16ビットパーソナルコンピュータの演算処理機能

藤　村　公　男

1　はじめに

近年におけるパーソナルコンピュータ（以下パソコンと略す）の発展は質量ともに著しい。いうまでもなく，これは高性能な高密度集積回路の多量生産が可能となったためである。低価格のパソコンが市場に提供され，需要を刺激し，これがまた高性能製品の開発を促すといった需要供給の好循環によって，極めて単期間にパソコン市場が拡大された。現在でもパソコン市場は年率20%の急成長が続いている。パソコン本体だけでなく，ディスクやディスプレイ，プリンタなどの周辺装置も高機能，低価格へと向かっている。利用者にとってこれほど有難いことはなく，大学の研究室にも，場合によっては自宅にもパソコンを導入し，在宅研究が可能となった。

ハードの技術革新はソフトの多様化をもたらした。パソコンはワードプロセッサや表計算，データベース，データ通信など，主として事務用ソフトの開発によって広く普及したといえる。大ヒットしたNECのPC98シリーズ

では6,000本以上の応用ソフトが利用可能といわれている。

パソコンの利用は事務用ばかりではない。Per Bak［1］は専門的な科学技術計算の分野で，大型汎用コンピュータと比較しながら，パソコンの利用は有効で，もっと多用すべきであると主張している。彼は“A large computer is like a dinosaur : a large body with a small brain”（「大型コンピュータは，巨大な体躯に小さな頭脳しか持たない恐竜のようなものである」）と評している。大型コンピュータシステムを維持してゆくには，コンピュータ本体の価格より専用の建物や空調設備，ワークステーションなどの周辺装置，システムを管理する人件費等のオーバーヘッドが巨額となり，コンピュータの使用コストを著しく高めている。彼はNSF（National Science Foundation : 米国科学研究財団）の検閲者であるという立場にある。NSFの研究費の中で占める大型コンピュータ使用料金は無視し得ない程度となり，これが米国政府の財政を圧迫する一因となっている。またNSFの研究費を得て行われている研究の大部分は，オーバーヘッドの少ないパソコンで代用できるとしている。これに対して多くの科学者が反論している［2］。Per Bakの論評は少し偏っている面があるが，科学研究の分野でパソコンの果す役割は増大しつつあることは確かであろう。

Per Bakの論文が動機となって筆者は現在最も普及していると考えられる16ビットパソコン（FM16β : 80186MPU, 8MHz, RAMとして1MB実装）を用いて，その演算処理機能（演算精度，計算速度，メモリ利用率）について，系統的に検討することにした。科学計算はFORTRANなどのプログラミング言語によって行われる。パソコンの普及と共にBASIC, PASCAL, Cなどの高級言語も開発された。本稿では，これらの言語の演算機能を比較検討する。また，MPUと連結して，MPUだけでは出来ない浮動小数点演算をハードウェア的に高速処理する数値演算コプロセッサが開発されている。この効果についても検討したい。

2　高級言語と数の内部表現

使用する高級言語は富士通のF-BASIC86インタプリタV3.1［3］，同コンパイラV3.10［4］，Prospero Software社のPro FORTRAN Version iid 2.1［5］と同社のPro PASCAL Version iid 2.2［6］，Lattice社のLattice C V3.10［7］の5種である。オペレーティングシステム（OS）はマイクロソフト社のMS-DOS V3.1［8］である。F-BASIC86インタプリタとコンパイラはソースプログラムの処理方式が異なるだけで文法は同じである。インタプリタはプログラムの一命令ごとに機械語に翻訳，実行する。コンパイラはソースプログラムの全体を機械語に翻訳して実行可能形式のプログラムを作成し，実行する。後者はコンパイルという面倒な操作を必要とするが，インタプリタと比べて，プログラムの実行が高速となる。FORTRAN, PASCAL, Cはコンパイラ言語であり，ソースプログラムの編集，コンパイル，リンクという複雑な手続きを経て，実行可能形式のプログラムが作成される。しかし，プログラムが構造化されていること，アセンブラを含めた他のプログラムとの結合が容易なこと，プログラムの実行速度が高いことなど，インタプリタにない優れた特徴を持っている。

コンピュータの内部ではすべての情報は0と1の2進法で表現され，処理される。従って数値計算で扱う数をいかに2進法で表現するかが問題となる。16ビットMPU（マイクロプロセッサユニット）は基本的に16ビット（2バイト）単位[1)]でデータ処理を行う。16ビットMPUである8086[2)]では，16ビットの汎用レジスタが使われ，RAMメモリも16ビット単位でアクセスされる。これらのレジスタ－レジスタ間またはレジスタ－メモリ間で加減乗除の四則演算が行われる。従って数値表現も16ビットか16ビットの整数倍の

注1）情報量の単位としてビットの他にバイト（＝8ビット）を用いることもある。
2）本稿で使用したFM16βは80186だが，これは8086と周辺チップを一つにまとめてオンチップ化したもので，機能は8086と同じである。

ものが最も効率が良い。[3)]

一方，高級言語で処理する数は整数型と実数型に区別される。FORTRANでは更に複素数型があるが，これは複素数の実部と虚部をそれぞれ実数型とする構造体データとみなされる。従って複素数型も基本的には実数型に帰着される。コンピュータによる数値計算で問題となるのは計算精度または有効桁数[4)]と計算速度である。精度と速度は少しでも高い方が良い。しかし，精度を高めるため有効桁数を多くすると計算に多くの時間を要する。従って，計算の精度と速度は互いに相反する。整数演算はMPU自身が行うので，一般に高速でクロック周波数にほぼ比例して速度は高くなる。しかし，実数型の場合は表現方式も演算方式も多様で言語によって異なる。MPU自身は実数型を直接処理することは出来ないので，通常，高級言語は実数型演算処理を行うプログラムを内蔵している。一般に実数演算時間は整数型より数十倍になる。MPUでは出来ない実数演算をハードウェア的に高速処理する専用の集積回路が開発されている。これを数値演算コプロセッサ（Numeric Data Processor：NDP）という。NDPはソフトウェアでは手間のかかる実数型演算をMPUと連結してハードウェア的に高速処理する。パソコン本体のMPU基板に既にソケットが配線済みとなっており，NDPをそのソケットに装着するだけで使用できる。NDPは特に数値関数の計算速度の向上に効果がある。

