．

6.3 π 計算の全体使用領域削減

6.3.1 DRM 法を使用して π 計算公式の変更

計算機による円周率の世界記録は表 6.1 の 1982 年の NO.11 の記録を境に，大きく二つに分けられている．1981 年の NO.10 による 200 万桁の計算までは総て Arctan 公式による級数展開を利用している．一方，NO.11 以降は算術幾何平均法 (AGM 法，相加

相乗平均化法，サラミン・ブレント法，ガウス・ルジャンドル法とも呼ばれる）及び Ramanujan 公式の一種である Chudonovsky の公式である．Chudonovsky 兄弟が実施した，Chudonovsky の公式の計算手法は公開されていないが，推測によると今回使用した DRM 法の類似のアルゴリズムを使用したと思われる．NO.11 以降に世界記録に使用されてきた AGM 法は，1976 年に Brent と Salamin の 2 人によってまったく独立に発見された，楕円積分に関する Gauss の計算法と，Legendre の関係式を用いて円周率を求める公式である．この公式の利点は，Arctan 公式による桁数 n の計算量が $O(n^2)$ であったのに対して，n 桁同士の多数桁乗算の計算量を $M(n)$ とし AGM 法を適用すると，円周率計算の計算量を $O(M(n)\log n)$ に削減できることである．また，FFT を使用すると $M(n) = O(n \log n(\log\log n))$ となることが知られている．このため，1982 年以降の円周率世界記録の計算には AGM 法が使用されてきた．しかし，AGM 法の欠点は求める桁数と同じ桁数の乗算，除算及び平方根の計算が $\log n$ 回程度必要なことと，同一桁数の配列を計算の過程で保持する必要なことである．この当時まだ，ファイル I/O を使用した高速な分割乗算方式がなかったため，AGM 法の計算には多くのメモリ量が必要であった．AGM 法による 1999 年の 2061 億桁の世界記録の例で見ると，1 兆バイト (1T バイト) のメモリ量を持つ計算機を使用している．このため，バイト当たりの計算桁数は 0.2 桁/バイトとなる．1999 年の世界記録の達成においては，メモリ削減のために Karatsuba 法と実 FFT を組み合わした多数桁乗算方式の採用等，種々の工夫がされている．今回，16 進で 1 兆桁，10 進で 1.2 兆桁の計算を目標とした．使用メモリは最大 2 兆バイト (2T バイト) のため，バイト当たりの計算桁数は 0.6 桁/バイトとなり，2061 億桁の場合の 3 倍になる．このため，多数桁乗算のメモリ当たり桁数を多少改善したのでは不可能で，計算公式の変更が必要となった．幸い，考案した DRM 法を級数計算公式に適用すると，n 桁の円周率の計算量は $O(M(n)(\log n)^2)$ と AGM 法とほぼ同一になり，メモリ使用量を削減することが可能となる．目的の円周率の桁数で，計算量を推定すると，Ramanujan 級数展開の Chudnovsky 公式が最も少ないことが判明した．計算量からの推定では，Chudnovsky 公式が最速で，次に Ramanujan の公式で，その次に Arctan 公式及び AGM 法となる．その比率は，メモリが十分利用できるとし Chudnovsky 公式を 1 とすると，それぞれ約 1.6 倍，約 3.0 倍及び約 3.0 倍かかる．面白いことに，1 兆桁の計算では Arctan 公式と AGM 法は同等の計算量となる．両公式とも多数の計算公式が知られているが，高速なもの数種類に限るとほぼ同等で，差は 1 割以内である．2001 年に 2T バイトのメモリを使用し計算時間が短くなるように，主計算に式 (6.1) の Chudnovsky 公式を，検証計算に式 (6.2)Ramanujan の公式を使用することにした．式 (6.1) の計算項数 $N1$ は求める 10 進桁数の約 0.07 倍で，式 (6.1) の計算項数 $N2$ は約 0.13 倍である．しかし，これはプログラム不正のため 16 進約 3597 億桁のところで結果の不一致が発生し，2T バイトのメモリの計算機を使用する機会が得られなかったために断念した．この 2 つの公式は円周率の逆数を求める公式のため，

最後に求める桁数の逆数計算が必要になる．この部分の計算は，16 進 1 兆桁を 4 分割しファイル I/O を使用して実施したが，2T バイトのメモリが必要なため，半分の 1T バイトでは実行できないプログラムになっていた．そこで，2002 年は正計算の計算時間が 3 倍必要ではあるが，1T バイトのメモリで実行可能な Arctan 公式に正計算及び検証計算共に変更することにした．

$$\frac{1}{\pi} = \frac{12}{640320^{3/2}} \sum_{j=0}^{N1} \left(\prod_{k=1}^{j} \frac{(2k-1)(6k-5)(6k-1)}{9k^3 106720^3} \right) \cdot(545140134j + 13591409) \tag{6.1}$$

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{j=0}^{N2} \left(\prod_{k=1}^{j} \frac{(2k-1)(4k-3)(4k-1)}{2k^3 198^4} \right) \cdot (26390j + 1103) \tag{6.2}$$

6.3.2 DRM 法の分割による使用領域削減

DRM 法による Arctan 関数のトーナメント通分処理は，最終的に 1 個の分数になるまで計算し，分子，分母共に計算桁数以上になればその桁数で打ち切るのが計算量を最小にする．しかし，今回はメモリの使用量を少なくし，1T バイトのメモリで 16 進 1 兆桁を計算するのが目標であったため，計算量の多少の増大には目をつぶり，通分処理は計算桁数の 1/8 で打ち切り，正確な分数（整数）から桁数の長い実数を計算する分割 DRM 方式を採用した．このため，通分化した級数の計算は後の項から行った．通分処理を計算桁数の 1/16 で打ち切れば，メモリ的には楽であるが，計算量が大きく増加すると推定されたため，ぎりぎり 1T バイトのメモリに納まる 1/8 で打ち切る方法を採用した．この時の計算量は，分割しないときの計算量の約 1.5 倍である．この方式により，16 進 1 兆桁同士の乗算及び除算は不要となり，ファイルに記憶した 1 兆桁とメモリ上の 1/8 兆桁の乗算及び除算が最大桁数の乗除算処理となった．

6.3.3 16 進数の π を計算し 10 進数に変換

多数桁の円周率の計算は，多数桁を数桁の 10 進数に分けて記憶し，計算もそのまま数桁の 10 進数単位で行うのが一般的である．1999 年の 2061 億桁の計算までの世界記録は総て 10 進数単位で計算されたと思われる．しかし，今回はメモリ利用効率と，DRM 法が 2 進 10 進変換などの基底変換にも適用可能なことを実証する目的で，Arctan 公式により 16 進数の π を計算し，16 進の値を 10 進数に変換する方式を採用した．10 進数

への変換が正しくおこなわれたかの検証は，変換した 10 進数を 16 進数に逆変換して，元の 16 進数に一致することでおこなう．16 進数（2 進数）の π を計算し 10 進数に変換することのメリットは，計算機の整数及び浮動小数点が 2 進数表示のためである．16 進数だと 64 ビット整数に 16 進 15 桁（2 進 60 桁）記憶し，ビットシフトにより，桁上げ処理が容易におこなえる．また，64 ビット整数に記憶した 2 進 60 桁を 4 分割して，浮動小数点に渡す処理が容易である．一方，ほぼ同一のメモリ効率で 10 進表示だと，64 ビット整数に 10 進 18 桁つめる必要がある．しかし，10 進 18 桁つめると桁上げ処理にシフト演算が使用できなく，ビットの余裕が少ないため，桁上げ処理が複雑になる．また，18 桁つめだと 4 分割して浮動小数点に渡す処理はできない．このため，10 進数表示だと 64 ビット整数に 16 桁つめて計算することになり，メモリ効率が低下する．また，Arctan 公式による π の計算と 16 進 10 進変換の計算量を比較すると，16 進 10 進変換の方が約 1/20 と小さい．このため，π 計算を 16 進で行い，10 進数に変換することが効果的と判断した．10 進数で直接計算することに対するデメリットは，計算方式が複雑になりプログラム行数が長くなることである．今回作成したプログラム行数が約 8 万行になった内の，約 3 割は 16 進 10 進及びその逆変換用のものである．

