浮動小数点演算ユニットを利用した 4×4 行列式の適応的符号判定処理*

山内俊哉** 吉田典正*** 土井 淳† 山口富士夫††

Adaptive Sign Detection Method of 4×4 Determinants using a Floating Point Processing Unit

Toshiya YAMAUCHI, Norimasa YOSHIDA, Jun DOI and Fujio YAMAGUCHI

In solid modeling systems, the stability of Boolean set operation algorithms is an important issue. The stability can be achieved by using the homogeneous processing, which is proposed by one of the authors, and the exact integer arithmetic. The variable integer data type can represent very large integers by changing the size of the data length. However, the increase of the data length of integers leads to the increase of the computation time. In geometric algorithms, the signs of determinants are necessary to modify the topological data structure. The previous method adaptively and repeatedly extracts 8 bits from variable integer data to detect the signs of 4×4 determinants. The previously proposed method detects the signs of determinants using an FPU. In this paper, a more efficient method is proposed for detecting the signs of 4×4 determinants by extracting 16 bits. The algorithm can detect the signs of 4×4 determinants faster than the previous algorithm.

Key words: solid modeling, Boolean set operation, exact integer arithmetic, geometric algorithm

1. 結 言

ソリッドモデリングの集合演算, Voronoi 図の作成などの幾何アルゴリズムには, 浮動小数点演算の利用が一般的となっている。浮動小数点演算は, 高速に演算をおこなうことができるが, 固定精度演算であるため演算に誤差を伴う。幾何アルゴリズムでは, 幾何判定の結果を用いて位相構造を変更する。したがって, 幾何判定に浮動小数点演算を利用した場合, 数値誤差が原因となって位相情報の矛盾を引き起こし, 幾何アルゴリズムが破綻することがある。

このような問題を解決する手法として, 完全同次処理¹⁾が提案されている。完全同次処理では, 点は同次座標により表現される。同次座標の各成分を可変倍長整数型で表現することにより, 必要十分な有理数の範囲において数値を完全に正確に表現することができる。完全同次処理では除算は必要ないので, 幾何演算を無誤差でおこなうことが可能である。

可変倍長整数型はデータ長を変化させることのできるデータ型であり, 倍長を増やすことにより絶対値の大きな数値データを正確に表現することができる。同次処理に基づく整数演算では除算をおこなわずに加算と乗算を用いて演算をおこなうので, 可変倍長整数演算が繰り返された場合, 数値のデータ長が増大してしまう。それに伴い, 処理時間も増大し, 処理効率が低下するという問題が発生する。

多くの幾何アルゴリズムでは, 行列式の符号判定が頻繁に必要とされる。ほとんどの場合, 行列式の値そのものよりも行列式の符号のみを必要とする。そこで, 行列式の符号を高速に判定するための手法として, 適応的符号判定処理²⁾³⁾が提案されている。適応的符号判定処理ではすべてのデータ長を用いて行列式の符号を計算するのではなく, 数値データの上位データのみを取り出して適応的に演

算結果の符号を求める。適応的符号判定処理は, ほとんどの場合, 可変倍長整数型のデータ長に関係なくほぼ一定の処理時間で符号判定をおこなうことができる。

適応的符号判定処理を更に高速におこなう手法に, FPU(浮動小数点演算ユニット)を利用した手法⁴⁾がある。この手法では, 可変倍長整数型データがある一定の処理単位に分割し, 分割された整数値を浮動小数点型で表現する。分割された数値は浮動小数点型の仮数部を超えないように表現され, そのため, 無誤差で判定がおこなわれる。従来の FPU を利用した適応的符号判定処理では, Intel 系 FPU で採用されている拡張倍精度浮動小数点型(指数部 15 ビット, 仮数部 64 ビット)を用いて, 16 ビット単位でデータを取り出し 3×3 行列式の符号判定をおこなっている。しかし, 4×4 行列式に対して処理単位ビット長を 16 ビット単位とした場合, 計算結果が FPU の仮数部に収まらないため数値誤差を生じてしまう。したがって, 従来の手法では, 4×4 行列式については処理単位を 8 ビット単位としている。FPU を利用した適応的符号判定処理では, 処理単位データ長が大きいほど, 符号を判定できるまでの計算回数を減らすことができ, その結果, より高速に符号判定をおこなうことが可能である。そのため, 4×4 行列式に対しても 16 ビット単位でデータを取り出せることが望ましい。

