拡張 PASCAL を核言語とした数式・数値処理ハイブリッドシステムの実現と評価†

森 和好†† 一橋正己††† 飯田三郎††

最近，数式処理と数値処理の混用ができるシステムがいくつか開発されている．Mathematica はその代表的な例であるが，著者らは数式処理の手続きと数値処理の手続きを取り込むことが可能なハイブリッドシステムを Pascal で作成した．このシステムはフロントエンドから数式処理と数値処理の手法をインタラクティブに使用可能であり，また数式処理と数値処理の間で互いにプログラムの利用，データの転送が可能である．本論文では数式処理と数値処理の混用が可能なハイブリッドシステムの作成を通して異種言語の結合の際に生じる問題点およびその対処法について具体的に明らかにした．

1. はじめに

近年，ワークステーション等の普及により数式処理システムがさまざまな研究・応用分野で利用されている．数式処理の大きな特徴は誤差を全く含まない数式の評価が可能なことにある．しかし数式的（記号的）に解くことのできない問題も存在し[1),2)]，また現実の問題では数式的に解けない場合も多い．

このようなときに数式処理システムのインタラクティブな機能を用いて数値的に解をある程度予想しながらシステムを操作できることが望ましい．さらに数式処理システムにおける数値処理は遅いという問題に対しては，記号を用いずに数値だけで計算できるような箇所については数値処理専用の記述を行うことにより高速化を計ることができる．

REDUCE[3)], MACSYMA[4)] は代表的な数式処理システムである．これらのシステムにおいても数値処理の機能が実現されているが，数値処理言語による実行に比べると遅いことが報告されている[5)]．一方，数値処理は一般に非インタラクティブに行われることが多いが，APL[6)], Speakeasy[7)], S[8)] などにみられるインタラクティブな機能は数値処理においても重要である．

このような両者を合い補わせる観点から数式処理と数値処理との混用可能なハイブリッドシステムの作成がなされてきた．Sammet et al.[9)] による FORMAC はその先駆的位置にあり，その後 Purtilo[10)], Sasaki et al.[11)], Suzuki et al.[12)] はハイブリッドシステムを数式処理と数値処理のそれぞれのプロセスによるプロセス間の通信により実現した．最近では SMP[13)], Mathematica[14)] のように，他言語で記述された数値処理機能を取り込める数式処理システムも開発されている．

数式処理と数値処理のデータの受渡しについては，その形態が三井[5)] により，実現上のいくつかの方法が Sasaki et al.[11)] により論じられている．数式処理と数値処理の混用を考えるならば，これら異種言語の手法を対等に受け渡せるようにすることが望ましい．著者らは数式処理と数値処理の既存の資源を利用でき，数式処理と数値処理の手法が対等に混用できるような単一プロセスからなるハイブリッドシステムの実現を計画し試作した．この単一プロセスの記述言語を核言語と呼ぶ．

数式処理機能の実現には数式を処理する記号処理機能の実現と数式を表現するデータ構造を定める必要があるが，著者らはこれを REDUCE に沿うものとした．このことにより REDUCE 上の数式処理の手続きを本ハイブリッドシステム上に取り込むことが可能となる．REDUCE は RLISP で記述されているが，これは Standard LISP[15)] (以下 SLISP と略す) に容易に変換される．したがって REDUCE の機能は SLISP の処理系を実現することにより達成される．一方，数値処理の手続きは Fortran のライブラリとして極めて良く整備されている．したがって核言語がこのライブラリを取り込めるような数値処理機能を有するならばハイブリッドシステムの有効性がより向上

† Implementation and Evaluation of the Hybrid System of Formula and Numerical Manipulations Using an Extended PASCAL as a Kernel Language by KAZUYOSHI MORI (Information and Computer Sciences, Faculty of Engineering, Toyohashi University of Technology), MASAMI HITOTSUBASHI (Yokohama Office, Development Laboratory, New Products Center, Shinko Electric Co., Ltd.) and SABUROU IIDA (Information and Computer Sciences, Faculty of Engineering, Toyohashi University of Technology).

†† 豊橋技術科学大学工学部情報工学系
††† 神鋼電機(株)開発本部横浜研究室

するものと考えられる.

以上のような方針から, SLISP および Fortran との親和性ならびにアルゴリズムの記述性を考え, 本システムでは核言語として Pascal を採用した.

本論文では数式処理と数値処理の混用が可能なハイブリッドシステムの作成を通して異種言語の結合・混用のさいに生じる問題点およびその対処法について具体的に明らかにすることを目的とする. 以下2章ではシステム構成, 3章ではシステム作成上の技法, 4章ではシステムの利用法, 5章では作成したシステムの評価, 6章ではまとめとして結論と今後の課題について述べる.

2. システム構成

ユーザから見たハイブリッドシステムの利用法を図1に示す. 同図においてフロントエンド (Front-End) はユーザがシステムをインタラクティブに利用することを可能とする. このフロントエンドを通してユーザは数式処理 (Formula Manipulation) と数値処理 (Numerical Manipulation) の手法を利用することができ, さらにこれらの手法からお互いに他の手法を呼び出せる.

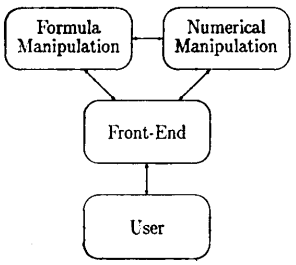


図1 ユーザとシステムのインタフェース
Fig. 1 Interface between user and system.