以下，コンピュータ言語で扱う数の内部表現の代表的な例を整数型と実数型に分けて要約する。詳細については文献［9］を参照されたい。

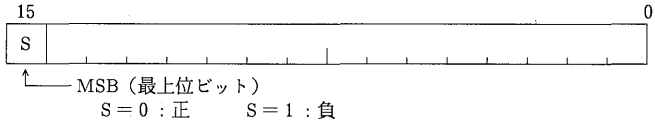
（A） 整数型の内部表現

16ビットMPUでは機械語の一命令で16ビットの四則演算が実行されることから，高級言語でも16ビットを標準的な整数単位としている。2進整数表

3）8ビットコンピュータとの互換性を考慮して8ビット演算も可能である。
4）整数型の場合，ここでいう有効桁数というのは，整数として扱える数の大きさの範囲である。整数の最大，最小が決められると，その範囲内で計算誤差はない。

示では0または正のみの符号なし整数と，正負の符号つき整数とを区別しなければならない。nビットの符号なし整数の範囲は0から2^n-1までである。16ビット長では0から$2^{16}-1=65535$までとなる。一方，符号つき整数はやや複雑となるが主として次の二つの表現方式がある。一つの方式は最上位ビット（MSB）を符号ビットとして，MSBが0のときは正，1のときは負とし，残りのビットを通常の符号なし整数とする。この方式では正数は$+0$～$2^{n-1}-1$，負数は-0～$-(2^{n-1}-1)$となる。0は$+0$と-0の2つある。この方式はわれわれが通常10進法で正負を表現する方式であり，単純で理解し易いが，演算が複雑になるという欠点を持っている。Pro FORTRANとPro PASCALの4バイトinteger型はこの表示法を採用している。符号つき整数を表現するもう一つの方式は2の補数表示である。これはビット長が有限で，整数の値が決められた範囲内で有効であるというコンピュータハードの特徴を利用したものである。nビットの2の補数表示では0および正数は0～$2^{n-1}-1$，負数は-1～-2^{n-1}の範囲となる。加減算は簡単となり，効率の良い演算が可能である。Lattice Cはこの方式を採用している。いずれの方式もMSBを符号ビットとする（図1）。符号なしに

図1　16ビット符号つき整数の2進表示



しろ，符号つきにしても，16ビットでは整数値としての値の範囲はせますぎるので，F-BASIC86を除く他の言語では16ビットの倍長となる32ビット（4バイト）整数を扱うことが可能である。

整数型の特徴は値が決められた範囲内で誤差は皆無で，演算速度も高いということである。従ってプログラムでは可能な限り整数型を使うべきである。しかし，絶対値の範囲が実数型に比べてせまいので，四則演算（特に乗除算）の結果，オーバーフロー（桁あふれ）になる危険性が大きい。

プログラム中で整数型を指定するためにはプログラムの最初で型宣言しな

ければならない。以下，変数 x が整変数であることを宣言する命令（型宣言文）を各言語ごとに示す。[5)]

（1）F-BASIC86

DEFINT x …最初の文字が x で始まるすべての変数は 2 バイト符号つき整数

ただし，変数名の末尾に型宣言文字%をつけると上記の宣言文がなくとも良い。型宣言文も型宣言文字もない変数はすべて自動的に後述の単精度実数型とみなされるので注意が必要である。

（2）Pro FORTRAN

INTEGER x　……x は 4 バイト符号つき整数

INTEGER＊1 x …x は 1 バイト符号つき整数

INTEGER＊2 x …x は 2 バイト符号つき整数

INTEGER＊4 x …x は 4 バイト符号つき整数

上記の INTEGER と INTEGER＊4 は同じである。また FORTRAN の伝統に従って，宣言文がない場合は，最初の文字が I, J, K, L, M, N である変数はすべて INTEGER 型とみなされる。

（3）Pro PASCAL

INTEGER x …x は 4 バイト符号つき整数

これ以外の整数型を扱うときは TYPE 宣言によって行う。Pro PASCAL の型宣言の仕方は独特であり，型自身をプログラマーが定義できる。例えば

TYPE int = −32767 : 32767 ;

VAR int x ;

とすると x は 2 バイトの符号つき整数となる。

（4）Lattice C

char x …x は 1 バイトの符号つき整数[6)]

5）通常，ソースプログラム中の命令文や変数名では，ローマ字の大文字，小文字の区別をしない。本稿でもその習慣に従う。

6）1 バイト整数は数値としてよりも，文字列の ASCⅡコードとして使われることが多い。数値としての演算は可能である。

int x ……xは2バイトの符号つき整数

long int…xは4バイトの符号つき整数

負数は2の補数表示に従う。また上記のそれぞれにunsignedをつけると符号なし整数となる。

（B）　実数型の内部表現

プログラミング言語における実数型とは，整数型では表わせない有理数や無理数を近似的に2進または16進で表現する型であり，科学計算では不可欠である。1／2（0.5）や1／4（0.25）のように小数点つき2進表示の有限桁で表現できる数の他に1／3や$\sqrt{2}$，円周率のように2進でも10進でも無限桁まで続くものがある。注意すべきことは10進法で有限桁の数も2進法で無限桁となる場合もあるということである。例えば10進の0.6は2進表示で0.100110011…と無限に続く。もちろん，現在のコンピュータでは無限桁の数を表現，処理することは不可能である。従ってどんなプログラミング言語でも，実数型をいかに2進または16進の有限桁で近似表現するかが問題となる。上記の0.6を2進8桁0.10011001で近似すると，これは10進で0.59765625に変換され，真値0.6に比べると約0.4%の相対誤差となる。一般に有効桁の多い数値を有効桁の少ない数値に変換することで生じる誤差を「丸め誤差」といわれている。われわれが日常行っている四捨五入もこの丸め誤差を承知の上で，有効桁数を少なくして，表現を簡単にしていることに他ならない。計算の精度を高めるためにはコンピュータの内部表現のビット長を長くする必要がある。しかしビット長を増すほど，10進から2進への変換およびその逆変換，また演算に要する時間が増加する。整数型の場合と同様に，実数型でも精度（有効桁数）と計算速度は相反する。丸め誤差は10進から2進に変換するときに生じる。その誤差をなくするために，コンピュータの内部表現を10進表示とし，演算も10進で行うソフトもある。しかし，これはメモリの利用効率が悪く，演算速度も遅くなるのであまり一般的でない。