本論文では, 処理単位を 16 ビット単位として, 4×4 行列式の適応的符号判定処理をおこなう手法を示す。計算結果が FPU の仮数部を超える場合には, 2 つのレジスタを用いて数値を正確に保持する。これにより, 処理単位を 16 ビット単位としても, 無誤差で符号判定をおこなうことができる。この手法は, 処理単位を 8 ビット単位とした場合に比べ, より高速に処理をおこなうことが可能となる。

2. 4×4 行列式の適応的符号判定処理

2.1 行列式の適応的符号判定

行列式の適応的符号判定処理では, 行列成分の可変倍長整数型データの上位からある一定ビット長の処理単位を取り出し, それを用いて行列式 D の近似値 D_{approx} を計算する。そして, その近似

* 原稿受付 平成 11 年 10 月 20 日

** 学生会員 早稲田大学大学院 (東京都新宿区大久保 3-4-1)

*** 正 会 員 東京農工大学工学部 (東京都小金井市中町 2-24-16)

† 日本アイ・ビー・エム(株)東京基礎研究所 (大和市下鶴間 1623-14)

†† 正 会 員 早稲田大学理工学部

Fig. 1 Adaptive sign detection

により生じる誤差を適切に見積もる。この値を誤差限界値 (Error Bound) と呼び、近似値 D_{approx} と誤差限界値 EB との関係から行列式の符号を判定する。実際の行列式の値 D は、

$$D_{approx} - EB \leq D \leq D_{approx} + EB \quad (1)$$

の範囲に存在する。この範囲内に 0 が存在しなければ、実際の値 D と近似値 D_{approx} の符号は一致する (図 1(a))。すなわち、

$$|D_{approx}| \geq EB \quad (2)$$

が成り立つ場合に、符号判定は成功する。

誤差限界値の範囲内に 0 が含まれた場合には、必ずしも実際の値と近似値の符号が一致するとは限らない (図 1(b))。このような場合には、更に下位のデータを取り出すことにより誤差限界値を小さくして、再び適応的符号判定をおこなう。

このように適応的符号判定処理は、一部の上位データのみを用いて効率的に行列式の符号判定を可能にする処理である。

2.2 4×4 行列式の適応的符号判定法

適応的符号判定処理では、ある処理単位を取り出し行列式の符号を判定する。従来の手法では、処理単位を 8 ビット単位として 4×4 行列式の符号判定をおこなっていた。適応的符号判定処理では、処理単位が大きいほど、高速に符号判定がおこなわれる。

ここでは、4×4 行列式の適応的符号判定を 16 ビット単位でおこなう手法について述べる。

4×4 行列

$$M = [m_{ij}] \quad (i, j = 0, 1, 2, 3) \quad (3)$$

を次のように 4 つの行ベクトルを用いて表現する。

$$M = \begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix} \quad (4)$$

ただし、

$$V_i = [m_{i0} \ m_{i1} \ m_{i2} \ m_{i3}] \quad (5)$$

である。行ベクトル V_i の各成分を可変倍長整数型で表現し、16 ビットを処理単位としてこれらの成分を分割する。

ここで、行ベクトル V_i のデータ長 l_i は、 V_i の成分の中で絶対値が最大のデータ長とする。ベクトル V_i の成分 m_{ij} の分割された各成分に下位より 0 から番号を付け、下位から k ($0 \leq k \leq l_i$) 番目の成分を $m_{ij,k}$ と表す。すなわち、ベクトル V_i は、

$$V_{i,k} = [m_{i0,k} \ m_{i1,k} \ m_{i2,k} \ m_{i3,k}] \quad (6)$$

とすると、

$$V_i = \sum_{k=0}^{l_i} 2^{16k} V_{i,k} \quad (7)$$

と表される。

これを用いることにより、行列式 D は、

$$D = \begin{vmatrix} \sum_{k=0}^{l_0} 2^{16k} V_{0,k} \\ \sum_{k=0}^{l_1} 2^{16k} V_{1,k} \\ \sum_{k=0}^{l_2} 2^{16k} V_{2,k} \\ \sum_{k=0}^{l_3} 2^{16k} V_{3,k} \end{vmatrix} = \sum_{r=0}^{l_0} 2^{16r} \sum_{i=0}^{l_1} \sum_{j=0}^{l_2} \sum_{k=0}^{l_3} \begin{vmatrix} V_{0,i} \\ V_{1,j} \\ V_{2,k} \\ V_{3,r-(i+j+k)} \end{vmatrix} \quad (8)$$