ハイブリッドシステムの構成を図2に示す. 以下, 図2について述べる.

(1) Formula Manipulation Part

ユーザの作成した数式処理の手続き (Formula Manipulation Procedures) は RLISP により記述される. これらの手続きは RLISP から SLISP へのトランスレータ (RLISP→SLISP Translator) により SLISP のプログラムに変換される. その後これらのプログラムは著者らが作成した SLISP から Pascal へのトランスレータ (SLISP→Pascal Translator) により Pascal に変換される. このさい Pascal に変換されたプログラムを SLISP の動作環境から利用するために識別子情報 (Identifier Information, 図中では Id. Info. と略す) も出力される.

(2) Support System Part

数式処理の手続きは記号処理にさいして SLISP の機能を使用する. このために SLISP の動作環境を Pascal により実現した. 以下, この Pascal のプログラムを SLISP 処理系 (SLISP Processor) と呼ぶ. またユーザがシステムをインタラクティブに使用するためのフロントエンドとしては REDUCE 上のフロントエンド (REDUCE Front-End) を Pascal に変換して用いた.

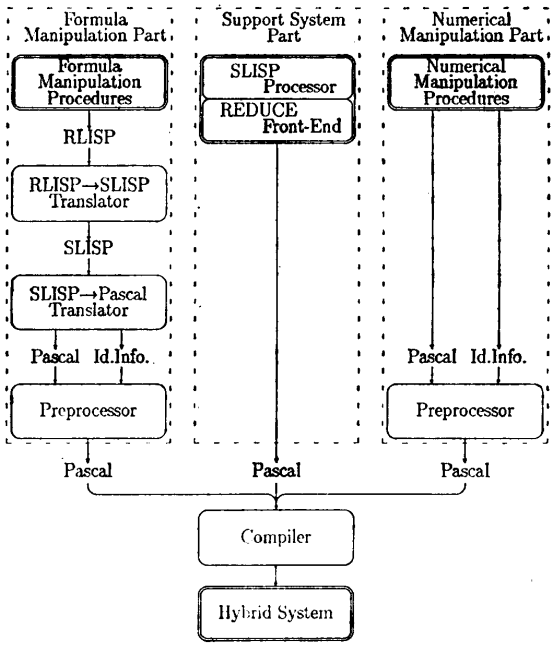


図2 システム構成
Fig. 2 System configuration.

(3) Numerical Manipulation Part

ユーザの作成した数値処理の手続き (Numerical Manipulation Procedures) は当面 Pascal 自身により記述することとした. ハイブリッドシステムのフロントエンドから数値処理の手続きを扱うさいには, この手続きを SLISP の動作環境上で利用できるように記述し, かつ識別子情報を与える.

(4) Preprocessor

Formula Manipulation Part と Numerical Manipulation Part におけるプリプロセッサ (Preprocessor) は, Pascal プログラムと識別子情報を入力として SLISP の動作環境を定めたのち, Pascal プログラムを出力する.

このようにして数式処理の手続きと数値処理の手続

きが Pascal により実現されるので，数式・数値のハイブリッド処理を行うことができる．これらの Pascal のプログラムをコンパイルしたのち，図2の単一プロセスからなるハイブリッドシステムが構成される．

3. システム作成上の技法

著者らは SELCOS 30[16] 上にシステムを作成した．ここでは作成上の技法について述べる．

3.1 SLISP 処理系

数式処理システム REDUCE を動作させるためには図2の Support System Part 内の SLISP 処理系を作成する必要がある．SLISP の基本的仕様は文献15) にあり本システムではそれに沿って作成した．以下に SLISP 処理系の核言語 Pascal による作成の概要について述べる．

3.1.1 データ

データは大別すると，整数，浮動小数点数，識別子(文献 15) における Identifier に相当)，文字列，2進セル，ベクトル，関数ポインタの7種類がある．さらに本システムでは，整数は短整数($-8\cdot16^6 \leqq z < 8\cdot16^6$)と長整数 (短整数の範囲外)，文字列は 3 文字以下と4文字以上 80 文字以下に分かれる．

データの型に関する情報は Utilisp[17] に習い，**図3**に示されるようにデータを指すポインタの側にタグとして格納する方式を用いた．タグの割当を**表1**に示す．これらのデータの表現について以下に述べる．

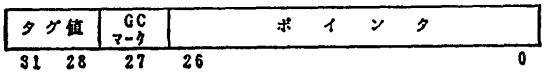


図3 タグ付きポインタ
Fig. 3 Tagged pointer.

表1 タグ値の割当
Table 1 Assignment of tag value.