6.4 多数桁の記憶方法と多数桁乗算方法

6.4.1 多数桁の記憶方法

π の世界記録計算では，64 ビット整数 (8 バイト整数，I*8) に 2 進 60 桁 (16 進 π の計算) 又は 10 進 16 桁 (10 進数への変換) を記憶した．多数桁乗算及び除算の計算は，実数 FFT(下位 FFT) 使用時は 4 分割し，倍精度浮動小数点 (8 バイト) に 2 進 15 桁又は 10 進 4 桁を記憶し計算した．整数 FFT(上位 FFT) 使用時及び多数桁の加減算処理では 64 ビット整数に 2 進 60 桁又は 10 進 16 桁を記憶したまま計算した．ただし，PC での原型プログラムでは 64 ビット整数が利用できないため，32 ビット整数 (4 バイト整数，I*4) に 2 進 28 桁又は 10 進 8 桁を記憶し，実 FFT 使用時は 2 分割してテストした．プログラム上は I*8 及び I*4 共に上記桁数を上限としそれぞれ 4 及び 2 刻みの可変桁数が指定可能とし，記憶桁数を変えたテストも実施した．SR8000 での世界記録の計算では，多数桁の記憶はノード内で桁数が小のときは A-type データ形で，桁数が大の時は B-type データの形で記憶し，複数ノードに分散記憶するときは C-type データの形で記憶した．いずれの type のデータも，固定小数点形式と浮動小数点形式の 2 種類を用意し，必要に応じて利用した．C-type データはノード並列化のときに必要なもので，PC での原型プログラムには実装していない．

A-type データ

1. 固定小数点形式
   64 ビット整数 (I*8) に 2 進 60 桁又は 10 進 16 桁を詰めて記憶する．記憶要素数 (桁数)n は別に記憶する．n 要素で構成される多数桁を E の $n-1$ 次の多項式 f で示すと式 (6.3) の　のようにになる．多項式 f の係数 a_k を a_1 から順に a_n まで 64 ビット整数として記憶する．

$$\begin{aligned} f &= a_1 E^{n-1} + a_2 E^{n-2} + \cdots + a_{n-1} E + a_n \\ & E = 2^{60} \; or \; 10^{16}, \quad \mid a_k \mid < E/2, \quad k = 1, 2, \cdots, n \end{aligned} \tag{6.3}$$

   係数 a_k は上記以外に，非ゼロとなる先頭の要素と同一符号を持たす表示もオプションとして可能である．この時は $\mid a_k \mid < E$ である．

2. 浮動小数点形式
   64 ビット整数 (I*8) の指数部 e と I*8 の n 個の要素の仮数部で構成する．仮数部の 1 要素に 2 進 60 桁又は 10 進 16 桁を詰めて記憶する．浮動小数点表示の多数桁を E の $n-1$ 次の多項式 f で示すと式 (6.4) のようになる．計算結果は正規化する．即ち，a_1 がゼロとなるのは真のゼロのときだけである．

$$\begin{aligned} f &= E^e (a_1 E^{-1} + \cdots + a_{n-1} E^{1-n} + a_n E^{-n}) \\ & E = 2^{60} \; or \; 10^{16}, \quad \mid a_k \mid < E/2, \quad k = 1, 2, \cdots, n \end{aligned} \tag{6.4}$$

   係数 a_k は固定小数点の場合と同様に各係数が同一符号を持つ表示も可能である．

B-type データ

本 type のデータの乗算は 2 段階 FFT を利用しておこなう．2 段階 FFT の上位 FFT の要素数を N とし，1 要素は 64 ビット整数 (I*8)n 個で構成する．本 type のデータは乗算の中間結果の格納にも利用するため，各要素の先頭の 1 個 (I*8) は，多数桁の格納時にはダミーとして常にゼロにしておく．

1. 固定小数点形式
   A-type データと同様に 64 ビット整数 (I*8，ワード) に 2 進 60 桁又は 10 進 16 桁を詰めて記憶する．式 (6.5) のように N 要素 (nN ワード) の多数桁を E の $(n-1)(N-1)$ 次の多項式で示す．g の最終的な係数 $a_{j,k}$ を $a_{1,1}$ から順に $a_{n,N}$ まで記憶する．

$$g = b_1 E^{(n-1)(N-1)} + b_2 E^{(n-1)(N-2)} + \cdots + b_{N-1} E^{n-1} + b_N$$

$$
\begin{aligned}
& b_k = a_{1,k}E^{n-1} + a_{2,k}E^{n-2} + \cdots + a_{n-1,k}E + a_{n,k} \\
& E = 2^{60}\ or\ 10^{16}, \quad a_{1,k} = 0, \\
& \mid a_{j,k} \mid < E/2, \quad j = 2, 3, \cdots, n, \quad j = 1, 2, \cdots, N
\end{aligned} \tag{6.5}
$$

係数 $a_{j,k}$ は A-type の場合と同様に各係数が同一符号を持つ表示も可能である．

2. 浮動小数点形式
   64 ビット整数 (I*8) の指数部 e と Nn 個の I*8 の仮数部で構成する．B-type データの固定小数点形式を浮動小数点形式に変更したもので，指数は要素単位 (I*8 が n 個) に指定する．浮動小数点表示の多数桁を E の $(n-1)(N-1)$ 次の多項式 g で示すと式 (6.6) のようになる．指数 e と g の仮数部の最終的な係数 $a_{j,k}$ を $a_{1,1}$ から順に $a_{n,N}$ まで記憶する．計算結果は正規化する．即ち，b_1 の係数 $a_{l,1}, l = 2, \cdots, n$ が総てゼロとなるのは真のゼロのときだけである．

$$
\begin{aligned}
g \;=\; & E^{e(n-1)}(b_1 + b_2E^{-(n-1)} + b_3E^{-2(n-1)} + \cdots + b_NE^{-(n-1)(N-1)}) \\
& b_k = a_{1,k}E^{n-1} + a_{2,k}E^{n-2} + \cdots + a_{n-1,k}E + a_{n,k} \\
& E = 2^{60}\ or\ 10^{16}, \quad a_{1,k} = 0, \\
& \mid a_{j,k} \mid < E/2, \quad j = 2, 3, \cdots, n, \quad j = 1, 2, \cdots, N
\end{aligned} \tag{6.6}
$$

C-type データ

本 type のデータは B-type データと基本的構成は同じであるが，複数ノードで１つの値を記憶する．ノード間のデータの配置方法はブロック・サイクリック分割である．１つのデータに対するノード間の並列処理は本 Type のデータに対してだけおこなう．使用する全ノードで１つの値を持つことも，ノードを複数のグループに分けてグループ毎に１つの値を保持することもできる．

1. 固定小数点形式
   構成方法は B-type データと同じで，要素数を N とし，１要素は 64 ビット整数 (I*8)n 個で構成する．各要素の先頭の１個 (I*8) はダミーとして常にゼロにしておく．ブロック・サイクリック分割の単位は１要素，つまり I*8 が n 個である．

2. 浮動小数点形式
   構成方法は B-type データの浮動小数点形式と同様である．仮数部のブロック・サイクリック分割方法は C-type データの固定小数点形式と同じである．一方，指数部は各ノード毎に保持する．但し，有効な指数部は先頭のノードのものだけである．

6.4.2　2 段階 FFT による多数桁乗算

円周率世界記録用プログラム作成時には，FMT(高速剰余変換) の理論を完成していなかったため，2 段階 FFT を使用した．FFT と FMT は導出の考えが異なるだけで，計算式は同一である．2 段階 FMT による多数桁の原理と使用メモリ量の比較は,「高速剰余変換による多数桁乗算」の 2 段階 FMT による多数桁乗算の節に記載した．また，具体的計算方法は「高速剰余変換による多数桁分割乗算」の 2 段階 FMT による分割乗算の節に記載した．そのため，ここでは説明を省略する．本 2 段階 FFT による多数桁乗算は，上位 FFT は分割しないで正巡回乗算と負巡回乗算の 2 回実施し，元の多数桁乗算にする方式を使用した．この方式を採用した理由は，メモリ量の削減と誤演算した場合のチェックが可能なためである．誤演算のチェックは，正巡回乗算と負巡回乗算の結果の一部が理論的に一致するため，その部分が計算時に一致するかチェックすることで実施した．また下位 FFT による多数桁乗算には，実 FFT による多数桁乗算方式を使用した．

6.4.3　有効ビットの確保

16 進表示の 1 兆桁 (10 進) を，実 FFT でそのまま乗算すると下記の様に，乗算結果の 1 要素の最大値が倍精度浮動小数点の精度（2^{53}）を超える．即ち，入力の 1 要素に 1 桁入れても乗算結果の 1 要素の値が，最大で要素数× 16 進 2 桁となり丸め誤差の累積（最大で 16 進 2 桁程度）を考慮すると，倍精度浮動小数点（2^{53} 精度）で正しく求まらない．