と表すことができる。ただし、

$$r - (i + j + k) < 0 \text{ または } r - (i + j + k) > l_3 \quad (9)$$

ならば $V_{3,r-(i+j+k)} = 0$ であり、

$$l_s = l_0 + l_1 + l_2 + l_3 \quad (10)$$

である。

式(8)の下位データを切り捨て、 r の範囲を $0 \leq r \leq l_s$ から $l_s - \alpha \leq r \leq l_s$ にしたものを行列式の近似計算値 $D(\alpha)$ として用いる。

$$D(\alpha) = \sum_{r=l_s-\alpha}^{l_s} 2^{16r} \sum_{i=0}^{l_1} \sum_{j=0}^{l_2} \sum_{k=0}^{l_3} \begin{vmatrix} V_{0,i} \\ V_{1,j} \\ V_{2,k} \\ V_{3,r-(i+j+k)} \end{vmatrix} \quad (11)$$

ここで、 α ($0 \leq \alpha \leq l_s$) は再計算の回数である。 $D(\alpha)$ を $2^{16(l_s-\alpha)}$ で割ったものを $D'(\alpha)$ とすると、以下のように漸化式として表すことができる。

$$D'(0) = \begin{bmatrix} V_{0,l_0} \\ V_{1,l_1} \\ V_{2,l_2} \\ V_{3,l_3} \end{bmatrix}, \quad (12)$$

$$D'(\alpha) = 2^{16} D'(\alpha - 1) + \sum_{i=0}^{l_1} \sum_{j=0}^{l_2} \sum_{k=0}^{l_3} \begin{bmatrix} V_{0,i} \\ V_{1,j} \\ V_{2,k} \\ V_{3,l_2-\alpha-(i+j+k)} \end{bmatrix} \quad (13)$$

行列式の実際の値 D と近似値 $D(\alpha)$ との誤差 $Er(\alpha)$ は、式(8)及び式(11)から次式のようになる。

$$Er(\alpha) = D - D(\alpha)$$

$$= \sum_{r=0}^{l_s-(\alpha+1)} 2^{16r} \sum_{i=0}^{l_1} \sum_{j=0}^{l_2} \sum_{k=0}^{l_3} \begin{bmatrix} V_{0,i} \\ V_{1,j} \\ V_{2,k} \\ V_{3,r-(i+j+k)} \end{bmatrix} \quad (14)$$

式(14)の括弧の中は、 $(l_0+1)(l_1+1)(l_2+1)$ 個の 4×4 行列式が足されたものである。 (i, j, k) の組合せが式(9)を満たす、すなわち、

$$0 \leq r - (i + j + k) \leq l_3 \quad (15)$$

を満たさない行列式の値は0となる。したがって、 $(l_0 + 1)(l_1 + 1)(l_2 + 1)$ 個すべての行列式を計算するのではなく、式(15)を満たす行列式のみを対象とすればよい。行列式の各成分は16ビットの整数であるので、行列式の最大値は $16(2^{16} - 1)^4$ である。よって、 $D(\alpha)$ に対する最小誤差限界値 $EB(\alpha)$ は、

$$EB(\alpha) = \sum_{r=0}^{l_3 - (\alpha+1)} 2^{16r} E_s(r) \quad (16)$$

と表される。ただし、

$$E_s(r) = 16(2^{16} - 1)^4 P(l_3 - r), \quad (17)$$

$$P(t) = \sum_{i=0}^{l_0} \sum_{j=0}^{l_1} \sum_{k=0}^{l_2} N(l_3 - t - (i + j + k)), \quad (18)$$

$$N(x) = \begin{cases} 1 & (0 \leq x \leq l_3 \text{ の時}) \\ 0 & (\text{それ以外の時}) \end{cases} \quad (19)$$

である。 $P(t)$ は、式(15)を満たす (i, j, k) の組合せの個数である。 $EB(\alpha)$ を $2^{16(l_3 - \alpha)}$ で割ることにより、 $D'(\alpha)$ に対する誤差限界値 $EB'(\alpha)$ が求まる。

$$EB'(\alpha) = \sum_{r=0}^{l_3 - (\alpha+1)} 2^{16(r - l_3 + \alpha)} E_s(r) \quad (20)$$