タグ値	データ型
0 0 0 0	正の短整数 ($0 \leqq z < 8\cdot16^6$)
0 0 0 1	関数ポインタ
0 0 1 0	未使用
0 0 1 1	長整数 ($-8\cdot16^6 > z, z \geqq 8\cdot16^6$)
0 1 0 0	ベクトル
0 1 0 1	浮動小数点数
0 1 1 0	文字列 (3文字以下)
0 1 1 1	2進セル
1 0 0 0	識別子 (Id)
1 0 0 1	未使用
1 0 1 0	未使用
1 0 1 1	ファイルポインタ
1 1 0 0	システム予約
1 1 0 1	文字列 (4文字以上)
1 1 1 0	スタック
1 1 1 1	負の短整数 ($-8\cdot16^6 \leqq z < 0$)

(1) 短整数

短整数は即値データである．すなわちこのデータは直接に演算の対象となる．そこで表1に示すように短整数にタグを付与するとデータが2の補数表示となり，SLISP 上の短整数表現を Pascal の integer 型へ変換するためのビット操作が不要となる．このことにより Pascal の integer 型の演算がそのまま利用できるとともに，数式処理と数値処理の手続きの短整数値の受渡しを自然な形式で記述することができる．

ただし負の短整数では図3の第 27 ビット目のガーベジコレクション (以下 GC と略す) 用のマークビットが常にセットされた状態にある．一般に GC のマークビットは，該当セルが GC 時に使用されているとセットされることにより回収を防ぎ，最後にクリアされる．本システムにおいても同様なアルゴリズムを用いているが，負の短整数に対してこの操作が行われると GC のマークビットがクリアされ，データが2の補数表示とならない．したがって負の短整数については GC 時のマークビットの扱いを逆に，つまり第 27 ビットをクリアすることにより回収されることを防ぎ，最後にセットするようにした．

(2) 文字列

SLISP における文字列空間は Pascal における文字列の配列による表現と同じである．このため1～3文字で1つの配列要素を占有することは記憶領域の無駄であるので，3文字以下の文字列はポインタ自体に格納した．このことは REDUCE のフロントエンド部で EXPLODE 関数を用いているので特に有効である．

(3) 2進セル

2進セルは LISP のデータとして最も基本的であり使用頻度が非常に高い．このため高速にデータの判別ができることが望ましい．速い判別法としては符号付き整数型すなわち Pascal の integer 型の大小比較がある．そこでタグとして 0111 を与えると符号付き整数とみたときに $7\cdot16^7$ 以上のポインタは2進セルと判定 (pairp) され

$$\mathrm{pairp}(x) := (\mathrm{ord}(x) \geqq 7\times16^7)$$

である．

SLISP における関数ポインタ，長整数，ベクトル，浮動小数点数は2進セルを用いたリストで表現され

る．したがって，これらのリスト型への変換は必要不可欠であり速い変換ができることが望ましい．そこで2進セルのタグ 0111 の 28〜31 ビットをセットまたはクリアする操作により変換ができるようにタグを採用した．

SLISP 処理系における2進セル空間は配列名 PairSpace__ なる配列で構成されている．したがって数値処理の手続きは SLISP 処理系内の2進セルに対して値の参照・代入を行うことができる．SLISP の RPLACA, RPLACD に相当する数値処理手続き上の記述は

PairSpace__[P_1]. car :=Q_1

PairSpace__[P_2]. cdr :=Q_2

なる代入である．ここで P_1, P_2 は2進セルを指すポインタ，Q_1, Q_2 は任意のデータである．SLISP の CAR, CDR もこの配列の car, cdr 要素をとるので，これに相当する数値処理手続き上の記述は

PairSpace__[P_1]. car

PairSpace__[P_2]. cdr

となる．

（4） 識 別 子

識別子のタグには2進セルと同じように 1000 を与える．したがって符号付き整数とみた場合 $-7\cdot16^7$ 未満のポインタは識別子と判定 (idp) される．すなわち

idp (x) :=(ord (x)<-7×16^7)

である．

識別子も数値処理の手続きから利用できる．識別子が SLISP 上で変数として用いられているときに，文献 15) における Global Binding 機構，Fluid Binding 機構を有する変数をそれぞれ global 変数，fluid 変数と呼び，それ以外の変数を local 変数と呼ぶ．数値処理の手続き中でこれらの識別子が記述されるとプリプロセッサはこれらを識別子を指すポインタに変換する．したがって数値処理の手続きを作成するユーザはプログラミングにおいて識別子だけを知っていれば十分であり，そのポインタ値を知る必要はない．数値処理の手続きは Pascal で記述されているので Pascal の予約語と識別子が衝突する可能性がある．識別子と予約語が衝突したときには数値処理プログラム上で識別子の後に“0”（ゼロ）を付加する．例えば SLISP 上において

(CONS 'EVAL 'BEGIN)

なる表現は Pascal 上では

cons_(EVAL, BEGIN0)

とする．ここで cons_ は SLISP の CONS 関数に相当する数値処理上の関数である．識別子が特殊文字や小文字アルファベットなどを含むときは，そのアスキーコードが 16 進数で $\alpha\beta$ のとき CODE$\alpha\beta$_ としてその字面を変えた．例えば SLISP 上の識別子 !*MODE の“*”はアスキーコードが $2A_{(16)}$ なので，数値処理の手続き上では CODE2A_MODE と表現する．

SLISP 処理系における識別子の空間は配列名 IdSpace__ なるレコード型の配列からなり，インデックスは識別子を指すポインタそのものである．識別子が global 変数あるいは fluid 変数ならば，その値はポインタの指す識別子空間のレコード型の value フィールドにある．したがって数値処理の手続きは識別子の値を

IdSpace__[識別子名]. value

として得ることができる．もちろん値の代入もできて

IdSpace__[識別子名]. value :=式

とすればよい．local 変数は SLISP の仕様においてスコープがローカルなのでスコープ外からの参照・代入をすることはない．

（5） 関 数

SLISP の関数を数値処理の手続き上で用いるときは関数名の後に“_”（アンダスコア）をつける．予約語との重複や大文字アルファベット以外の使用は識別子の場合と同様に扱い“0”のあとにアンダスコアをつける．

本システムでは SLISP の関数の型として EXPR, FEXPR, MACRO を用いることができる．仮引数と実引数との対応関係は次のとおりである．

EXPR ：実引数を評価・展開して仮引数と1対1に対応した束縛がなされる．

FEXPR ：1つの仮引数に非評価・非展開の実引数のリストが束縛される．

MACRO：1つの仮引数にマクロ名と非評価・非展開の実引数からなるリストが束縛される．

数値処理の手続きによる SLISP 関数の利用の場合も上記の対応関係に従うように作成する．したがって次のように使用する．

EXPR ：SLISP 上の実引数と同数の仮引数をもち，それらはすでに評価されているとする．

FEXPR ：1つの仮引数で非評価の実引数リストを

受け取る.