結果の 1 要素の最大値 = 要素数× 1 要素桁数の 2 乗×丸め誤差の累積

$10^{12} \cdot 16^2 \cdot 16^2 > 2^{40} \cdot 2^{16} = 2^{56} > 2^{53}$

この問題点の対策方法として，多数桁を α 進 m 桁単位に要素に分割するとき，各要素の値ごとに正負の値を持たし，各要素の絶対値が $\alpha^m/2$ 以下とするように桁上げ処理を実施する．この方法を用いると，結果の 1 要素の最大値が正負の混合加算により打ち消しあって小さくなり，2061 億桁の π 世界記録の計算では実 FFT に Karatsuba 法を組み合わせて，1 要素に 10 進 3 桁詰めの計算を実現している．16 進の 1 兆桁を実 FFT だけで計算すると仮定すると，1 要素に 16 進 2 桁詰めがぎりぎり可能と思われる．更に，1 要素当たりに保持できる有効ビット数を向上させるために，2 段階 FFT が有効である．2 段階 FFT の上位 FFT は，整数の加減算とシフトだけで構成可能で，正巡回と負巡回乗算を重ね合せることにより，64 ビット整数に 16 進 15 桁（2 進 60 桁）を入れて計算可能となる．そのため，実 FFT で 8 バイト (64 ビット) の倍精度浮動小数点に 16 進 2 桁詰めにすることと比較し，7.5 倍の有効ビットの利用となる．1 兆桁の計算でも 2 段階 FFT の下位 FFT による各乗算は，それぞれ数百万桁と計算が格段に短

い計算となるため，実 FFT 変換のためのメモリ使用量は問題にならない．図 6.1 に 2 段階 FFT による多数桁乗算において，下位実 FFT の倍精度浮動小数点に 2 進 15 桁をつめて計算した時の，結果の実数を整数化するときに発生する最大誤差を示す．図中の◆印は分割した各要素の値を正で 2^{15} 以下とした場合で，×印は各要素の値を正負混合でその絶対値を $2^{15}/2$ 以下とした場合である．横軸の計算桁数は 2 段階 FFT による乗算を使用した場合の桁数であり，実 FFT だけで計算する場合の桁数はその平方根の桁数となる．

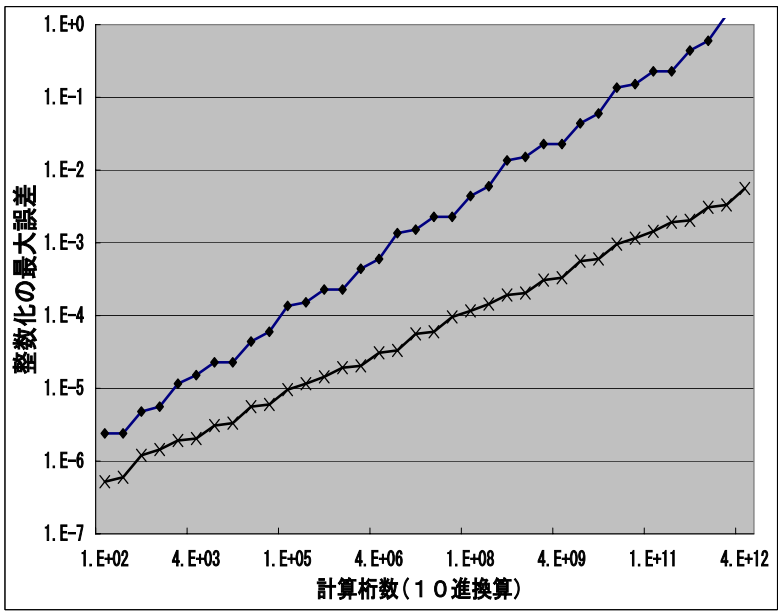

図 6.1: 2 段階 FFT による多数桁乗算の整数化最大誤差

6.5 誤演算自動判定方式

6.5.1 実 FFT 結果の整数化における許容範囲の活用

多数桁の乗算は 2 段階 FFT で計算した．2 段階 FFT は上位 FFT と下位 FFT で構成され，上位 FFT は整数 FFT を使用し下位 FFT は実 FFT を使用した．整数 FFT は文字通り整数計算のため計算誤差は発生しない．一方，実 FFT は倍精度浮動小数点を使用して計算を行うため，整数になるべき乗算結果の要素に丸め誤差が発生して整数でなくなる．一見これは，1 兆桁に達する多数桁の乗算を正確に実行するには問題の様に思われる．実 FFT による多数桁乗算の倍精度浮動小数点結果を四捨五入し整数化し

た値を乗算結果とした．計算が正しく行われたかを監視するため，全ての実FFTによる多数桁計算における整数化誤差の最大値を求めて，DRM法の分割計算ごとに出力した．また，この整数化誤差の最大値が0.1を超えると計算が停止する仕組みにした．1兆桁のπ計算では，実FFTによる多数桁乗算結果の合計要素数は10^{14}を超える．このため，全ての乗算結果の要素の整数化誤差の最大値が0.1以下で，誤って整数化する確立は10^{-140}以下と，天文学的に小さな数となる．逆に，実FFTの計算時におけるハード及びソフトによる計算不正や，FFTのsin，cosのテーブルを作成する部分の精度が悪い場合はこの値が0.5に近い値となる．実際に，sin，cosの精度不良や，ハード計算不正がテスト段階で発生し，これにより結果不正を発見し対策している．

6.5.2 正及び負巡回FFTの利用

2段階FFTによる$2n$要素の多数桁乗算結果$C_i, (i = 0, 1, \cdots, 2n-1)$は，$n$要素の正巡回乗算結果$A_i, (i = 0, 1, \cdots, n-1)$と負巡回乗算結果$B_i, (i = 0, 1, \cdots, n-1)$を計算し,$C$の前半部分を$C_i = (A_i + B_i)/2, (i = 0, 1, \cdots, n-1)$で後半部分を$C_{i+n} = (A_i - B_i)/2, (i = 0, 1, \cdots, n-1)$で求めた．このとき，$A_0$と$B_0$は理論的に同一の値となる．この性質を利用して，$A_0$と$B_0$の一致性を誤演算判定に利用した．2段階FFTの上位FFTの1要素は最大で16進約100万桁となる．また，FFTの性質より，途中計算で発生した不正結果は最初の要素A_0，B_0の結果に影響する．このため，多数桁乗算におけるこの誤演算判定は実に良く機能した．多数桁乗算のプログラム不良（並列化不良を含む），精度不足及び通信不正をテスト段階で検出した．ハード計算不正も検出している．

6.5.3 I/Oの誤り防止対策

Arctan公式による16進π計算及び16進10進と10進16進変換には，多量のDiskファイルI/Oを利用した．そのため，I/Oの高速化と共にI/O誤りには細心の注意を払った．データを記憶するファイル以外に，I/O時に誤りが発生したときのチェック用のファイルを別に用意した．チェック用ファイルには，データファイルの種類，名称，サイズ等の他に64ビット整数単位に，データファイルの出力時にロジカルaddと排他的orによる合計値を求めて記録し，データファイルの入力時に，記録した二つの合計値によるチェックを行い，一致しない場合には計算を中止する仕組みにした．

6.6 世界記録達成の経過

6.6.1 記録挑戦への 5 年間の経過

DRM 法の考案と論文化

分割有理数化法 (DRM 法) の基本アイデアを 1998 年初めに考案し，1998 年 11 月の京都大学数理解析研究所の研究集会「数値計算における前処理の研究」で「逆数型無限級数の n 桁計算のを削減する前処理方式」と題して発表した．1999 年 1 月に同研究所の講究録に「無限級数に基づく多数桁計算の削減を実現する分割有理数化法」と題して投稿し，1999 年 6 月に情報処理学会論文誌に「級数に基づく多数桁計算の削減を実現する分割 有理数化法」と題して投稿し，2000 年 6 月号に掲載．1998 年の京大発表では，入力値が $O(1)$ 桁の有理数の無限級数の値を多数桁で求める方式であった．その後の論文化で，加法定理が適用できる級数関数に対して，入力値が結果と同じ n 桁の多数桁の場合の計算方法，連分数の計算及び基底変換への DRM 法の適用方法を示した．1999 年末に，2T バイトのメモリの計算機による 1 兆桁の π の可能性について検討を開始した．