式(20)は複雑であり計算量も多いため、多くの計算時間を必要とする。そこで、

$$EB'(\alpha) \leq 16 \times 2^{48} P(\alpha + 1) \quad (21)$$

が成り立つことから、式(21)の右辺の値を誤差限界値として利用する。すなわち、誤差限界値 $Er'(\alpha)$ は次式のようになる³⁾。

$$Er'(\alpha) = 16 \times 2^{48} P(\alpha + 1) \quad (22)$$

適応的符号判定処理は、

$$|D'(\alpha)| \geq Er'(\alpha) \quad (23)$$

を満たす場合に処理を終了する。このとき、 $D'(\alpha)$ と求める行列式の符号は一致する。式(23)を満たさない場合には、 α を1増やして $D'(\alpha)$ 及び $Er'(\alpha)$ を計算する。式(23)が成り立つあるいは $\alpha = l_3$ となるまで、この計算をおこなう。

2.3 適応的符号判定処理における処理単位

適応的符号判定処理では、ある単位ビットのデータを取り出して、行列式の計算をおこなう。この処理単位ビット長が大きいほど、判定が成功するまでに繰り返される処理回数が減少し、効率的に判定がおこなわれる。処理単位のデータ長をビット単位で設定することは可能であるが、ビット単位で処理するよりもバイト単位で処理したほうがシフト演算などの余分な処理を必要としない分高速に処理がおこなえる。

本研究では、Intel系のFPUで採用されている拡張倍精度浮動小数点型を利用して適応的符号判定処理をおこなう。拡張倍精度浮動小数点型は、1ビットの符号ビット、15ビットの指数部、64ビットの仮数部を持つ。これを利用することにより、64ビットの符号なし整数を正確に表現することができる。この範囲内の整数値であれば、丸め誤差やけた落ちなどの数値誤差を生じることなく、無誤差で演算をおこなうことが可能である。

Table 1 Bit length needed to calculate an $n \times n$ determinant

Processing bit length	2×2 determinant	3×3 determinant	4×4 determinant
8	17	27	36
16	33	51	68
32	65	99	132

$n \times n$ 行列 Δ の各成分の絶対値が m 以下であるとする、Hadamardの不等式から、

$$|\Delta| \leq n^{n/2} \cdot m^n \quad (24)$$

が成り立つ。式(24)より、処理単位ビット長を8, 16, 32ビットとしたとき、2×2, 3×3, 4×4行列式の計算結果を正確に表現するために必要なビット長を表1に示す。拡張倍精度浮動小数点型を利用した場合には、演算途中及び演算結果の数値が64ビット以下の整数である必要がある。表1より、2×2, 3×3行列式の符号判定は、16ビット単位で処理をおこなうことができる。しかし、4×4行列式の適応的符号判定を16ビット単位でおこなう場合には、計算結果が64ビットを超えてしまう。したがって、従来の手法では、4×4行列式の適応的符号判定を16ビット単位でおこなうことはできない。

本論文で提案する手法では、96ビットの整数を正確に表現することができるので、4×4行列式の適応的符号判定を16ビット単位でおこなうことが可能である。処理単位を32ビット単位とすると、4×4行列式を正確に表現するためには132ビット必要となり、96ビットを超えてしまう。そこで、4×4行列式の適応的符号判定処理は、処理単位を16ビット単位としておこなう。

3. 適応的符号判定処理の高速化

4×4行列式の適応的符号判定を16ビット単位でおこなう場合には、68ビットのデータ長が必要である。しかし、拡張倍精度浮動小数点型の仮数部は64ビットであるため、行列式の計算結果を正確に表現することができない。

ここでは、FPUの複数のレジスタを利用することにより4×4行列式の計算結果を正確に表現する手法を示す。これにより、処理単位を16ビットとして、4×4行列式の適応的符号判定処理をおこなうことができる。処理単位が大きいほど高速に符号判定をおこなうことができるため、16ビット単位での処理は、従来の8ビット単位での処理に比べ効率的に処理をおこなうことができる。