MACRO: 1つの仮引数でSLISP上の関数名と非評価の実引数からなるリストを受け取る.

REDUCE上の関数はSLISP処理系内に関数として定義されているので，ここで述べた方法を用いてREDUCEの関数を数値処理の手続きで利用できる．例えば前述の(3)におけるSLISPのRPLACA, RPLACDは，配列の操作以外にrplaca_, rplacd_を用いてもよい.

表2 マクロ ($x_1, x_2, \cdots, x_{10}$ は適当な式)
Table 2 Macro.

記法	等価な式	対応するSLISPの関数
rplaca•(x_1, x_2)	PairSpace_-[x_1].car:=x_2	RPLACA
rplacd•(x_1, x_2)	PairSpace_-[x_1].cdr:=x_2	RPLACD
car•(x)	PairSpace_-[x].car	CAR
cdr•(x)	PairSpace_-[x].cdr	CDR
caar•(x)	PairSpace_-[car•(x)].car	CAAR
⋮	⋮	⋮
cddddr•(x)	PairSpace_-[cdddr•(x)].cdr	CDDDDR
list1•(x_1)	cons_(x_1, NIL0)	LIST
list2•(x_1, x_2)	cons_(x_1, cons_(x_2, NIL0))	LIST
⋮	⋮	⋮
list10•($x_1, \cdots, x_{10}$)	cons_($x_1, \cdots$, cons_(x_{10}, NIL0)$\cdots$)	LIST

3.1.2 実行時スタック

SLISP（あるいはLISP一般）の処理系をPascalで作成すると，プログラムの実行中のデータは実行時スタックに置かれる．そのときにGCが起こると本システムではGCの方式としてリンクたどり一括回収(Mark & Sweep)方式を採用しており，Pascalのプログラム自体からは実行時スタックの内容を見ることができない．例えば2引数をとる関数 fn を考えたとき

fn(cons_(0, 1), cons_(2, 3))

なるPascalの記述があるとする．仮にcons_(2, 3)の実行中にGCが起きたとする．このときcons_(0, 1)の結果のセルを指すポインタは実行時スタック中にあり通常の方法ではマークすることはできない．そこで著者らは機械命令で記述された手続きを用いて直接に実行時スタックのスタックポインタを知り，スタックフレーム中のアドレスからデータをマークする方法を採用した．このことによりPascalの機能を一部拡張する必要が生じた.

3.1.3 マクロ

数値処理のプログラムを書くとき，配列名を記述することは字面を長くして見づらい．そこで本システムでは表2に示されているマクロを用意した．これらの記法を用いてプログラムを記述したときには，後述するプリプロセッサが表中の等価な式のプログラムに変換する.

3.2 トランスレータ

図2のFormula Manipulation Part内の，SLISPからPascalへのトランスレータはPascalにより記述されておりハイブリッドシステム上に実現されている．トランスレータはトランスレータの開始と終了の間に定義されたSLISPの関数をPascalに変換する．さらにそのPascalの関数をシステムに取り込むときに必要な手続きと識別子情報も出力する.

SLISPからPascalへの変換上の問題点として，SLISPの関数の引数として自由変数を含むラムダ式が現れる場合があげられる．一般にRLISPのFOR EACH文はMAP関数族を用いたS式に変換され，そのとき関数の引数として自由変数を含むラムダ式が生成される場合がある．REDUCE内にはそのような箇所があるが，自由変数を含むラムダ式が現れるのはMAP関数族の引数としてだけである．そこで著者らは，このようなときにはMAP関数族をマクロのクラスとして捉え[18]，自由変数をもつラムダ式をPascalのローカル関数として達成した．Pascalのローカル関数を用いることによりSLISPの変数環境を自然にPascal上で実現でき，SLISP上のラムダ式内の自由変数であるlocal変数はPascal上の外側の関数のローカル変数として存在する．したがって内側のローカル関数内でも自由変数に対する参照・代入ができる.

3.3 プリプロセッサ

図2のFormula Manipulation Part内のプリプロセッサは識別子情報によりSLISPプログラム内の識別子のポインタを定める．一方，同図のNumerical Manipulation Part内のプリプロセッサはPascalで記述したプログラム中の識別子をポインタへ変換し，またマクロ展開を行う．このようにして出力されたプログラムはPascalコンパイラの入力となる.