DRM 法による π 計算用プログラム開発

2T バイトのメモリの計算機で DRM 法を適用し，1 兆桁の π 計算が可能との見通しが立ち，1999 年夏から Ramanujan 型公式による π 計算の原理プログラムの作成を開始した．最初は 10 進でファイル I/O を利用しないプログラムを作成．多数桁乗算は実 FFT ＋ Karatsuba 法及び整数 FFT 結果の中国剰余定理による重ね合わせなど多数の試作プログラムを作成した．DRM 法の計算主体には効率の点から，ファイル I/O を利用しない方針としたため，1 兆桁の計算にメモリが不足すると判明した．そこで π 計算は 16 進で計算し，計算結果を 10 進に変換することにした．当初，使用する計算機 (SR8000/MPP) の特性を考慮し，多数桁乗算は整数 FFT の重ね合わせが有利と判断した．SR8000 は倍精度浮動小数点の乗算が演算器内において，4 倍精度浮動小数点として表される (浮動小数点レジスタには取り出せない) ことを上手に利用すると，A, B, C を倍精度浮動小数点数として，$A \cdot B \pmod{C}$ が高速に計算可能である．但し，この処理は FORTRAN や C 言語では上手に利用できないため，アセンブラーで記述する必要がある．この機能を利用した多数桁乗算方式を試作し，性能テストを実施した．この方式は，メモリ使用面及び速度の面でも有効と判断した．しかし，この方式は SR8000 でだけ有効で，アセンブラー記述のため変更が面倒なことがあり，パソコンを使用した 1 兆桁の π 計算プログラムの開発では効率が悪いと判断し，採用を断念した．次に，実 FFT ＋ Karatsuba 法による多数桁の乗算を検討したが，メモリ効

率を上げるためにはKaratsuba法の適用回数を5回にする必要があり，Karatsuba法は1回適用すると計算量が1.5倍増加するため，効率の面から断念した．そこで，2段階FFTによる多数桁乗算方式を考案した．メモリ面でも，速度面でも1兆桁計算には効率的と判断し，2段階FFTを採用することにした．2001年初めにSR8000の1ノードに移植し，2段階FFTの上位FFTはパソコンの32ビット整数から64ビット整数に変更し，ノード内並列化 (メモリ共用8プロセッサーの並列化) を実施した．2001年4月から，東京大学の金田康正教授をプロジェクトリーダとする，東京大学金田研究室と日立製作所の共同プロジェクトを発足させ，SR8000/MPPによる並列化とチューニングを実施した．16進10進及び10進16進変換のプログラムの開発は少し遅れて実施した．

Ramanujan型公式による2001年の挑戦

2001年9月にRamanujan型の3公式の並列化プログラムが完成し，1兆桁π計算を開始した．本計算には，SR8000/MPPの128ノード (2Tバイトのメモリ) で一番高速なChudnovsky公式を使用し，16進1兆桁の計算を実施した．次に，128ノードが使用不可能のため64ノード (1Tバイト) で二番目に高速なRamanujan公式を使用し，16進5千億桁を計算した．その結果，約3597億以降で16進の値が不一致となり，2001年の計算は断念した．約2ヶ月の机上調査の結果，初期値作成の精度不足によるプログラムの誤りと判明した．

Arctan公式による2002年の挑戦

SR8000/MPPの128ノード (2Tバイト) の使用は諦め，半分の64ノード (1Tバイト) を使用する方針とした．このため，Chudnovsky公式に比較して約3倍計算量が必要であるが，最後に1兆桁同士の除算が不要で，メモリの削減が容易なArctan公式を採用することにした．正計算の計算時間は，計算機能力が半分で，計算量が約3倍になり，ファイルI/Oを本格的に使用するため，2001年に比べて約6倍以上が必要となる．しかし，これはプログラムのチューニングを徹底して行うことで，4倍以内に削減することを目標にした．本プログラムのパソコン版は2001年末に完成していた．2001年6月までは，パソコン用16進10進及び10進16進変換のプログラムの開発を実施した．プロジェクトとして，本格的にArctan公式による16進π計算プログラムの並列化及びチューニングを開始したのは7月からである．多数桁乗算やDRM法による通分化等の並列化の基本部品は2001年のものがそのまま使用できるが，64ノード計算でメモリ使用量が半減するため，DiskへのI/Oの量が桁違いに増加した．Diskの16台並列I/Oの性能を最大限引き出すための工夫は開発したπ計算のプログラムだけで

なく，OS や Disk コントローラの最適化も含めて実施した．その結果，16 台の Disk のカタログ性能の約 9 割の $1.1GB/s$ の I/O 性能をが並列 I/O で達成できた．これでも，Disk の I/O 時間は 16 進 1 兆桁の正計算時間 400 時間の 1/4 の 100 時間を必要とした．

6.6.2 利用公式

主計算は高野喜久雄 (1982 年) の式 (6.7) を，検証計算は F.C.M.Stormer(1896 年) の式 (6.8) の公式を利用した．両式で arctan(1/57) 及び arctan(1/239) が共通に存在しその係数はそれぞれ異なるため，検証計算を含めた総計算時間が短くなることで両公式を利用した．式 (6.8) の方が式 (6.7) より収束が速く，より高速に計算できるが検証時間も含めた総計算時間は変らないため，日本人の発見した公式を主計算に使用した．

$$\begin{aligned}\frac{\pi}{4} &= 12\arctan(\frac{1}{49}) + 32\arctan(\frac{1}{57}) - 5\arctan(\frac{1}{239}) \\ &\quad +12\arctan(\frac{1}{110443})\end{aligned} \tag{6.7}$$

$$\begin{aligned}\frac{\pi}{4} &= 44\arctan(\frac{1}{57}) + 7\arctan(\frac{1}{239}) - 12\arctan(\frac{1}{682}) \\ &\quad +24\arctan(\frac{1}{12943})\end{aligned} \tag{6.8}$$

arctan の級数展開式は式 (6.9) を利用した．

$$\arctan(\frac{1}{x}) = \frac{1}{x} - \frac{1}{3x^3} + \frac{1}{5x^5} - \frac{1}{7x^7} + \frac{1}{9x^9}\cdots \tag{6.9}$$

6.6.3 計算経過

東大情報基盤センターの Hitachi SR8000/MPP の 64 ノードを使用して 2002 年 9 月から 11 月に行なった．SR8000/MPP の 1 ノードは 8 プロセッサーで，16GB のメモリと最大 14.4GFLOPS の性能を有する．ノード間は 3 次元クロスバーで接続され，64 ノードで合計 1TB のメモリ容量となる．π の主計算と検証計算は 16 進数で計算し，16 進数の結果を 10 進数に基底変換する方法を採用した．16 進数の π 計算結果の正しさは，独立な π の Arctan の 2 公式に基づくそれぞれの計算結果を比較して確認し，16 進 10 進への基底変換の正しさは，16 進入力と同じ結果が，16 進 10 進及び 10 進 16 進

の連続する 2 つの基底変換により得られることで確認した．16 進での π の計算は主計算，検証計算共に 1,030,775,439,375 桁で実施した．2 種類の計算結果は最後の桁まで一致した．その理由は式 (6.3) の C-Type 浮動小数点で各 Arctan の値（すべて 1 以下）を計算し，整数係数を掛けて加え π(1 以上) の値を計算したことで，各 Arctan の値を結果的に 100 万桁多くなっていたためである．16 進 π の公表桁数は最後の 75,439,375 桁を除いた 1 兆 307 億桁とした．10 進 16 進への基底変換は 1,241,177,304,180 で実施した．10 進 π の公表桁数は億の単位で 1 兆 2411 億桁とした．以下に 16 進 π の計算及び 10 進数への変換過程を具体的に示す．主計算においては，16 パスを持つ Disk(磁気記憶装置）への性能を調査する目的で，入出力量及び入出力時間を測定した．

16 進 π の計算

(1) 主計算
開始日時： 2002 年 9 月 3 日 22 時 9 分
終了日時： 2002 年 9 月 25 日 17 時 0 分
計算時間： 1,439,973 秒 : 約 400 時間 0 分
中断回数： 5 回
Disk への入出力量： 395,000 GB
Disk への入出力性能： 1.1 GB/s
Disk への入出力時間： 100 時間（計算時間の内数）

(2) 検証計算
開始日時： 2002 年 9 月 26 日 16 時 46 分
終了日時： 2002 年 10 月 4 日 10 時 40 分
計算時間： 565,468 秒 : 約 157 時間 4 分
中断回数： 4 回

(3) 結果の検証
開始日時： 2002 年 10 月 4 日 12 時 47 分
終了日時： 2002 年 10 月 4 日 13 時 58 分
計算時間： 4,275 秒 : 約 1 時間 11 分