実数型では，精度（有効桁数）の他に，数の絶対値がどの程度まで可能か

が問題となる。絶対値の大きさに幅がある数を表現するには，以下のように指数表示を使う。

$$\pm M \times R^{E} \qquad (2-1)$$

±は数の正負符号，Mは仮数部，Eは指数部，Rは数表現の基数である。例えば10進の123億は0.123×10^{11}と表わす。0.001は0.1×10^{-2}と表現する。（2－1）のような数表現を浮動小数点表示という。2進の浮動小数点表示では基数はR＝2となるので

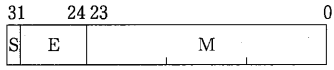
$$\pm M \times 2^{E} \qquad (2-2)$$

となる。±，M，Eは2進表示とする。コンピュータ言語における実数型はすべてこの浮動小数点表示に従う。問題は全体としてのビット長と，M，Eにどれだけのビット長を割り当てるかである。

以下，実際のコンピュータで採用されている代表的な浮動小数点形式を示す。ここで掲げた三形式とも，32ビット（4バイト）長の単精度型と64ビット（8バイト）長の倍精度型がある［9］。

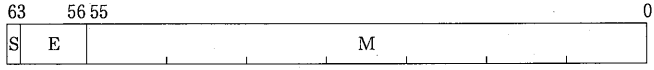
（i） IBM社の汎用コンピュータIBM360，370シリーズおよびこれらの互換機で採用されている実数型で，16進表示である。

○単精度（4バイト）



有効桁は10進で約7桁

○倍精度（8バイト）



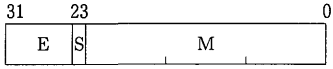
有効桁は10進で約16桁

単精度，倍精度とも，指数部は7ビット長で値の範囲は $V \fallingdotseq \pm5.4\times10^{-79}$ ～ $\pm7.2\times10^{75}$ である。

（ii） マイクロソフト社の2進浮動小数点形式

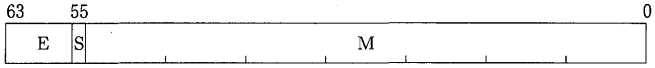
これはマイクロソフト系BASICで採用されている。米国ではIBM PC，アップル，日本ではNEC，富士通，日立などのパソコン用BASICがこれに従っている。

○単精度（4バイト）



有効桁数は10進で約7桁

○倍精度（8バイト）



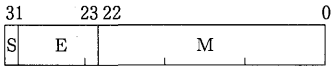
有効桁数は10進で約16桁

両精度とも，指数部は8ビットで，値の範囲は $V \fallingdotseq \pm 2.9\times10^{-39} \sim \pm 1.7\times10^{38}$

（iii） IEEE規格の2進浮動小数点形式

これは米国電気電子学会（Institute of Electrical and Electronics Engineers）がパソコンの分野で統一的に使用するよう定めた規格である。Pro FORTRAN, Pro PASCAL, Lattice Cはこれを採用している。また数値演算コプロセッサ8087はこの形式を採用している。

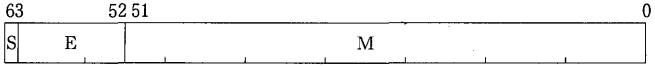
○単精度（4バイト）



指数部は8ビット，仮数部は23ビット。

有効桁数は10進で約7桁。値の範囲は $V \fallingdotseq \pm 1.1\times10^{-38} \sim \pm 3.4\times10^{38}$

○倍精度（8バイト）



指数部は11ビット，仮数部は52ビット。

有効桁数は10進で約16桁。値の範囲は $V \fallingdotseq \pm 2.2\times10^{-308} \sim \pm 1.8\times10^{308}$

IEEE 規格の（ｉ），（ii）と異なる特徴は，倍精度が単精度より指数部のビット長が大きくなることである。その結果，倍精度の絶対値の幅が大きくなる。

上記の三形式とも，有効桁はあまり差がないが，10進表示での指数部の差が大きいことに注意すべきである。以上の浮動小数点形式は実数型の代表的な例であり，これ以外の形式もある。Borland 社の Turbo PASCAL V1.00A では，実数型は単精度，倍精度の区別はなく，６バイト長のみである。一般にプログラミング言語はなるべくハードウェアに依存しない「移植性」の高い仕様が期待される。従って数の表現，特に浮動小数点形式のようなハードウェア依存性の強いものは，言語の標準仕様で厳密に規定されていない場合が多い。例えば Pro PASCAL の倍精度実数型は標準 PASCAL の拡張仕様となっているに過ぎない［６］。また標準Ｃ言語とされているカーニハン，リッチのテキスト［10］でも単精度（float）と倍精度（double）の区別はあるが，それらの具体的なビット長や表示形式は決められていない。どのような表示形式を採用し，どのように演算処理するかは，コンパイラ言語の処理系（コンパイラ）の作成者に任されている。Lattice C では，入出力およびシステム関数などの基本ライブラリと浮動小数点演算ライブラリが別個に存在し，後者は浮動小数点演算が必要なときオプションとしてリンクすれば良い。従ってこの浮動小数点演算ライブラリをアセンブラで作り直すことによって，利用者の目的に最適なライブラリとすることが可能である。例えば倍精度より更にビット長を大きくして超倍精度演算が可能なライブラリをアセンブラで作り，それを Lattice C にリンクして利用することなどである。最近発表された Borland 社の Turbo C2.0では80ビット（10バイト）長の実数型で $3.4\times10^{-4932} \sim 1.1\times10^{4932}$ が可能である。

以下，変数 x が単精度型，変数 y が倍精度型であることを指定する宣言文

を各言語ごとに示す。

（1）F-BASIC86（マイクロソフト系BASIC規格）

DEFSNG x …… x で始まる変数は単精度型

DEFDBL y …… y で始まる変数は倍精度型

宣言文がなくても，x!，y#のように型宣言文字!（単精度），#（倍精度）を変数の末尾につけても良い。型宣言文も型宣言文字もない変数はすべて単精度とみなされる。

（2）Pro FORTRAN（IEEE規格）

REAL＊4 x（またはREAL x）

REAL＊8 y（またはDOUBLE PRECISION y）

（3）Pro PASCAL（IEEE規格）

real x ;

longreal y ;