3.1 64ビットを超えるデータ長の表現

データ長が64ビット以内の数値は、FPUの拡張倍精度浮動小数点型で正確に表現することができる。しかし、64ビットを超えたデータ長の数値は、数値誤差のため正確に表現できない。そこで、演算結果が64ビットを超えるような場合には、FPUレジスタ2本を用いて数値を表現する。この方法により、最大96ビットのデータ長の数値を正確に表現することができる。

64ビットを超えるデータ長である整数を a とする。整数 a は、2つの64ビット整数 a_0, a_1 を用いて、 a_1 を32ビット左にシフトしたものに a_0 を加算した形で、96ビット整数として表現される(図2参照)。

$$a = a_0 + a_1 \times 2^{32} \quad (25)$$

96ビット整数 a に64ビット整数 b を乗じる処理は、

$$a \times b = a_0 \times b + a_1 \times b \times 2^{32} \quad (26)$$

であるので、64ビット整数 a_0, a_1 に b を乗じる処理としておこなわれる。このとき、乗算によりけたあふれが起こる可能性もあり、そのよう

Fig. 2 Representation of a 96 bit length integer

な場合には数値誤差が生じてしまい、後に述べる対策が必要となる。また、2つの96ビット整数 a , c の加算は、

$$a + c = a_0 + c_0 + (a_1 + c_1) \times 2^{32} \quad (27)$$

として、64ビット整数どうしの加算として処理される。この場合にも、けたあふれが起こる可能性がある。

そこで、4×4行列式を計算するときに、けたあふれが発生することがないように乗算及び加算をおこなう必要がある。

3.2 4×4行列式の計算

16ビットのデータ長の成分を持つ4×4行列式の計算方法を以下に述べる。

可変倍長整数型から取り出した16ビットデータは符号なし整数型であり、符号は別に保持されている。そこで、このデータを符号付き整数型に拡張する。符号なし16ビット整数を符号付きとして表現するためには、17ビット必要である。したがって、符号なし16ビット整数型を32ビット符号付き整数型に展開する。

4×4行列式を効率的に計算するために、4×4行列式を次のように展開する。

$$\begin{aligned} |M| = & m_{00}(m_{11}d_0 - m_{12}d_1 + m_{13}d_2) \\ & + m_{01}(-m_{10}d_0 + m_{12}d_3 - m_{13}d_4) \\ & + m_{02}(m_{10}d_1 - m_{11}d_3 + m_{13}d_5) \\ & + m_{03}(-m_{10}d_2 + m_{11}d_4 - m_{12}d_5) \end{aligned} \quad (28)$$

ただし、

$$\begin{aligned} d_0 &= \begin{vmatrix} m_{22} & m_{23} \\ m_{32} & m_{33} \end{vmatrix}, d_1 = \begin{vmatrix} m_{21} & m_{23} \\ m_{31} & m_{33} \end{vmatrix}, \\ d_2 &= \begin{vmatrix} m_{21} & m_{22} \\ m_{31} & m_{32} \end{vmatrix}, d_3 = \begin{vmatrix} m_{20} & m_{23} \\ m_{30} & m_{33} \end{vmatrix}, \\ d_4 &= \begin{vmatrix} m_{20} & m_{22} \\ m_{30} & m_{32} \end{vmatrix}, d_5 = \begin{vmatrix} m_{20} & m_{21} \\ m_{30} & m_{31} \end{vmatrix} \end{aligned} \quad (29)$$

であり、これらはブリュッカー座標¹⁾と呼ばれる。

FPUを利用して4×4行列式を計算するアルゴリズムを以下に示す。

- ① ブリュッカー座標 $d_0, \dots, d_5$ を計算する。この計算結果を正確に保持するためには、最大で33ビット必要である。この段階では、FPUレジスタの仮数部64ビット内に収めることが可能である。
- ② 式(28)の括弧の内部を計算する。ここでの計算結果もFPUレジスタの64ビットの仮数部で十分表現可能である。

③ ②の演算結果に $m_{00}, \dots, m_{03}$ を乗じる。この乗算による計算結果は、64ビットを超えてしまう可能性がある。そこで、②の演算結果を式(25)の形式で表現し、64ビット整数 a_0 , a_1 に $m_{00}, \dots, m_{03}$ を乗じる。このとき、式(25)の a_0 , a_1 は上位32ビットが空の状態であるので、これらに有効ビット数17ビットの $m_{00}, \dots, m_{03}$ を乗じてもけたあふれは生じない。