10 進数への変換

(1) 10 進数への変換
開始日時： 2002 年 11 月 15 日 13 時 41 分
終了日時： 2002 年 11 月 16 日 13 時 1 分
計算時間： 84,019 秒 : 約 23 時間 20 分

中断回数： 0 回

(2) 16 進数への逆変換

開始日時： 2002 年 11 月 23 日 16 時 7 分

終了日時： 2002 年 11 月 24 日 13 時 39 分

計算時間： 77,540 秒：約 21 時間 32 分

中断回数： 0 回

(3) 結果の検証

開始日時： 2002 年 11 月 24 日 13 時 48 分

終了日時： 2002 年 10 月 24 日 15 時 07 分

計算時間： 4,661 秒：約 1 時間 18 分

6.6.4 計算結果

16 進計算結果

(1) 計算桁の手前の 50 桁

小数点以下 1,030,775,439,357 桁まで計算し，一致を確認した．

1,030,775,439,308〜1,030,775,439,357 桁までの 50 桁は下記の通り．

```
B556AACBA8 4A22B25C79 26AB3CD02F 532B7E71EE 4E10E249E5
```

(2) 宣言桁の手前の 50 桁

計算宣言桁数は小数点以下 1 兆 307 億桁とした．

宣言桁の手前の 50 桁は下記の通り．1 兆 307 億桁目は 0 である．

```
28C7C6A1DC 75E2D2B765 2D23340DAE 5573D62B57 1BAF9CC340
```

(3) 1 兆桁の手前の 50 桁

1 兆桁の手前の 50 桁は下記の通り．1 兆桁目は 5 である．

```
CDDEE9D16E B4A295E569 54E5E30B51 24A9EA02A9 3F89341CD5
```

(4) 最初の 500 桁 (最初の 3 は桁数に含めず)

```
3.243F6A8885 A308D31319 8A2E037073 44A4093822 299F31D008
2EFA98EC4E 6C89452821 E638D01377 BE5466CF34 E90C6CC0AC
29B7C97C50 DD3F84D5B5 B547091792 16D5D98979 FB1BD1310B
A698DFB5AC 2FFD72DBD0 1ADFB7B8E1 AFED6A267E 96BA7C9045
F12C7F9924 A19947B391 6CF70801F2 E2858EFC16 636920D871
574E69A458 FEA3F4933D 7E0D95748F 728EB65871 8BCD588215
4AEE7B54A4 1DC25A59B5 9C30D5392A F26013C5D1 B023286085
F0CA417918 B8DB38EF8E 79DCB0603A 180E6C9E0E 8BB01E8A3E
D71577C1BD 314B2778AF 2FDA55605C 60E65525F3 AA55AB9457
```

48986263E8 144055CA39 6A2AAB10B6 B4CC5C3411 41E8CEA154

10 進計算結果

(1) 計算桁の手前の 50 桁

小数点以下 1,241,177,304,180 桁まで計算し，一致を確認した．

1,241,177,304,131〜1,241,177,304,180 桁までの 50 桁は下記の通り．

8508566667 3937845843 1454411298 9344841413 3413557641

(2) 宣言桁の手前の 50 桁

計算宣言桁数は小数点以下 1 兆 2411 億桁とした．

宣言桁の手前の 50 桁は下記の通り．1 兆 2411 億桁目は 5 である．

5591198918 2262704528 2696896699 2856706487 3410311045

(3) 1 兆桁の手前の 50 桁

1 兆桁の手前の 50 桁は下記の通り．1 兆桁目は 2 である．

2976735807 0882130902 2460461146 5810642210 6680122702

(4) 最初の 500 桁 (最初の 3 は桁数に含めず)

3.1415926535 8979323846 2643383279 5028841971 6939937510
5820974944 5923078164 0628620899 8628034825 3421170679
8214808651 3282306647 0938446095 5058223172 5359408128
4811174502 8410270193 8521105559 6446229489 5493038196
4428810975 6659334461 2847564823 3786783165 2712019091
4564856692 3460348610 4543266482 1339360726 0249141273
7245870066 0631558817 4881520920 9628292540 9171536436
7892590360 0113305305 4882046652 1384146951 9415116094
3305727036 5759591953 0921861173 8193261179 3105118548
0744623799 6274956735 1885752724 8912279381 8301194912

6.6.5 プログラムの規模

プログラム作成言語は FORTRAN で一部に C 言語を利用した．ノード内 8 プロセッサーの並列化は FORTRAN の自動並列化機能を利用した．但し，一部の計算はプログラムの上位から並列化した．ノード間の並列化は SR8000 専用のノード間通信機能（リモート DMA）と汎用機能の通信ライブラリ（MPI）の 2 つの通信ライブラリで行なった．リモート DMA は主に 2 段階 FFT による多数桁乗算の並列化に利用し，MPI は多数桁の加算や桁上げ処理等に利用した．プログラムは Arctan による 16 進 π 計算，計算した 16 進 π の 10 進 π への変換及び，検証のため 10 進 π を 16 進に変換する 3 部に

分けて作成した．Arctan による 16 進 π 計算は，正計算及び検証計算に使用した 2 公式以外に 6 公式を組み込み，全公式とも複素ノードで数十億桁の計算まで一致することを確認した．下記に各プログラムの規模を行数で示す．本プログラム行数には 2001 年に行なった Ramanujan 型公式の 16 進 π 計算プログラムは含まれていない．プログラム行数はコメント文も含めた行数である．規模がこれほど大きくなった原因は，極限に近いチューニング及びメモリ使用量を少なくするための各種工夫を多く行なったためである．

(a) 16 進 π 計算 (正，検証)： 5 万 4600 行
(b) π の 16 進 10 進変換　： 1 万 4600 行
(c) π の 10 進 10 進変換　： 1 万行
(d)　　合　計　　： 7 万 9200 行

6.7　前年度の経過と計算誤りの原因追求

6.7.1　2001 年世界記録挑戦への経過

主計算は SR8000/MPP 128 ノードを使用して 16 進 1 兆桁を計算し，検証計算は計算機利用上の都合で 64 ノードを使用して 16 進 5 千億桁を計算し，両者の結果を比較した．著者の精度に関するプログラムミスで，3597 億余桁以降が不一致となり 2001 年の挑戦は諦めざるを得なかった．主計算に式 (6.1) の Chudnovsky 公式を，検証計算に式 (6.2)Ramanujan の公式を使用した．共に，$1/\pi$ が求まる公式のため，DRM 法による級数計算の部分と，除算して π を求める部分に分割して計算した．

16 進 1 兆桁の主計算

SR8000/MPP 128 ノード (2T バイトメモリ) を使用して計算した．

(1) DRM 法による級数計算
開始日時： 2001 年 9 月 16 日 4 時 15 分
終了日時： 2001 年 9 月 22 日 19 時 2 分
計算時間： 343,652 秒 : 約 95 時間 28 分
中断回数： 4 回

(2) 除算して 16 進 π を求める計算
開始日時： 2001 年 10 月 6 日 0 時 39 分
終了日時： 2001 年 10 月 6 日 17 時 41 分
計算時間： 61,296 秒 : 約 17 時間 2 分
中断回数： 0 回

(3) 合計計算時間
343,652 + 61,296 = 404,948 秒： 約 112 時間 29 分

16 進 5 千億桁の検証計算

プログラムの確認等を行なう目的で，5 千億桁を 64 ノードで (1T バイト) 計算した．
(1) DRM 法による級数計算
開始日時： 2001 年 10 月 9 日 21 時 49 分
終了日時： 2001 年 10 月 14 日 19 時 4 分
計算時間： 354,443 秒 : 約 98 時間 27 分
中断回数： 2 回
(2) 除算して 16 進 π を求める計算
開始日時： 2001 年 10 月 15 日 1 時 56 分
終了日時： 2001 年 10 月 17 日 9 時 43 分
計算時間： 51,594 秒 : 約 14 時間 20 分
中断回数： 1 回
(3) 合計計算時間
354,443 + 51,594 = 406,037 秒： 約 112 時間 47 分

6.7.2 2001 年計算結果

2001 年 10 月 19 日に主計算と検証計算の 16 進の結果比較を行なったところ，359,712,809,450 桁以降が異なっていることが判明した．この時点では，正計算と検証計算のどちらが誤っているかは不明であった．約 2ヶ月の机上での調査の結果，両計算ともに異なる桁数から誤っていることが判明した．両計算の結果が 359,712,809,450 桁以降で異なるのは正計算の結果が誤りと判明した．両計算の 359,712,809,441〜359,712,809,460 桁までの 20 桁は下記の通りであった．