（4）Lattice C（IEEE規格）

float x ;

double y ;

3　浮動小数点演算の精度

実数型の場合，単精度にするか倍精度にするかはプログラムの目的に応じて決定しなければならない。高精度で指数部の範囲が大きい場合はIEEE規格の倍精度を使うべきである。ただし計算時間とメモリ使用量は単精度に比べて多くなる。

単精度か倍精度かを決定するもう一つの要因は演算の結果生じる「桁落

ち誤差」である。この簡単な例は2次方程式 $ax^2+bx+c=0$ の根を計算する場合である。2実根は $x_0=(-b+\sqrt{b^2-4ac})/(2a)$，$x_1=(-b-\sqrt{b^2-4ac})/(2a)$ となるが，$b>0$ かつ $b^2\gg|4ac|$ の場合に x_0 の計算に桁落ち誤差が生じる。このとき b と $\sqrt{b^2-4ac}$ とはほとんど同じ値となり x_0 の分子である $-b+\sqrt{b^2-4ac}$ の有効桁は失なわれる。この解決法は $x_0=-2c/(b+\sqrt{b^2-4ac})$ のように変形すれば桁落ち誤差は防げる。この例のように簡単な場合は良いが桁落ち誤差はいろいろな場合に生じ，時によっては桁落ち誤差が何回も積み重なって最終結果がほとんど無意味な値になることがある。

桁落ち誤差が最も深刻でその解決が困難な問題は逆行列の計算である。行列演算は単に自然系分野に限らず，経済経営の分野でも重要な役割を演ずる。W. レオンチェフ等によって展開された有名な経済学の投入・産出原理［11］やオペレーションズリサーチにおける線型計画法［12］は逆行列演算と深く関係している。

N次連立線型方程式は行列形式で

$$Ax=c \qquad (3-1)$$

と書ける。x はN次の未知数ベクトル，cはN次の定数ベクトル，AはN行N列の係数行列である。一般にこれらの要素または成分は実数型として扱われる。問題はAのN^2個の要素とcのN個の成分がデータとして与えられたとき（3－1）を満足するxのN個の成分を求めることである。$N=2$次のときは

$$A=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix},\quad x=\begin{bmatrix} x_1 \\ x_2 \end{bmatrix},\quad c=\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \qquad (3-2)$$

とすると（3－1）は次の2元連立方程式となる。

$$\begin{cases} a_{11}x_1+a_{12}x_2=c_1 \\ a_{21}x_1+a_{22}x_2=c_2 \end{cases} \qquad (3-3)$$

この解は

$$\begin{cases} x_1 = (c_1\ a_{22} - c_2\ a_{12}) / \det A \\ x_2 = (-c_1\ a_{21} + c_2\ a_{11}) / \det A \end{cases} \qquad (3-4)$$

$$\det A = a_{11}\ a_{22} - a_{12}\ a_{21} \qquad (3-5)$$

$\det A$ はAの行列式である。（3－4）の解で $\det A$ が分母にくるので $\det A$ が0になると解は無意味となる。$\det A$ が0にならないまでも桁落ち誤差の生じる可能性は充分にある。この問題は一般的なN次の場合でも同様である。一般式である（3－1）の解はAの逆行列をA^{-1}とすると形式的に

$$x = A^{-1}C \qquad (3-6)$$

となる。A^{-1}を求めるときに $\det A$ が分母に現われ，$\det A$ が0に近くなるとオーバーフローかまたは桁落ち誤差が発生する危険性がある。この困難を避けるためには計算時間の多くなることを無視して，なるべく高精度な計算をすることである。

実際のコンピュータを使って，逆行列の計算を行い，どの程度の誤差が生じるかをテストする一つの方法は以下に示すヒルベルト行列の逆行列を計算することである［13］。

N次のヒルベルト行列Aの各要素 a_{ij} は

$$a_{ij} = 1 / (i + j - 1) \qquad (3-7)$$

と定義される。ただし，$i, j = 1, 2, 3, \cdots, N$。この行列の各要素は正で，$N$が大きくなるほど，近接する要素の差が少なくなるので逆行列の計算の際，桁落ち誤差が発生する。本稿では単精度型と倍精度型の両方で計算を行い，両者の差を比較する。A^{-1}の一般式は代数的に決められないのでA^{-1}の計算結果と真値を直接比較することは出来ない。そこで$A^{-1}A = I$（IはN次単位行列）となることを利用して，計算されたA^{-1}から$A^{-1}A$を計算し，更にIを減じ，その結果が0からどの程度離れるかを調べた。すなわち

$$A^{-1}A - I = 0 \qquad (3-8)$$

（0はゼロ行列）

表1　$A^{-1}A-I$の要素の中で絶対値が最大のもの（真値は0）
（AはN次ヒルベルト行列。F-BASIC86による）

N	2	3	5	7	10
単精度	0	7.6×10^{-6}	1.4×10^{-2}	6	192
倍精度	0	2.2×10^{-16}	2.3×10^{-12}	2.2×10^{-9}	2.1×10^{-4}

の左辺を計算機で計算し，その各要素が真値0からどの程度ずれるかをみる。逆行列A^{-1}を標準的で最も一般的に用いられているGauss-Jordan消去法[14]によって求める。N次正方行列の要素はN^2個あるので高次ではこれらのすべての要素を出力することは困難である。従って，$A^{-1}A-I$のN^2個の要素の中で絶対値が最大となるものを抜き出して出力した。F-BASIC86による結果を表1に示す。この結果は他の言語でもほとんど同じである。表1からわかるとおり単精度では$N=3$次元から，倍精度では5次元から既に桁落ち誤差が生じている。

ヒルベルト行列はコンピュータによる桁落ち誤差をチェックするにはきびしすぎるかも知れない。通常の線型計算では，このようなことはあまり起らないであろう。しかし，桁落ち誤差が生じているか否か一般的に判断することは極めて困難である。N次正方行列の逆行列を求めるには$2N^3$回程度の実数演算が必要である。従ってNが大きくなると計算時間が無視し得なくなる。演算ごとに桁落ちが生じているか否かを判断するプログラムも可能だが，そのためには更に多くの余分の時間が必要となる。コンピュータでこのような線型計算を行うときは少なくとも単精度と倍精度の両方で行い，両者の結果を比較して，桁落ち誤差の少ないことを確認する必要がある。倍精度は単精度より2倍のメモリ領域を必要とする。N次正方行列はN^2個の要素があるので，高次になると必要なメモリサイズは急速に増大する。