④ ③までで求めた4つの項の加算をおこなう。各々の項は $a = a_0 + a_1 \times 2^{32}$ の形式で表現されており、 a_0 , a_1 はそれぞれ少なくとも上位15ビットは空いている。したがって、各項の加算により、けたあふれが生じることはない。

⑤ ④の演算結果の a_0 の上位32ビットを空にする。この処理は a_0 の上位32ビットを a_1 に加算する処理となる。

なお、ここで述べた計算は、適応的符号判定処理の1回目の計算方法である。2回目以降の計算では、④で各項は前回の計算結果を左に16ビットシフトしたものに加算することになる。

3.3 誤差限界値との比較

4×4行列式の計算結果は、96ビット整数として式(25)の形式で表現されているので、64ビット整数で表現されている誤差限界値と直接比較することができない。そこで、4×4行列式の計算結果及び誤差限界値を比較することができる形に変換する。

式(22)より、誤差限界値は 2^{48} のけた以上の値のみが重要である。したがって、2つの値の 2^{48} のけた以上の部分について比較をおこなえばよい。この部分は64ビット整数型で十分表現可能なので、単純に数値を比較することができる。

3.4 限界再計算回数

FPUを利用した適応的符号判定処理では、行列式の近似値 $D'(\alpha)$ は96ビット整数として表現される。この処理では符号判定不能な場合に再計算がおこなわれるが、これが何度もおこなわれると $D'(\alpha)$ のデータ長が96ビットを超えてしまうことがある。このような場合には、数値誤差により正確な符号判定をおこなうことができない。そこで、 $D'(\alpha)$ が96ビットを超えない限界の再計算回数を求める必要がある。

$D'(\alpha)$ が96ビット以内であるためには、式(13)の第1項及び第2項が95ビット以内でなければならない。

第1項の $D'(\alpha-1)$ は、再計算において1回前に計算された行列式の値である。 $D'(\alpha-1)$ がこの段階における誤差限界値 $Er'(\alpha-1)$ より大きいならば、その時点で符号判定処理は終了している。よって、 $D'(\alpha-1)$ の最大絶対値は $Er'(\alpha-1)$ であるので、

$$|D'(\alpha-1)| \leq Er'(\alpha-1) = 16 \times 2^{48} P(\alpha) \quad (30)$$

が成り立つ。したがって、第1項が95ビット以内に収まるには、

$$2^{16} (16 \times 2^{48} P(\alpha)) \leq 2^{95} \quad (31)$$

を満足すればよい。ここで、

$$P(\alpha) \leq P'(\alpha) = \frac{(\alpha+1)(\alpha+2)(\alpha+3)}{6} \quad (32)$$

が成り立つ⁶⁾ので、 $P(\alpha)$ の代わりに $P'(\alpha)$ を用いて式(31)からこの不等式を満たす最大の整数を算出すると、 $\alpha = 928$ を得る。

第2項は $P(\alpha)$ 個の4×4行列式を足したものである。したがって、第2項が95ビット以内に収まるには、

$$16(2^{16} - 1)^4 P(\alpha) \leq 2^{95} \quad (33)$$

を満足すればよい。 $P'(\alpha)$ を利用することにより、式(33)を満たす最大の整数 $\alpha = 928$ が得られる。

Table 2 The computation time of a 4×4 determinant

Data length byte	(a) ms	(b) ms	(c) ms	(d) ms
1	0.3285	0.4313	0.0096	0.0071
2	0.3238	0.4127	0.0098	0.0070
4	0.3484	0.4143	0.0096	0.0071
8	0.4378	0.4163	0.0096	0.0069
16	0.7319	0.4117	0.0099	0.0070
32	1.7476	0.4171	0.0106	0.0071
64	5.4443	0.4284	0.0107	0.0071
128	19.6178	0.4136	0.0111	0.0070
256	75.5340	0.4145	0.0103	0.0070
512	296.7711	0.4161	0.0107	0.0070
1024	1261.6750	0.4496	0.0108	0.0074