4. システムの利用法

4.1 数式処理手続きの作成とその利用法

数式処理プログラムはフロントエンドから直接入力できるので RLISP により記述する．これらのプログラムをコンパイルして利用する場合にはトランスレータにより Pascal に変換する．Pascal に変換する場合のハイブリッドシステムの利用例を**図4**に示す．この例では自然対数を施した Stirling の近似式を関数として定義したのち Pascal に変換して新たにハイブリッドシステム内に取り込んでいる．

4.2 数値処理手続きの作成とその利用法

数値処理の手続きをハイブリッドシステム上のフロントエンドから利用するためには，3章で述べたように SLISP 処理系にインタフェースを合わせるとともに識別子情報を与える．

数式処理手続きの作成にさいして SLISP 上の浮動小数点数データを Pascal の倍精度実数型に変換する場合，あるいはその逆変換の場合には，関数として Pfloat2double___ と Pdouble2float___ が用意されておりユーザはこれらを用いる．短整数を Pascal の integer 型に変換するには ord 関数を用いる．逆変換は通常コンパイラにより自動的になされる．

識別子情報としては，SLISP の印字名，属性，束縛型・関数型，変数の場合は値，関数型で EXPR 型のときには引数の数を与える．この情報を図2の Numerical Manipulation Part 内のプリプロセッサを通すことにより識別子空間の割当がなされ，Pascal のプログラムが出力される．また数値処理の手続きからマクロを使用している場合は，プリプロセッサによりマクロの展開が行われる．

図5に数値処理の手法により Stirling の近似式を関数として定義した例を示す．図5のプログラムを動作させるときには，Stirling の近似式の識別子情報と

```
$ hybrid                                              ⟶ ハイブリッドシステム起動
1: TRANSSTART() ;                                     ⟶ トランスレート開始
2: SYMBOLIC PROCEDURE STIRLING(ARGS) ;                ⟶ 関数定義
   BEGIN
     REAL  A,X ;
     X:=REVAL(CAR(ARGS)) ;
     A:=0.5*LOG(2*3.14159265358979 3)+0.5*LOG(X)
                                   +X*(LOG(X)-1)+LOG(1+1/(12*X));
     RETURN (('!:FT!: . A) . 1) ;
   END ;
3: SYMBOLIC PUT('STIRLING,'SIMPFN,'STIRLING) ;        ⟶ ALGEBRAIC モードで使用
4: TRANSEND() ;                                       ⟶ トランスレート終了
5: BYE ;                                              ⟶ ハイブリッドシステム終了
$ preprocess stirling.h                               ⟶ プリプロセッサ
$ Lpc -o hybrid hybrid.p                              ⟶ コンパイル
$ hybrid                                              ⟶ ハイブリッドシステム起動
```

図4 数式処理の手法における Stirling の近似式のプログラム（下線部利用者入力）
Fig. 4 Program of Stirling's approximation using formula manipulation.

```
function stirling_(args:ptr__):ptr__ ;
const   pi      = 3.14159265358979 3(*238...*) ;
var     a,x     : double ;
begin
    x:=float2double___(reval_(car_(args))) ;
    a:=0.5*log(2*pi)+0.5*log(x)+x*(log(x)-1)+log(1+1/(12*x)) ;
    stirling_:=cons_(cons_(CODE3A_FTCODE3A_,double2float___(a)),1) ;
end ;
```

図5 数値処理の手法における Stirling の近似式のプログラム
Fig. 5 Program of Stirling's approximation using numerical manipulation.

して SLISP 処理系上の関数名，属性 (SIMPFN)，関数の型 (EXPR) と引数の数（1つ）を与える．

4.3 インタプリタの利用法

SLISP 処理系には APPLY, EVAL 関数が作成されている．したがって数値処理の手続きは，これらの関数を用いて Pascal に未変換の SLISP で記述された数式処理の手続きを利用できる．すなわち，SLISP で記述された数式処理手続きはインタプリタにより実行される．一方，トランスレータにより SLISP から Pascal に変換された数式処理手続きは Pascal コンパイラで処理されたのち実行される．

5. 評　　価

数式処理プログラムとして REDUCE パッケージ中の "ALG1. RED", "ALG2. RED", "MATR. RED", "SOLVE. RED" をハイブリッドシステム上に移植した．さらに同一機能のプログラムを RLISP, Pascal によりそれぞれ記述し計算時間の測定値から両者を比較した．

プログラムとしては，(1)データ量の大きな問題として連立一次方程式の解法，(2)データ量が一定でかつ数値処理の手法だけを用いる数値関数（ガンマ関数）の計算，(3)逐次的に数式処理の手法を用いる常微分方程式の解法を用いた．ただし，数式処理プログラムに対しては REDUCE の SYMBOLIC モードで記述し ALGEBRAIC モードで使用した．

以下の記述において，処理時間 (Processing Time) はプログラムの実行開始から終了までの時間，演算時間 (Execution Time) は処理時間からデータ変換時間を除いた時間とする．また RLISP により記述されたプログラムを数式処理手続き，Pascal で記述されたプログラムを数値処理手続きと呼び

$$R_p=\frac{(\text{数式処理手続きによる処理時間})}{(\text{数値処理手続きによる処理時間})}$$

$$R_e=\frac{(\text{数式処理手続きによる演算時間})}{(\text{数値処理手続きによる演算時間})}$$

$$R_f=\frac{(\text{数式処理手続きによる演算時間})}{(\text{数式処理手続きによる処理時間})}$$

$$R_n=\frac{(\text{数値処理手続きによる演算時間})}{(\text{数値処理手続きによる処理時間})}$$

により R_p, R_e, R_f, R_n を定義する．

5.1 連立一次方程式

データとして，10, 20, 30, 40, 50 元の連立一次方程式に対してガウスの消去法を用いた際の処理時間，演算時間を測定した．ここで数式処理手続き，数値処理手続きともデータ構造としてそれぞれの配列すなわち RLISP 上の ARRAY と Pascal 上の array を用いている．