4　四則演算と数値関数の計算速度

ここではクロック周波数が8 MHzの80186をベースとしたFM16βを使っ

て各言語ごとに演算処理時間がどの程度異なるものかを比較検討する。浮動小数点演算については数値演算コプロセッサ80187の効果も検討する。

演算時間を決定するためには通常，演算を繰り返し実行し，その間に経過した時間を測定すれば良い。演算時間は計算する数の型によって異なる。従って四則演算の場合は短整数型（2バイト），長整数型（4バイト）[7]，単精度実数型（4バイト），倍精度実数型（8バイト）の各々につき行う。また数値関数の場合は単精度と倍精度の両方で行う。実行時間の決定はコンピュータに内蔵のシステム時計を利用する。MS-DOS V3.1では秒以下を測定できないので計算速度の信頼性を高めるため繰り返し数を数万回から数百万回にする必要がある。

リスト1は2バイト整数の加算を8万回実行し，その計算時間を秒単位，1回当りの計算時間をミリ秒単位で出力するF-BASIC86のプログラムと実行結果である。リスト1の計算時間はFOR～NEXTループと代入文に要する時間も含めたもので，純粋な加算$A+B$の時間ではない。しかし実際のプログラムでは純粋な加算のみということは少なく，代入文など他のルーチンが伴うのが普通である。本稿ではループや代入文も含めた計算時間とする。リスト1と同様のプログラムを他の言語でも作成し実行した結果を表2（整数型），表3（実数型）に示す。演算の繰り返し回数は全体の計算時間が100秒以上となるように決定した。従って表の結果は1％程度の誤差があるものと考えられる。表2でループというのは代入文も演算もないループのみに要する時間である。代入文$p=a$というのはループと代入文のみで演算がない場合の時間である。従って表の各演算時間から代入文$p=a$の時間を差し引いた値が演算のみの時間となる。整数除算a/bはPro FORTRANとLattice Cの表記法であるが，F-BASIC86の場合は$a¥b$，Pro PASCALの場合はa div bとなる。平均は四則演算の平均値である。比率はF-BASIC86インタプリタの平均を100とした場合の，それぞれの平均の比率である。

7）ここでは仮に，2バイト，4バイトの整数型をそれぞれ短整数型，長整数型と記すことにする。

リスト1　整数型加算の計算時間を出力する F-BASIC86プログラムと実行結果

```
10 DEFINT N,M
20 DEFINT A,B,P
30 A=1:B=2
40 NMAX=1000:MMAX=80
50 T1=TIME
60 FOR N=1 TO NMAX
70     FOR M=1 TO MMAX
80         P=A+B
90     NEXT M
100 NEXT N
110 T2=TIME
120 PRINT "integer P=A+B=";P;
130 PRINT "  time=";T2-T1;"sec     ";
140 PRINT 1000*(T2-T1)/NMAX/MMAX;"ms"
150 END

RUN
integer P=A+B= 3    time= 114 sec      1.425 ms
```

表2　整数型四則演算1回当りの計算時間（ミリ秒）

型	演　算	F-BASIC86		Pro		Lattice C
		インタプリタ	コンパイラ	FORTRAN	PASCAL	
短整数（2バイト）	ループ	0.26	0.12	0.0070	0.0061	0.0080
	$p=a$	0.91	0.19	0.0098	0.0104	0.0115
	$p=a+b$	1.4	0.28	0.023	0.024	0.013
	$p=a-b$	1.4	0.29	0.023	0.024	0.013
	$p=a*b$	1.4	0.29	0.016	0.016	0.017
	$p=a/b$	1.4	0.30	0.058	0.057	0.020
	平　均	1.4	0.29	0.030	0.030	0.016
	比　率	100	21	2.1	2.1	1.1
長整数（4バイト）	$p=a$	—	—	0.014	0.015	0.015
	$p=a+b$	—	—	0.028	0.029	0.018
	$p=a-b$	—	—	0.029	0.030	0.018
	$p=a*b$	—	—	0.067	0.066	0.050
	$p=a/b$	—	—	0.21	0.21	0.16
	平　均	—	—	0.084	0.084	0.062
	比　率	—	—	6.0	6.0	4.4

表2の結果を要約すると，整数演算1回当りの計算時間は

（1）F-BASIC86インタプリタに比べると，F-BASIC86コンパイラは約1/5，Pro FORTRANとPro PASCALは約1/50，Lattice Cは約1/100である。

（2）長整数（4バイト）は短整数（2バイト）より3～4倍。

（3）Pro FORTRANとPro PASCALはほとんど同じ。

表3（実数型四則演算）の（　）内の値は数値演算コプロセッサ付の計算時間である。表3の結果を要約すると

（1）F-BASIC86インタプリタに比べて，F-BASIC86コンパイラは約1/3～1/4，Pro FORTRANとPro PASCALは約1/12～1/6，Lattice Cは1/3～1/4。

（2）倍精度は単精度より1.2～2倍（Lattice Cを除く）。

（3）Pro FORTRANとPro PASCALはほぼ同じ。

NDPの効果は

（4）F-BASIC86では効果なし。

（5）他の言語では単精度で1/2，倍精度で1/4。

Lattice Cの単精度は倍精度より計算時間が大きくなるのは問題である。カーニハン・リッチー［10］p. 198の付録A：C参照マニュアルによると「Cでは，すべての浮動小数点演算は倍精度で実行される」とある。ということは単精度データは倍精度に変換され，倍精度演算を行い，結果を単精度に変換するという手間のかかる手続きを経ているものと考えられる。型変換を2回行うことに余分の時間がかかり，このような結果になるのであろう。従ってC言語では単精度にしても，これはメモリが倍精度より節約できるというだけで，計算速度が高くなることはない。