主計算 (誤)　： `8234700EE3 3B42B4CB6C`
検証計算 (正)： `8234700EE0 B6CC5EE098`

6.7.3 計算誤り原因調査と発生桁の推定

約 2ヶ月の机上での不良原因調査の結果，2001 年 12 月 6 日に不良原因が判明した．原因は単純な精度に関するプログラムミスであった．計算精度に対して最も注意を必要とし複雑な計算が要求される，2 段階 FFT による乗算とその Arctan 公式への適用部分

で，ビット落ちによる結果不正が発生してるとの思い込みにより，その部分の利用ビット範囲のチェックに１ヶ月以上を要した．しかし，この部分は十分タフに設計しプログラム実装しており，万一発生してもプログラムでの自動精度チェックにかかる確率が非常に高いことが判明した．このため，視点を変えて調査を実施した．その結果，級数の各項の初期値を作成するプログラムにビット落ちの不良が発生することが判明した．原因は k 番目の級数の項の計算において，式 (6.1) 及び式 (6.2) で発生する $9k$ や $4k-1$ 等の数値を，2 個の倍精度浮動小数点に分けて表示すべきところを 1 個で済ませてしまった単純ミスである．k 番目の級数の項の値は $2k-1$，$6k-5$，$9k$，$4k-1$ 等の値を倍精度浮動小数点で表し，途中の乗算結果は，2 進 15 桁単位で桁上げし，複素の倍精度浮動小数点で表していた．IEEE の倍精度浮動小数点の仮数部は 52 ビットで構成されている．また，正規化表示のためゼロ以外の値は先頭に 1 ビット有効ビットが仮定される．このため，正確に表示される整数の絶対値は $2^{53}-1$ である．これを超えた整数はビット落ちのため正しい値とならない．即ち，乗算結果が $2^{53}-1$ を超えると，この初期値作成プログラムは不正となる．プログラムでは，式 (6.1) の Chudnovsky 公式を ID=0 とし，式 (6.2) の Ramanujan の公式を ID=1 と使用しており，以下この記号で説明する．k 番目の級数の項の計算において定数を除き，ID=0 では $9k$ が最大の値で，ID=1 では $4k-1$ が最大の値となる．これらの値と最大 2 進 15 の値の乗算結果が倍精度浮動小数点で，正確に表現されなければならない．即ち，$9k \cdot 2^{15} < 2^{53}$ 及び $(4k-1) \cdot 2^{15} < 2^{53}$ を満足しなければいけない．逆に，ID=0 では $k > 2^{38}/9$，ID=1 では $k > 2^{38}/4$ となる k 番号の項の計算は結果不正が発生する．これは，ID=0 で $k = 30,541,989,660$ 番以降の項に，ID=1 で $k = 68,719,476,737$ 番以降の項になる．実際は，2 進 15 桁に桁上げした乗算相手が 2 進 15 桁の最大値とはならないため，パソコンを使用して乗算結果が $2^{53}-1$ を超える k の値を算出した．その値は ID=0 で $k = 30,542,106,126$，ID=1 で $k = 68,719,907,265$ となる．また，ID=0 は 1 項の計算で 16 進 $\log_{16} 53360^3$ 桁計算でき，ID=1 は $\log_{16} 99^4$ 桁計算できる．このため，不正発生桁数は項数と 1 項当りの桁数を掛け，ID=0 で 359,712,809,427 桁，ID=1 で 455,568,772,158 となる．不正発生項の倍精度浮動小数点の最後のビットが誤っている等で，実際の不正発生桁は上記の桁数より 20〜30 桁多いと推定した．2002 年 10 月 4 日の 16 進 1 兆桁の計算結果と比較し，ID=1 も 455,567,772,187 桁で結果不正が発生していると判明した．不正発生桁と推定桁の差は予測した範囲内で ID=0 では 23 桁，ID=1 では 29 桁であった．

6.8 まとめ

DRM 法の考案による 1 兆桁の π 計算の計画から足掛け 5 年かけ，2002 年 11 月 24 日に 16 進で 1 兆 307 億桁，10 進で 1 兆 2411 億桁の世界記録の達成に成功した．本計

算は 1999 年の 2061 億桁の世界記録の計算に比較し，同一メモリ容量の計算機で 6 倍の桁数を計算した．16 進 π の値は Arctan 公式に DRM 法を使用して計算し，16 進 π の結果を DRM 法で 10 進に変換する方法を使用した．本計算の目的の一つは，多数桁級数計算及び基底変換における DRM 法の評価であり，目的は十分達成されたと考える．円周率の世界記録の計算は，自動車の F-1 レースの様に限界に近い計算機のハード，ソフトのテストに対応する．そのため，通常の使用では発生しないハード及びソフトによる計算誤りが発生している．そのため，出荷前の計算機システムの信頼性テストプログラムとして使用されている．

第7章　おわりに

7.1　まとめ

本研究で考案した多数桁計算における高速計算アルゴリズムは
(1) 高速剰余変換（Fast Modulo Transformation，FMT）
(2) 分割有理数化法（Divide and Rationalize Method，DRM 法）
(3) 多倍長精度の係数を持つ行列乗算
である．FMT は多数桁乗算への適用において結果的に FFT と同じ計算式となるが，多数桁乗算における途中変換の意味が明確という利点を持つ．FMT を使用した多数桁の乗算への応用として，複素 FMT を直接利用した乗算，自然数 α に対して $\pm\alpha$ で巡回する乗算，2 段階 FMT による乗算及び分割乗算を考案した．2 段階 FMT による乗算は 1 兆桁の多数桁乗算で，実 FFT だけによる乗算より約 7 倍のメモリ利用効率の向上となり，2002 年の円周率世界記録達成に寄与した．DRM 法は級数関数の多数桁関数値の計算のために考案したものであり，基底変換や連分数の計算にも適用可能で，関数の多数桁計算で著名な Brent のアルゴリズムより適用範囲が広く，アルゴリズムが単純で分り易い特長がある．また，DRM 法は入力値が $O(1)$ の有利数の級数関数だけでなく，入力値が結果の桁数 n と同じ実数の級数関数への適用できることを示した．これは，多数桁関数計算においてシステム的に閉じるために重要なことと考える．多倍長精度の係数を持つ行列乗算は，連立一次方程式の 4 倍精度以上の桁数の解の計算の高速化のための基本アルゴリズムとして研究した．行列乗算への中国剰余定理及び FFT の適用において，それらの線形性に着目し，n 次元の行列乗算で中国剰余定理及び FFT の適用回数を $O(n^3)$ から $O(n^2)$ に削減することに成功した．その結果，多数桁の行列乗算に中国剰余定理適用する方式で，4 倍精度で既に筆算方式より高速になり，桁数が比較的短いときに中国剰余定理が有理で，桁数が長くなると FFT が有理となることが判明した．考案した FMT と DRM 法を実際に評価するための数値実験を行なった．FMT は多数桁の分割乗算に適用し，DRM 法は 10 進 1.2 兆桁の円周率世界記録計算に適用した．FMT を適用した分割乗算では，m 分割した n 桁の乗算の計算量が $O(n \cdot \log n \cdot (\log\log n))$ で，ファイル I/O の総量が $O(n)$ となり，共に分割前と同一にすることができた．これは，FFT による多数桁の計算を m 分割すると，共に分割前の m 倍になるとの常識を覆すものである．2001 年 11 月に達成した，16 進 1 兆 307

億桁及び 10 進 2411 億桁の世界記録は，1999 年の 10 進 2061 億桁の記録を同じメモリ量 (1TB) の計算機で 6 倍更新した．そのため，それまで世界記録更新に使用されてきた AGM 法 (算術幾何平均化法，ガウス・ルジャンドル公式) から Arctan 公式に変更し，DRM 法と 2 段階 FMT(FFT) による多数桁乗算を使用した．また従来とは異なり，Arctan 公式で 16 進の π の値を計算し，16 進 π の結果を 10 進に変換する方式を採用した．これにより，DRM 法の多数桁級数関数値計算及び基底変換への高速適用性が確認できた．以上から本研究の目的である，多数桁で構成される計算システムの基となる高速計算アルゴリズムの確立は達成できたと考える．

7.2 今後の課題

多数桁計算における今後の課題として下記の事項がある．

(1) DRM 法に基づく高速高精度な数学関数計算パッケージの作成と性能評価
(2) 高速行列乗算を適用した連立一次方程式の多倍長精度の解の計算
(3) 分割乗算を適用した，次期円周率の世界記録の達成

第8章　研究業績

8.1　論文

(1) ○後 保範：高速剰余変換による多数桁乗算，情報処理学会論文誌，Vol. 44, No. 12, pp. 3131–3138(2003).