データの受渡しはハイブリッドシステムに対応させた．すなわち数式処理手続きにおいてはデータを数値的に扱えるように Pascal の倍精度実数型に変換し，算出された解は RLISP に対応するように変換される．出力形式は REDUCE の SOLVE 関数と同一である．

R_p, R_e の測定結果を**図 6** に示す．同図より数値処理手続きは数式処理手続きに対し，処理時間で約 3.5 倍，演算時間で約 250 倍速い結果が得られた．演算時間は大幅に高速化されたが，このことは

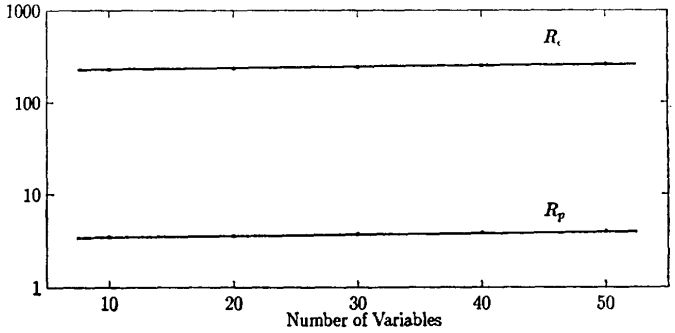


図 6 連立一次方程式の数式処理と数値処理の比較
Fig. 6 Comparison of formula manipulation with numerical manipulation on processing and execution times of a simultaneous linear equation.

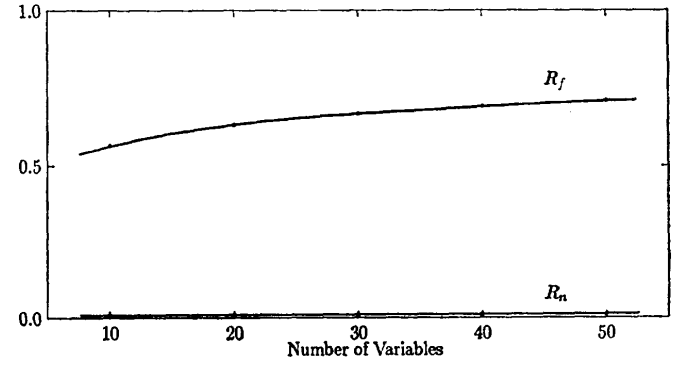


図 7 連立一次方程式の処理時間と演算時間の比較
Fig. 7 Comparison of execution time with processing time on formula and numerical manipulations of a simultaneous linear equation.

●数式処理手続きでは(1)演算の方法として SLISP の動作環境に依存する関数の使用，(2)2進セルの使用，(3)演算関数内における引数の型検査，

が行われるのに対し，

●数値処理手続きでは(1)配列要素へのアクセスの単純化，(2)実引数の型の非検査，

に起因すると考えられる．演算時間に対し処理時間の高速化が十分得られなかったことを調べるために R_f, R_n を求めた．結果を図7に示す．同図より R_f, R_n はそれぞれ約 0.6，約 0.01 である．R_n が約 0.01 であるということは数値処理手続きにおけるデータ変換のオーバヘッドが大きいことを示す．したがってこのオーバヘッドの軽減が数値処理における処理時間の高速化に必要なことが判明した．

5.2 数値関数

数値演算のみを行うプログラムとしてガンマ関数に対し評価を試みた．このプログラムは短整数または浮動小数点数を引数とする．短整数の場合はテーブルから値を取り出し返す．浮動小数点数の場合は再帰呼出しにより引数 X を $0.5 \leqq X \leqq 1.5$ の範囲に帰着させてマクローリン展開した式から値を導く．今回の測定では浮動小数点数のデータを扱った．このプログラムでは入力データはガンマ関数の引数であり，データ変換は浮動小数点数に対してのみ行われる．

R_p, R_e の測定結果を図8に示す．同図においてマクローリン展開式に要する時間が全体の演算時間の大半を占めると考えられる実引数値の小さな場合を除き，数値処理手続きは数式処理手続きに比べて処理時間で約 70～80 倍，演算時間で約 175 倍の速度向上が得られ連立一次方程式の場合よりも良好な結果を得ることができた．

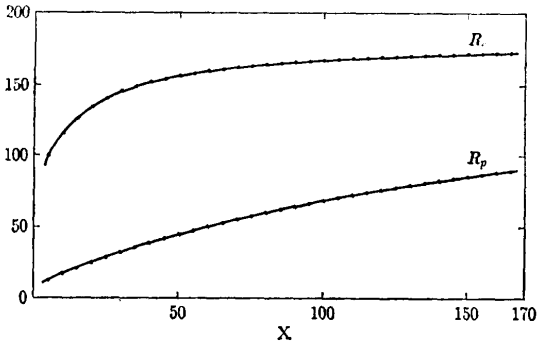


図 8 ガンマ関数 $\Gamma(X)$ の数式処理と数値処理の比較
Fig. 8 Comparison of formula manipulation with numerical manipulation on processing and execution times of gamma function $\Gamma(X)$.