したがって、再計算回数が $\alpha = 928$ 以下であるならば、 $D'(\alpha)$ が 96 ビットを超えることはない。

次に、誤差限界値について検討する。誤差限界値は、 2^{48} のけた以上の部分、すなわち、 $16P(\alpha+1)$ が 64 ビットで表現されている。したがって、誤差限界値が無誤差で表現されるためには、

$$16P(\alpha+1) \leq 2^{64} \quad (34)$$

を満たせばよい。この不等式を満たす最大の整数を算出すると、 $\alpha = 1905386$ が求まる。

以上より、正確に符号判定をおこなうことのできる再計算回数の限界値は、 $\alpha_{\max} = 928$ である。

4. 実験

ここでは、本論文で述べたアルゴリズムの性能を実験によって評価する。可変倍長整数型による全倍長計算、FPU を利用しない適応的符号判定処理、8 ビット単位で FPU を利用した適応的符号判定処理、16 ビット単位で FPU を利用した適応的符号判定処理のそれぞれについて、4×4 行列式の符号判定に必要な時間を計測し、その結果の比較をおこなう。

4.1 実験環境

使用コンピュータ: Dell Dimension XPS Txxx
(Pentium III 500MHz, メモリ 384MB)

OS: Microsoft Windows NT 4.00

使用ソフトウェア: Microsoft Visual C++ 6.0

4.2 実験方法

4×4 行列式の符号判定時間の計測を以下の 4 種類の方法に關しておこなう。

- (a) 可変倍長整数型による全倍長計算
- (b) 適応的符号判定処理 (16 ビット単位)
- (c) FPU を利用した適応的符号判定処理 (8 ビット単位)
- (d) FPU を利用した適応的符号判定処理 (16 ビット単位)

行列の各成分のデータ長が 1, 2, 4, ..., 256, 512, 1024 バイトであるような 4×4 行列式をランダムに 100 個生成し、符号判定にかかる時間を計測する。その平均値を行列式 1 個の符号判定時間として採用する。また、(d) に関して、符号判定のために取り出すデータ長を求める。

なお本実験では、可変倍長整数型には GNU の C++ クラスライブラリの Integer 型を利用した。

4.3 実験結果

実験結果を表 2 及び図 3 に示す。

Fig. 3 The computation time of a 4×4 determinant

(a)の全倍長計算では、データ長の増加に伴い処理時間も指数関数的に増加している。これに対し、適応的符号判定処理では、データ長によらずほぼ一定の計算時間で符号判定をおこなうことができることが分かる。

FPUを利用することにより、更に高速な処理が可能であることが分かる。(b)に比べ、(c)では約 40 倍、(d)では約 60 倍処理が高速化されている。16 ビット単位での処理は、8 ビット単位での処理よりも高速であることが確認できる。(d)の場合、1 ワードの取り出しでの符号判定成功率は 99%以上であり、2 ワードの取り出しですべての符号判定に成功した。この結果からも、データ長の大小にかかわらず一定の時間で符号判定をおこなえることが分かる。

5. 結 言

本論文では、4×4 行列式の適応的符号判定処理を高速化する手法を提案した。この手法では、2 つの FPU レジスタを用いることにより、処理単位を 16 ビットとして処理をおこなう。これにより、処理単位を 8 ビットとしていた従来の手法よりも、高速に符号判定をおこなうことが可能となる。

性能評価試験をおこない、本論文で述べた手法が、従来の手法に比べ高速であることを確認した。

参 考 文 献

- 山口富士夫: 4 次元理論による図形・形状処理工学, 日刊工業新聞社, (1996).
- 吉田典正, 塩川誠人, 山口富士夫: 4×4 行列式法における適応的符号判定処理 (第 1 報) - n 次の内積計算における適応的符号判定処理およびその特性 -, 精密工学会誌, 60, 9 (1994) 1247.
- 吉田典正, 榎 林, 坂根賢一, 雑賀定樹, 山口富士夫: 4×4 行列式法における適応的符号判定処理 (第 2 報) - 同次多項式の符号判定処理 -, 精密工学会誌, 62, 9 (1996) 1267.
- 花蜜宏晃, 吉田典正, 榎 林, 鈴木重行, 山口富士夫: 浮動小数点ユニットを利用した適応的符号判定処理 - 3×3 行列式の符号判定 -, 精密工学会誌, 63, 5 (1997) 657.
- 土井 淳, 津金尚志, 山内俊哉, 金 俊赫, 山口富士夫: 4×4 行列式を対象とした適応的符号判定処理の高速化, 1998 年度精密工学会春季大会学術講演会講演論文集, (1998) 4.
- 花蜜宏晃: FPU を利用した高速かつ無誤差な符号判定処理, 早稲田大学修士論文, (1997).