四則演算の実用的な評価をするために，数値積分の簡単な例[8)]を作成し実行した。プログラムの目的は積分値が既知の定積分

8）これは文献［9］で8087の効果を調べるために使われた。

表3　実数型四則演算1回当りの計算時間（ミリ秒）

（　）内はNDPつき

型	演　算	F-BASIC86		Pro		Lattice C
		インタプリタ	コンパイラ	FORTRAN	PASCAL	
単精度（4バイト）	$p=a$	0.93 (0.92)	0.19 (0.19)	0.057 (0.044)	0.057 (0.046)	0.015 (0.015)
	$p=a+b$	1.6 (1.6)	0.45 (0.45)	0.14 (0.066)	0.15 (0.066)	0.41 (0.27)
	$p=a-b$	1.6 (1.6)	0.47 (0.47)	0.15 (0.066)	0.15 (0.068)	0.42 (0.29)
	$p=a*b$	1.7 (1.7)	0.47 (0.47)	0.14 (0.068)	0.14 (0.066)	0.67 (0.27)
	$p=a/b$	2.3 (2.3)	0.46 (0.46)	0.15 (0.080)	0.15 (0.082)	0.82 (0.30)
	平　均	1.8 (1.8)	0.46 (0.46)	0.15 (0.070)	0.15 (0.070)	0.58 (0.28)
	比　率	100 (100)	26 (26)	8.3 (3.9)	8.3 (3.9)	32 (16)
倍精度（8バイト）	$p=a$	0.95 (0.94)	0.20 (0.19)	0.074 (0.046)	0.078 (0.049)	0.016 (0.016)
	$p=a+b$	1.6 (1.6)	0.61 (0.62)	0.23 (0.069)	0.23 (0.070)	0.27 (0.099)
	$p=a-b$	1.6 (1.6)	0.64 (0.64)	0.25 (0.069)	0.26 (0.069)	0.25 (0.099)
	$p=a*b$	1.9 (1.9)	0.74 (0.74)	0.34 (0.075)	0.34 (0.077)	0.54 (0.11)
	$p=a/b$	3.5 (3.5)	0.81 (0.80)	0.40 (0.082)	0.39 (0.086)	0.65 (0.11)
	平　均	2.2 (2.2)	0.70 (0.70)	0.31 (0.074)	0.31 (0.076)	0.43 (0.10)
	比　率	122 (122)	39 (39)	17 (4.1)	17 (4.2)	24 (5.6)

$$\int_0^{0.25} \frac{dx}{0.0625 + x^2} = 4\,(\tan^{-1} 1 - \tan^{-1} 0) = \pi\,(\text{円周率}) \qquad (4-1)$$

の左辺を台形公式によって数値積分し，結果を真値 π と比較する。計算時間を秒単位で表示する。台形公式では，積分区間を n 等分し，微小区間を台形と近似して，その微小台形面積を積分区間にわたって積算する。数式で表わすと積分区間（x_0，x_n）を n 等分し，きざみを h とすると

$$h = (x_n - x_0)/n,\ x_i = hi + x_0 \qquad (4-2)$$

$$s = h\,(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n))/2 \qquad (4-3)$$

s が求める積分値である。$f(x)$ は被積分関数で（4－1）の場合

$$f(x) = 1/(0.0625 + x^2) \qquad (4-4)$$

である。

計算は倍精度で行った。Lattice C によるプログラムとその実行結果をリスト2に示す。ここでは n ＝30000とした。

リスト2と同様のプログラムを他の言語でも作成し実行した結果を表4に示す。表4の最後列の Pro FORTRAN77 は引用文献［9］p. 319の結果である。これは NEC9801E（MPUは8086，クロック周波数8 MHz）で言語は Prospero Software 社の Pro FORTRAN77 による。台形公式による積分値はどの言語でも10桁まで真値（π）と一致した。従ってこのプログラム例では言語による計算精度の差はない。NDP の効果は F-BASIC86 にはないが，他の言語では4～10倍の効果がある。

平方根や三角関数，指数関数などの初等関数は科学計算で頻繁に使われる。ほとんどの高級言語にはこれらの関数を計算するルーチンが数値関数ライブラリとして用意されている。ここでは四則演算の場合と同じ方法で関数計算の時間を計測した。結果は表5のとおりである。この計算時間には四則演算のときと同様，ループと代入文の時間も含めてある。ただし，$\sin(x)$，$\cos(x)$，$\exp(x)$，$\tan^{-1}(x)$ に対しては x＝1.0とし，$\sqrt{x}$，$\log(x)$ に対し

リスト2　台形公式による数値積分
（Lattice Cによる倍精度，NDPつき）

```
 1:/*      numerical integral
 2: *      s-model, double precision with 80187
 3: *      by K.Fujimura,  02/12/89
 4: */
 5:#include  "stdio.h"
 6:#include  "math.h"
 7:#include  "time.h"
 8:
 9:extern char _NDP;
10:
11:double f(x)   double x;      { return( 1.0/(0.0625+x*x) ); }
12:double fi(x)  double x;      { return( 4.0*atan(x/0.25) ); }
13:double fdi(a,b) double a,b; { return( fi(a)-fi(b) );       }
14:
15:main()
16:{  int i,n;
17:   double h,x0,xn,x,s;
18:   unsigned long stime,etime;
19:
20:   printf("** numerical integral getting pai **¥n");
21:   printf("Lattice C  s-model  double precision¥n¥n");
22:   printf("_NDP==%d    ",_NDP);
23:   if(_NDP==0) printf("without 80187¥n¥n");
24:   else printf("with 80187¥n¥n");
25:
26:   n=30000;   x0=0.0;   xn=0.25;
27:
28:   printf("true value       = %18.15f¥n",fdi(xn,x0));
29:   h=(xn-x0)/(double)n;
30:   x=x0+h; s=0.0;
31:   time(&stime);
32:
33:           for(i=1; i<=n-1; i++) { s += f(x);  x += h;  }
34:
35:   s=h*( f(x0)+2.0*s+f(xn) )/2.0;
36:   time(&etime);
37:   printf("calculated value = %18.15f¥n",s);
38:   printf("used time        = %3d sec",etime-stime);
39:   printf("    n=%d¥n",n);
40:
41:   exit(0);
42:}
```

```
tm200
** numerical integral getting pai **
Lattice C  s-model  double precision

_NDP==1    with 80187

true value       =  3.1415926535897793
calculated value =  3.1415926535404718
used time        =  14 sec    n=30000
```