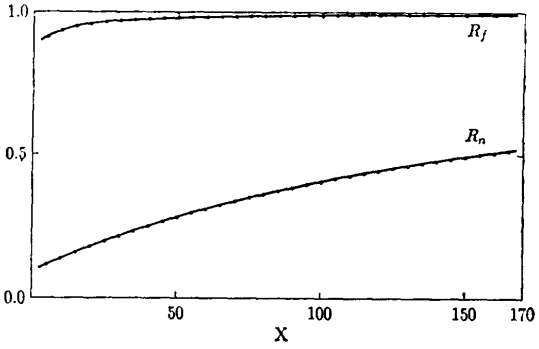


図 9 ガンマ関数 $\Gamma(X)$ の処理時間と演算時間の比較
Fig. 9 Comparison of execution time with processing time on formula and numerical manipulations of gamma function $\Gamma(X)$.

連立一次方程式のときと同様に R_f, R_n を求めた．結果を図9に示す．同図においてガンマ関数の実引数値が大きくなると R_f はほぼ1となり，R_n も 0.4～0.5 となる．ガンマ関数のように変換データ量の少ない場合は数値処理手続きにおけるデータ変換のオーバヘッドが現れず，演算時間をそのまま処理時間に反映させることができ本ハイブリッドシステムの有効性を確認できた．

5.3 連立常微分方程式

このプログラムは，$X'=F(X,u)$ なる形式の方程式と初期値より，ルンゲ・クッタ法を用いて解を算出する．プログラムの引数は数式と初期値である．数値処理手続きは演算中に数式の逐次的評価を必要とする．この数式はハイブリッドシステム上の数式処理プログラムにより評価されたのち Pascal の倍精度実数型に変換されて数値処理手続きに返される．このさい数値処理における処理時間中のデータ変換は初期値に対する変換のみと考え，数式評価のためのデータ変換時間は演算時間の一部とみなした．したがって演算時間はほぼ処理時間に等しい．

表 3 連立常微分方程式の数式処理と数値処理の比較
Table 3 Comparison of formula manipulation with numerical manipulation on processing time of simultaneous ordinary differential equations.

方程式	初期値	R_p
① $y_1'=y_1$	$y_1(0)=1$	3.59
② $y_1'=y_2$ $y_2'=y_1$	$y_1(0)=0$ $y_2(0)=1$	2.90
③ $y_1'=y_2-x+2$ $y_2'=-y_1+x+1$	$y_1(0)=0$ $y_2(0)=0$	2.01

いくつかの連立常微分方程式に対して測定した R_p を表3に示す．ここで方程式系は①，②，③の順に複雑となる．同表より数値処理手続きを用いることにより数式処理手続きに対し 2.01～3.59 倍の速度向上が得られた．このことから数値処理手続き内において数式処理手続きを用いてもハイブリッドシステムが有効に機能することが確認できた．

6. おわりに

（1） 著者らは，数式処理の手法と数値処理の手法をインタラクティブに使用できる単一プロセスで構成されたハイブリッドシステムを，核言語 Pascal を用いて作成した．そしてこれらの手法を結合・混用するさいに生じるいくつかの問題点およびその対処法を具体的に明らかにした．数式処理の手法は RLISP で記述できるので REDUCE のプログラムを本システム上で稼働することができる．一方，数値処理の手法は現在 Pascal で記述される．いくつかのプログラムを用いてシステムの性能を評価した結果，このようなハイブリッドシステムの数式・数値混用処理にさいする有効性を確認することができた．

（2） 本システムでは GC にさいし実行時スタックの走査機能を効率上の観点から用いた．このことにより Pascal の機能を一部拡張する必要が生じ移植性が若干損なわれることとなった．この問題は例えばアドレスへの参照・代入機能を持つ Turbo Pascal[19] 等を用いることにより回避できる．また循環リストの回収可能な参照カウント法[20]を用いれば時間的，空間的効率が低下するが，Pascal の機能を拡張する必要性は生じない．

さらに，本システムのような異種環境上のリスト処理においては Boehm[21] らにより提案されているタグを用いない方法も移植性の観点から有効であると考えられる．

（3） 著者らは，数式処理システムとして広く利用されている REDUCE の処理言語である SLISP ならびに Fortran で記述された数値計算ライブラリとの親和性，さらにアルゴリズムの記述性の観点から核言語として Pascal を用いた．これに対し SLISP の機能を拡張することにより SLISP における数値処理の実行効率を向上させることも考えられる．例えば配列の実現法，データの型検査法等に対しソフトウェア，ハードウェア両面の機能を拡張すれば，図8の R_p にみられるような大幅な差異は減少するものと考えられる．しかし少なくとも Pascal における数値演算程度の処理効率を有する機能拡張が必要となる．著者らは SLISP の機能拡張は将来の検討課題とした上で，本システムを Pascal を用いて作成した．

一方 Fortran を核言語すると，SLISP のような再帰プログラムを Fortran に変換するトランスレータの作成の容易さが欠如するものと考えられる．またCを核言語とすると，3.2 節で述べた自由変数を含むラムダ式は，Pascal による変換ほど自然な形には変換されない．このことは Pascal が実用性よりもアルゴリズムの記述性に重点をおいていることの現れであると考えられる．