(2) ○後 保範, 金田康正, 高橋大介： 級数に基づく多数桁計算の演算量削減を実現する分割有理数化法，情報処理学会論文誌，Vol. 41, No. 6, pp. 1811–1819 (2000).

(3) 小国 力, 後 保範, 長堀文子： スーパーコンピュータにおけるリストベクトルの利用技術, 情報処理学会論文誌, Vol.27 No.1, pp. 11–19 (1986).

(4) 小国 力, 後 保範, 西方政春, 長堀文子： 直接解法による連立一次方程式のコンピュータ解法の特性解析, 情報処理学会論文誌, Vol.25 No.5, pp. 804–812(1984).

8.2　総説

(1) ○後 保範： 多倍長精度の値を係数とする行列の高速乗算方式, 京大数理研講究録 [偏微分方程式の数値解法とその周辺 II], 1198, pp. 170-178 (2001).

(2) ○後 保範, 金田康正，高橋大介： 無限級数に基づく多数桁計算の演算量削減を実現する分割有理数化法，京都大学数理解析研究所講究録　No.1084，pp. 60–71 (1999).

(3) 田村良明，吉野さやか，後 保範，金田康正：円周率-高速計算法と数値の統計性，情報処理学会題 25 回プログラミングシンポジューム報告集，pp. 1–15 (1984)

(4) Y.Kanada, Y.Tamura, S.Yoshino, Y.Ushiro： Calculation of π to 10,013,395 Decimal Placed Based on the Gauss-Legendre Algorithm and Gauss Arctangent Relation，Computer Center University of Tokyo Technical Report, pp. 1–13 (1983).

8.3 講演

(1) ○後 保範： 円周率世界記録と多数桁乗算方式, 早稲田大学理工学部，戸田セミナー (2004/7/17)

(2) ○後 保範: 高速剰余変換 (FMT) の応用, 第 32 回数値解析シンポジウム (NAS2003) (2003/5/21-23)

(3) ○後 保範: 円周率世界記録 6 倍更新の工夫, 早稲田大学 7F セミナー (2003/5/16)

(4) ○後 保範： FFT と百五減算による多数桁乗算, 第 31 回数値解析シンポジウム (NAS2002) (2002/6/12-14)

(5) ○後 保範： 反復解法への多倍長精度高速計算の適用, 第 30 回数値解析シンポジウム (NAS2001) (2001/5/23-25)

(6) ○後 保範： 多倍長精度の値を係数とする行列の高速乗算方式, 京大数理解析研究所 [偏微分方程式の数値解法とその周辺 II] (2000/11/20-22)

(7) ○後 保範: 多倍長精度の行列乗算方式, 第 29 回数値解析シンポジウム (NAS200) (2000/6/7-9)

(8) ○後 保範： 逆数型無限級数の n 桁計算の演算量を削減する前処理方式, 京大数理解析研究所 [数値計算における前処理の研究] (1998/11/9)

(9) ○後 保範, 長堀文子： ベクトル計算機による整数上の FFT 計算, 情報処理学会全国大会, No. 27, pp. 1293–1294 (1983).

8.4 その他

8.4.1 論文

(1) Ogita,T., Oishi,S., Ushiro,Y.: Computation of Sharp Rigorous Component wise Error Bounds for the Approximate Solutions of Systems of Linear Equations, Reliable Computing Vol.9, pp.229-239 (2003).

(2) Ogita,T., Oishi,S., Ushiro,Y.: Fast inclusion and residual iteration for solutions of matrix equations, Computing [Supplement 16, Inclusion Methods for Nonlinear Problems], pp.171-184 (2002).

(3) Ogita,T., Oishi,S., Ushiro,Y.： Fast verification of solutions for sparse monotone matrix equations, Computing [Supplement 15, Topics in Numerical Analysis], pp.175-187 (2001).

8.4.2 総説

(1) ○後 保範： 行列乗算におけるストラッセンの方法の拡張, 京大数理研講究録 [数値計算アルゴリズムの研究] 1040, pp.61-99 (1998).

(2) ○後 保範： 有限要素法の連立一次方程式ベクトル並列向き反復解法, 京大数理研講究録 [数値解析とそのアルゴリズム] 791, pp.49-62 (1992).

(3) ○後 保範： 対称用改定 CR 法の収束性, 京大数理研講究録 [数値解析と計算科学]746, pp.37-46 (1991).

(4) ○後 保範： ナビェストーク方程式への連立 ILUCGS 法適用効果, 京大数理研講究録 [数値解析と科学計算] 717, pp.104-117 (1990).

(5) ○後 保範： 偏微分方程式向き数値シミュレーション言語 DEQSOL, 第 3 回ベクトル計算機応用シンポジウム論文集, pp.115-123 (1987).

(6) ○後 保範, 今野千里, 他 3： DEQSOL とスーパーコンピュータ向け応用ソフト, 日立評論 [スーパーコンピュータに関する論文集], Vol.69 No.12, pp.35-42 (1987).

(7) ○後 保範, 栂野佳子： スーパーコンピュータ S-810 による拡散方程式の数値計算, 京大数理研講究録 [スーパーコンピュータのための数値計算アルゴリズムの研究] 613, pp.12-27 (1987).

(8) ○後 保範: 大型疎行列に対する PCG,PCR 法, コンピュートロール, No.12, pp.16-22 (1985).

(9) ○後 保範： 移流拡散方程式に対するベクトル向き PCG 法, 京大数理研講究録 [大型の線形計算に関する研究] 548, pp.147-168 (1985).

(10) ○後 保範： ベクトル計算機向き ICCG 法, 京大数理研講究録 [並列数値計算アルゴリズムとその周辺] 514, pp.110-134 (1984).

(11) ○後 保範, 西方正春, 長堀文子： スーパーコンピュータ”HITAC S-810”による行列計算, 日立評論 [最近のコンピュータ技術とスーパーコンピュータに関する論文集], Vol.65 No.8, pp.29-34 (1983).

(12) ○後 保範： ベクトル計算向き不完全三角分解, 京大数理研講究録「数値計算アルゴリズムの研究」453, pp.102-126 (1982).

(13) ○後 保範, 小高俊彦： ベクトル計算機による大規模計算, 情報処理学会 数値解析研究会資料 2, pp.1-10 (1982).

(14) その他 10 件

8.4.3 講演

(1) ○後 保範： 丸め誤差を対称に保った角柱周りの流れ解析, 第 33 回数値解析シンポジウム (NAS2004) (2004/5/19-21)

(2) ○後 保範： ストラッセン法の拡張方式における SR2201 への適用効果, 東京大学地震研究所共同研究集会「大規模計算と並列計算機」(1998/7/6-7)

(3) ○後 保範： 行列計算におけるストラッセンの方法の拡張, 京大数理解析研究所「数値 計算アルゴリズムの研究」(1997/11/26-28)

(4) ○ Ushiro,Y.： Performance for Matrix Computation and its application, Cambridge University (Symposium on high performance computing for the tera-scale information age)　(1996/7/8)

(5) ○後 保範： 有限要素法の連立一次方程式ベクトル並列向き反復解法, 京大数理解析研究所「数値計算アルゴリズムの研究」(1991/11)

(6) ○後 保範： ナビェストーク方程式への連立 ILUCGS 法適用効果, 京大数理解析研究所「数値解析と科学計算」(1989/11/29-31)

(7) ○後 保範： 移流拡散方程式に対するベクトル向き PCG 法, 京大数理解析研究所「大型の線形計算に関するアルゴリズムの研究」(1984/9/6-8)

(8) ○後 保範： スーパーコンピュータ HITACH-S-810 による拡散方程式の数値計算, 電子通信学会全国大会 (1984/9)

(9) ○後 保範： ベクトル計算機向き ICCG 法, 京大数理解析研究所「並列数値計算アルゴリズムとその周辺」(1983/11/24-26)

(10) ○後 保範： CG 法と同時逆反復法の組み合わせによる固有値計算, 京大数理解析研究所「数値計算のアルゴリズムの研究」(1982/11/18-20)

(11) ○後 保範： 不完全三角分解と共役傾斜法による大次元行列の固有値計算, 情報処理学会全国大会 No.22 (1981/4)

(12) その他 19 件

8.4.4 特許

米国特許

(1) ○Ushiro,Y.: Method of and apparatus for preconditioning of a coefficient matrix of simultaneous linear equations. Patent(NO. 5,604,911), February 18, 1997

(2) ○Ushiro,Y., Nagashima,S., Kawabe,S.: Processor for carrying out vector operation wherein the same vector element is used repeatedly in succession. Patent(NO. 4,621,324), November 4, 1986

日本特許

(1) 江澤良孝, マーチン・フィールド, 後 保範： 連立一次方程式に関する計算装置, 特開 2002-149630, 平成 14(2002) 年 5 月 24 日

(2) ○後 保範： 連立一次方程式に関する計算装置，特開平 06-168262, 平成 6 年 (1994)6 月 14 日

(3) 田中慎一, 後 保範： 連立一次方程式の並列計算装置，特開平 06-028387, 平成 6 年 (1994)2 月 4 日

(4) ○後 保範： 連立一次方程式に関する計算装置，特開平 05 - 081310, 平成 5 年 (1993)4 月 2 日

参考文献

[1] 後 保範： 高速剰余変換による多数桁乗算，情報処理学会論文誌，Vol. 44, No. 12, pp. 3131–3138(2003).