表4　台形公式による代数積分（倍精度）

（　）内はNDPつき

	F-BASIC86		Pro		Lattice C	Pro FORTRAN 77
	インタプリタ	コンパイラ	FORTRAN	PASCAL		
計算時間（秒）	259 (259)	79 (79)	40 (8)	49 (12)	62 (14)	65 (6)
比　率	100 (100)	31 (31)	15 (3)	19 (5)	24 (5)	25 (2)

（Pro FORTRAN 77は文献［9］より引用）

ては$x=2.0$とした。注意すべきことは，関数計算のときは独立変数xの値によって計算速度が異なることである。コンピュータによる関数計算では，通常，何らかの近似公式（マクローラン展開など）に従って計算する。しかし，その近似公式はxの値の大きさによって異なる。従って計算し得るxの値とその大きさに何らかの制限がついており，またxの大きさに応じて計算時間も違ってくる。各言語ごとにどのような近似公式またはアルゴリズムを採用しているかは，筆者には不明である。そこでxの大きさをいろいろ変えて関数計算を実行した結果，$\log x$の計算時間がかなり変化したが，他はあまり変化しなかった。表5の関数計算時間はあくまでも前述のxの値によるものであることを断っておく。

数値関数の計算時間について要約すると

（1）F-BASIC86インタプリタに比べて，同コンパイラは約1/2，Pro FORTRANとPro PASCALは約1/2.5

（2）倍精度は単精度より約3～4倍（Lattice Cはほとんど同じ）

（3）Pro FORTRANとPro PASCALはほとんど同じ。

NDPの効果は

（4）単精度と倍精度の差はない（Lattice Cを除く）。

（5）単精度ではF-BASIC86は約1/2，Pro FORTRANとPro PASCALでは約1/6，Lattice Cでは約1/20。

（6）倍精度では，F-BASIC86は約1/8～1/10，Pro FORTRANとPro PASCALは約1/20，Lattice Cは約1/27。

表5　数値関数1回当りの計算時間（ミリ秒）

（　）内はNDPつき

型	数値関数	F-BASIC86		Pro		Lattice C
		インタプリタ	コンパイラ	FORTRAN	PASCAL	
単精度（4バイト）	$\sqrt{x}$	1.7 (1.3)	0.83 (0.45)	0.13 (0.11)	0.12 (0.098)	4.8 (0.26)
	$\sin(x)$	3.3 (1.6)	1.5 (0.75)	1.8 (0.35)	1.9 (0.36)	10.6 (0.60)
	$\cos(x)$	3.6 (1.6)	1.6 (0.76)	2.0 (0.39)	2.1 (0.39)	11.6 (0.59)
	$\tan^{-1}(x)$	5.5 (1.4)	2.3 (0.55)	1.5 (0.20)	1.6 (0.20)	10.2 (0.38)
	$\exp(x)$	4.6 (1.6)	2.1 (0.71)	1.9 (0.26)	1.8 (0.25)	8.3 (0.43)
	$\log(x)$	2.2 (1.3)	0.96 (0.47)	1.0 (0.11)	0.99 (0.11)	2.7 (0.26)
	平　均	3.5 (1.5)	1.5 (0.62)	1.4 (0.24)	1.4 (0.23)	8.0 (0.42)
	比　率	100 (42)	44 (18)	40 (6.8)	40 (6.7)	230 (12)
倍精度（8バイト）	$\sqrt{x}$	5.5 (1.3)	5.1 (0.50)	0.77 (0.11)	0.75 (0.10)	4.7 (0.14)
	$\sin(x)$	9.4 (1.6)	6.6 (0.78)	6.3 (0.37)	6.2 (0.37)	10.5 (0.49)
	$\cos(x)$	8.7 (1.6)	6.0 (0.79)	6.3 (0.39)	6.4 (0.39)	11.4 (0.47)
	$\tan^{-1}(x)$	28.4 (1.4)	12.0 (0.56)	6.0 (0.21)	6.0 (0.20)	10.0 (0.18)
	$\exp(x)$	9.1 (1.6)	5.9 (0.73)	5.0 (0.26)	5.0 (0.26)	8.2 (0.32)
	$\log(x)$	5.3 (1.4)	2.5 (0.50)	3.1 (0.11)	2.8 (0.11)	2.6 (0.15)
	平　均	11.1 (1.5)	6.4 (0.64)	4.6 (0.24)	4.5 (0.24)	7.9 (0.29)
	比　率	316 (42)	181 (18)	131 (6.8)	129 (6.8)	226 (8.3)

(6)からNDPは倍精度関数で効果が顕著である。Lattice Cの単精度が倍精度より遅くなるのは四則演算のときと同じ理由による。また表からも分かるとおり，NDPが付いているときは単精度関数計算はNDPの倍精度計算に従っている。

四則演算のときと同様に，台形公式によって数値関数の積分を実行し，各言語とNDPの効果を調べた。積分は［9］

$$\int_{x_0}^{x_n} \left\{ \sin(x) + \cos(x) + \tan^{-1}(x) + \exp(x) + \log(x) \right\} dx$$
$$= \left[-\cos(x) + \sin(x) + x \tan^{-1}(x) - \frac{1}{2} \log(1 + x^2) + \exp(x) + x(\log x - 1) \right]_{x_0}^{x_n} \qquad (4-5)$$

で，左辺を数値積分し右辺の真値と比較する。計算時間も表示する。$x_0 = 0.5$, $x_n = 1.5$, $n = 3000$とした。各言語ごとの結果は表6となる。

表6からNDPの効果が大きいことが分かる。精度は言語によってほとんど変わらない。

表6　台形公式による関数積分（倍精度）

（　）内はNDPつき

	F-BASIC86		Pro		Lattice C	Pro FORTRAN77
	インタプリタ	コンパイラ	FORTRAN	PASCAL		
計算時間（秒）	213 (30)	114 (15)	89 (5)	89 (5)	149 (6)	135 (4)
比　率	100 (14)	54 (7)	42 (2)	42 (2)	70 (3)	63 (2)