（4） 本システムは制御用計算機上に作成されたのでハードウェア的には利用可能な記憶空間が狭く，ソフトウェア的には低機能なオペレーティングシステムを使用するという強い制限を受けた．しかしこのようなインタラクティブなハイブリッドシステムにおいては，数値処理の手続きに対しても動的ロードが行えることがシステム利用上重要な機能であると考えられる．したがって今後は動的ロードが可能なオペレーティングシステムのもとで本システムを再構築し，Fortran で記述された数値処理ライブラリを用いて大規模な問題におけるハイブリッドシステムの有効性について検討する予定である．これらについては別途報告する．

謝辞 ハイブリッドシステムに関する情報を頂いた理化学研究所佐々木建昭氏，名古屋大学工学部三井斌友助教授，数式処理および REDUCE に関する助言を頂いた豊橋技術科学大学阿部健一教授，斉藤制海教授，さらに PASCAL コンパイラについての情報を数多く頂いた神鋼電機(株)横浜研究室，CS 本部の諸氏に感謝する．

参考文献

1) Davis, M., Putnam, H. and Robinson, J.: The Decision Problem for Exponential Diophantine Equations, *Annals of Mathematics*, Vol. 74, pp. 425-436 (1961).
2) Richardson, D.: Some Undecidable Problems Involving Elementary Functions of a Real Variable, *J. Symbolic Logic*, Vol. 33, No. 4, pp. 514-520 (1968).
3) Hearn, A.C. (ed.): REDUCE User's Manual (Ver. 3.2), The Rand Corporation, Santa Monica, CA. (1985).
4) MATHLAB Group: MACSYMA Reference Manual, MIT, Mass. (1977).

5) 三井斌友：数式処理と数値処理との界面，情報処理，Vol. 27, No. 4, pp. 422-430 (1986).
6) Iverson, K. E.: *A Programming Language*, John Wiley & Sons, New York (1962).
7) Speakeasy Computing Corporation: Speakeasy, Chicago, Illinois.
8) Becker, R. A. and Chambers, J. M.: *S; An Interactive Environment for Data Analysis and Graphics*, Wadsworth, Belmont, CA. (1984).
9) Sammet, J. E. and Bond, E. R.: Introduction to FORMAC, *IEEE Trans. Electron. Comput.*, Vol. EC-13, pp. 386-394 (1965).
10) Purtilo, J. M.: Application of a Software Interconnection System in Mathematical Problem Solving Environment, *Proc. SYMSAC '86*, pp. 16-23 (1986).
11) Sasaki, T., Fukui, Y., Suzuki, M. and Sato, M.: Proposal of a Scheme for Linking Different Computer Languages, *Preprint of ICPR* (1986).
12) Suzuki, M., Sasaki, T. and Fukui, Y.: A Hybrid Algebraic-Numeric System ANS and Its Preliminary Implementation, *LNCS, Vol. 378, Proc. EUROCAL '87*, pp. 163-172, Springer Verlag (1987).
13) 中山治久：SMP, bit 別冊，計算機による数式処理のすすめ，pp. 217-224, 共立出版 (1986).
14) Wolfram, S.: *A System for Doing Mathematics by Computer*, Addison-Wesley (1988).
15) Marti, J. B., Hearn, A. C., Griss, M. L. and Griss, C: Standard LISP Report, Univ. of Utah, Utah (1978).
16) 氷上美昭，花井泰政，伴 正和，信貴幹夫：制御用計算機「SELCOS-30」，*Shinko Electric Journal*, Vol. 30, No. 3, pp. 5-16 (1985).
17) 近山 隆：Utilisp システムの開発，情報処理学会論文誌，Vol. 24, No. 5, pp. 599-604 (1983).
18) Murata, T. and Iida, S.: An Approach to the FUNARG Problem Using the MACRO Facility, *J. Inf. Process.*, Vol. 11, No. 2, pp. 120-122 (1988).
19) Borland International: Turbo Pascal リファレンスガイド，(株)マイクロソフトウェア アソシエイツ編訳 (1989).
20) Kakuta, K., Nakamura, H. and Iida, S.: A Parallel Reference Counting Algorithm, *Inf. Process. Lett.*, Vol. 23, No. 1, pp. 33-37 (1986).
21) Boehm, H. and Weiser, M.: Garbage Collection in an Uncooperative Environment, *Softw. Pract. Exper.*, Vol. 18, No. 9, pp. 807-820 (1988).

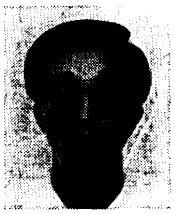
(平成元年8月 3 日受付)
(平成2年7月10日採録)

森 和好 (正会員)

1988 年豊橋技術科学大学工学部情報工学科卒業．1990 年同大学大学院修士課程情報工学専攻卒業．現在，同大学大学院博士後期課程システム情報工学専攻に在学中．数式処理システム，数式処理アルゴリズムに興味を持つ．

一橋 正己 (正会員)

1979 年大阪大学理学部物理学科卒業．同年，神鋼電機(株)入社．制御用計算機のプログラム開発の後，リアルタイム OS の開発に従事．現在，同社開発本部 VAR 開発室勤務．並列処理アーキテクチャ，オブジェクト指向言語，3次元コンピュータグラフィックスに興味を持つ．



飯田 三郎 (正会員)

1966 年東京大学理学部天文学科卒業．1968 年名古屋大学大学院修士課程物理学専攻卒業．1971 年同博士課程修了．現在，豊橋技術科学大学工学部情報工学系に勤務．計算機アーキテクチャ，数式処理アルゴリズムに興味を持つ．