[2] 後 保範： 高速剰余変換 (FMT) の応用, 第 32 回数値解析シンポジウム (NAS2003) (2003/5/21-23)

[3] 後 保範： FMT(高速剰余変換)，(2002). http://www.dept.edu.waseda.ac.jp/math/ushiro/ushiro/method/fmt.htm

[4] 後 保範： FFT と百五減算による多数桁乗算, 第 31 回数値解析シンポジウム (NAS2002) (2002/6/12-14)

[5] 後 保範： 多倍長精度の値を係数とする行列の高速乗算方式, 京大数理研講究録 [偏微分方程式の数値解法とその周辺 II], 1198, pp. 170-178 (2001).

[6] 後 保範： 反復解法への多倍長精度高速計算の適用, 第 30 回数値解析シンポジウム (NAS2001) (2001/5/23-25)

[7] 後 保範, 金田康正, 高橋大介： 級数に基づく多数桁計算の演算量削減を実現する分割有理数化法，情報処理学会論文誌，Vol. 41, No. 6, pp. 1811–1819 (2000).

[8] 後 保範, 金田康正，高橋大介：無限級数に基づく多数桁計算の演算量削減を実現する分割有理数化法，京都大学数理解析研究所講究録　No.1084，pp. 60–71 (1999).

[9] 後保範； 逆数型無限級数の n 桁計算の演算量を削減する前処理方式，京大数理研予稿集（数値計算における前処理の研究），p. 9 (1998).

[10] 小国 力, 後 保範, 長堀文子： スーパーコンピュータにおけるリストベクトルの利用技術, 情報処理学会論文誌, Vol.27 No.1, pp. 11–19 (1986).

[11] 小国 力, 後 保範, 西方政春, 長堀文子： 直接解法による連立一次方程式のコンピュータ解法の特性解析, 情報処理学会論文誌, Vol.25 No.5, pp. 804–812(1984).

[12] 後 保範, 長堀文子： ベクトル計算機による整数上の FFT 計算, 情報処理学会全国大会, No. 27, pp. 1293–1294 (1983).

[13] 後 保範： 円周率世界記録と多数桁乗算方式, 早稲田大学理工学部，戸田セミナー (2004/7/17)

[14] 後 保範： 円周率世界記録 6 倍更新の工夫, 早稲田大学 7F セミナー (2003/5/16)

[15] 後 保範： 世界記録のための工夫点, http://www.dept.edu.waseda.ac.jp/math/ushiro/pirecord/pistudy.htm，(2002).

[16] 後 保範： 円周率世界記録樹立，http://ushiro.jp/wrecord.htm, (2002).

[17] 高橋大介, 金田康正： 分散メモリ計算機による円周率の 515 億桁計算, 情報処理学会論文誌, Vol. 39, No. 7, pp. 2074–2083 (1998).

[18] 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，情報処理学会論文誌, Vol. 39, No. 3, pp. 519–528 (1998).

[19] 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，ガウスー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, Vol. 38, No. 11, pp. 2406–2409 (1997).

[20] 田村良明，吉野さやか，後 保範，金田康正： 円周率-高速計算法と数値の統計性，情報処理学会題 25 回プログラミングシンポジューム報告集，pp. 1–15 (1984)

[21] Y.Kanada, Y.Tamura, S.Yoshino, Y.Ushiro： Calculation of π to 10,013,395 Decimal Placed Based on the Gauss-Legendre Algorithm and Gauss Arctangent Relation，Computer Center University of Tokyo Technical Report, pp. 1–13 (1983).

[22] 金田康正： 東京大学金田研究室ホームページ，http://www.super-computing.org/, (2002).

[23] 金田康正： π の話，東京図書，(1991).

[24] Y. Kanada： Vectorization of Multiple-Precision Arithmetic Program and 201,326,000 Decimal Digits of π Calculation, Pi A Source Book, Springer, Copyright 1988 by Scientific American，pp. 576–587 (1989).

[25] Brent, R. P.： Fast Multiple-Precision Evaluation of Elementary Functions，J. ACM No.23，pp. 242–251 (1976).

[26] Chudnovsky, D. V. and Chudnovsky, G. V.：Approximations and Complex Multiplication According to Ramanujan, in， Academic Press Inc., Boston, MA， pp. 375–396 and pp. 468–472 (1988).

[27] Knuth, D. E.： The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, ， Addison-Wesley, Reading, MA 3rd (1997).

[28] Borwein, J. M. and Borwein, P. B.： Pi and the AGM — A Study in Analytic Number Theory and Computational Complexity， Wiley, New York， (1987).

[29] Cooley, J.W. and Tukey, J.W.： An Algorithm for the Machine Calculation of Complex Fourier Series, Math. Comp., Vol.19, pp. 297–301 (1965).

[30] Karatsuba, A. and Ofman, y.：Multiplication of multidigit numbers on automata, Doklady Akad. Nauk SSSR, Vol.145, pp. 293–294 (1962).

[31] Haible, B. and Papanikolaou, T.： Fast multiprecision evaluation of series of rational numbers， Technical Report No. TI-7/97, Darmstadt University of Technology, (http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/papanik/) , pp. 1–15 (1997).

[32] E. Salamin：Computation of π Using Arithmetic-Geometric Mean, Math. Comp., Vol. 30, pp. 565–570 (1976).

[33] Bailey, D.H.: The Computation of π to 29,360,000 Decimal Digits Using Borweins' Quartically Convergent Algorithm, Math. Comp. Vol.50, pp. 283–296 (1988).

[34] Bailey, D. H. and Borwein, J. and Borwein, P.：Ramanujan, Modular Equations, and Approximations to Pi or How to compute one Billion Digits of Pi， American Mathematical Monthly Vol.96， pp. 201–219 (1989).

[35] Sasaki, T. and Kanada, Y.： Parallelism in Algebraic Computation and Parallel Algorithms for Symbolic Linear Systems， Proc. the 1981 ACM Symposium on Symbolic and Algebraic Computation， pp. 160–167 (1981).

[36] 右田剛史, 天野晃, 浅田尚紀, 藤野清次：級数の集約による多倍長数の計算法と π の計算への応用，情報処理学会研究報告 98-HPC-74，pp. 31–36 (1998).

[37] 大浦拓哉： 円周率公式の改良と高速多倍長計算の実装, 情報処理学会研究報告, 98-HPC-74, pp. 25–30 (1998).

[38] V. Strassen：Gaussian Elimination is Not Optimal, Numer. Math. 13, pp. 354–356 (1968).

[39] Gene H. Golb： Matrix Computation, Johns Hopkins University Press, Third edition, pp. 32-34 (1996).

[40] 大野豊，磯田和男： 新版数値計算ハンドブック，オーム社 (1990).

[41] 大石進一： MATLAB による数値計算，培風館 (2001).

[42] 大石進一： 精度保証付き数値計算，コロナ社 (2000).

[43] 小国力，Jack J. Dongarra：MATLAB による線形計算ソフトウエア，丸善 (1998).

[44] 小国 力 外 4： 行列計算ソフトウエア，丸善 (1991).

[45] ウイルキンソン著，一松信他訳: 基本的演算における丸め誤差解析，培風館 (1974).

[46] 戸川隼人： 共役勾配法，教育出版 (1977).

[47] R.S. バーガ著, 渋谷政昭訳： 計算機による大型行列の反復解法，サイエンス社 (1975).