（Pro FORTRAN77は文献［9］より引用）

5　Cray社スーパーコンピュータとの比較

D. V. Andersonはパソコンからスーパーコンピュータまで8機種のコンピュータにつき，計算速度対価格を比較している（表7）。表7の最下行は

本稿で得た計算時間から概算して記入したものである。通常，大型コンピュータでは演算速度を表わすのにMflopsという単位を用いる。これは1秒間に可能な浮動小数点演算の回数を100万単位で表わすものである。もちろん，Mflopsが高いほど，高速となる。スーパーコンピュータでは数十Mflopsに達する。表の計算速度はFORTRANによる64ビットの実数型演算である。表の結果はどのような条件で得られたかの詳細は不明であるが，本稿で示したFM16β（NDPつき）の結果からMflopsおよびRelative Costを概算してみる。FM16βシステム（ハードディスクなし，NDPつき，1MB RAM実装，中級のディスプレイとプリンタ，ソフトはMS-DOSとPro FORTRAN）の標準価格は65万円程度である。これを1＄＝130円で換算すると5k＄になる。Memoryは1mである。計算速度として，ここではPro FORTRANで得た結果を使う。倍精度実数型の四則演算1回当たりの計算時間の平均値は表3から0.074msである。この値はNDPつきの場合である。これから1秒当たりの演算回数は $1/0.074\times10^{-3} \fallingdotseq 14000$／秒。従って計算速度は0.014Mflopsとなる。Relative Costは5k＄／0.014Mflops＝0.36（10^6＄／Mflops）と算出される。これらのデータは表7の最下行に示してある。

表7　Ratio of computer cost to speed compared（文献［2］より引用）

Computer	Cost of Computer(＄)	Memory (bytes)	Mflops with FORTRAN	Relative Cost (10^6＄/Mflops)
VIC 20	1k	20k	0.00009	11.1
IBM-PC No 8087 Chip	4k	256k	0.00037	10.8
IBM-PC with 8087 Chip	5k	256k	0.0073	0.68
VAX 11／750	80k	8m	0.057	1.4
DEC KL-10	480k	1.1m	0.18	2.7
CDC-7600	4.5m	3.7m	4.6	0.98
Cray-IS	11m	16m	18	0.61
Cray XMP (2CPU's)	15m	32m	53	0.28
FM16β with 80187 Chip	5k	1m	0.014	0.36

汎用コンピュータはディスクドライブを含む初期コストである。
最下行のFM16βは本稿のデータである。

FM16βのMflopsは最高速のスーパーコンピュータCray XMPと比較して問題にならないが，Relative Costは同水準になるということは注目すべきであろう。ただし，コンピュータの価格を如何に算出するかは疑問な点が多いので表のRelative Costをそのまま信用することは問題である。一般に大型コンピュータは管理費などのオーバーヘッドは無視できない。それに対してパソコンはこのような費用はほとんどないといってよい。

パソコンのMflopsはスーパーコンピュータに比べて決定的に低い。しかし，スーパーコンピュータで開発されている並行処理方式がパソコンにも適用されればMflopsの大幅な改良がなされるはずである。

6　結　論

クロック周波数8 MHzの80186MPUをベースとしたFM16βを使って，F-BASIC86，Pro FORTRAN, Pro PASCAL, Lattice Cの数値演算機能（計算精度と速度）について比較検討した。計算精度については言語による差はあまりなかったが，計算速度に関しては大きな差がある。計算速度に関して最高の結果を示したのはPro FORTRANとPro PASCALである。この2つのソフトのメーカーは同一の会社なので，両言語とも，数値演算について同じライブラリを使っているのであろう。FORTRANはコンピュータが発明された当初から35年以上も発展，改良が重ねられてきた科学技術用言語であり，当然のこととはいえ，本稿でもそれが実証された。一方C言語は主としてシステム開発用に考案された言語であり，浮動小数点演算の計算速度のことはあまり考慮していないようである。単精度実数型演算が倍精度より高速にならないというのは問題である。科学計算でも単精度で充分という場合が多いからである。F-BASIC86インタプリタは他のコンパイラ言語より，計算速度の点で著しく劣る。しかし，もともとBASICはTSSによるプログラミングの多数初等教育用に開発された言語である。BASICインタプリタはコンパイル，リンクという複雑な操作を必要としないし，パラメータ

を何回も変えてプログラムを再実行することが容易である。F-BASIC86コンパイラはインタプリタと他のコンパイラ言語との中間といったところである。NDP の効果は特に数値関数計算で著しい。

価格対計算速度の点では16ビットパソコンは大型コンピュータとほぼ同じであるということは注目すべきと考える。Per Bak の主張するとおり，科学技術計算の分野でも一層パソコンを利用すべきであろう。

本稿の原稿を書き終った時点では，16ビットパソコンの主流は既に80286 MPU をベースにしたものに変わっている。クロック周波数も10，12，16 MHz となった。16MHz は 8 MHz に比べて演算速度は約 2 倍となる。クロック周波数が16，20，25MHz の32ビットパソコンも，個人的に入手し易い低価格で市場に出てきた。ハードの急速な進歩にソフトが追いつけないのが現状であろう。32ビット MPU の性能を完全に活用できるような OS と高級言語の開発が期待される。

なお本稿では紙数の関係で RAM メモリの利用効率やメモリモデルについては言及しなかった。これらについては機会があれば別に発表したいと思っている。また32ビットパソコンを入手次第，その結果を報告したい。

引 用 文 献

[1] Per Bak, Physics today, Dec. 1983, p. 25
[2] David V. Anderson, Physics today, Jul. 1984
[3] 富士通，F-BASIC86 V3.1 文法書
[4] 富士通，F-BASIC86 コンパイラ使用手引書
[5] Pro FORTRAN user's manual, LIFEBOAT INC. 1985
[6] Pro PASCAL user's manual, LIFEBOAT INC. 1985
[7] Lattice C Compiler user's manual, LIFEBOAT INC. 1985
[8] 富士通，日本語 MS-DOS V3.1
[9] 大貫広幸「数値演算入門」CQ 出版社，1988
[10] B. W. カーニハン，D. M. リッチー「プログラミング言語C」石田晴久訳，共立出版，1986
[11] 二階堂副包「現代経済学の数学的方法」岩波書店，1974
[12] R. ドーフマン，P. A. サミュエルソン，R. M. ソロー「線型計画と経済分析 II」安井他訳，岩波書店，1972

[13] W. R. ベネット「パソコンプログラム文科系のための問題演習」竹内他訳，現代数学社，1983
[14] J. M. マコーミク，M. G. サルバドリ「フォートランによる数値計算プログラム」清水留三郎訳，サイエンス社，昭和49年