行列計算のためのMATLABベース静的型付け言語の設計と実装

川端英之[†] 北村俊明[†]

数値計算プログラム記述に広く用いられている言語の 1 つに MATLAB がある．MATLAB は各変数が行列を表し，また密行列と疎行列をコード上でほとんど区別なく扱えるので，行列計算を簡潔に記述できる．しかしながら MATLAB は動的型言語であるために処理速度に難があり，特に大規模な疎行列を扱う行列計算や頻繁な手続き呼び出しを含む計算には使いにくい．これに対し我々は，行列計算を高速に処理できかつ記述のしやすい言語処理環境の実現を目的とし，MATLAB をベースとする記述から C や Fortran 90 などのコンパイル型言語によるプログラムを生成する処理系を開発している．本論文では，このたび新たに改良を加えた行列計算記述向け言語処理系 CMC の仕様について述べる．CMC の入力言語は MATLAB をベースとし，静的型とすることで実行時のオーバヘッドを極力排除して高速化を図るという方針で設計されており，いくつかの実測により効果が確認されている．しかし，従来の言語仕様は柔軟性に欠けるもので，MATLAB との互換性が十分ではなかった．これに対して，新たに代入による暗黙の変数宣言に対応し，同一名の変数を同一手続き中の複数箇所で別の型で扱うことができるようにした．また，値呼びによる引数授受に対応し，行列添字式の動的チェックや行列サイズの動的拡張を行う箇所をユーザが指定できるようにした．ハウスホルダ QR 分解を用いた実測により，実行速度を高く保ちながら MATLAB との互換性の高い記述に対応することの可能性が確認できた．

Design and Implementation of a MATLAB-based Statically-typed Language for Matrix Computations

Hideyuki Kawabata[†] and Toshiaki Kitamura[†]

MATLAB is a programming language that is used widely for implementing matrix computations. Variables of the MATLAB language are of (sparse or dense) matrix type and operators of the language support matrix computations, so that MATLAB programs tend to be quite simple and readable. However, because of its dynamic nature, the execution speed of programs in MATLAB does not compete with that of codes written in compiled languages such as C or Fortran, especially when computations with large-scale sparse matrices are involved. In order to embody a computational environment where the user can implement efficient programs for matrix computations easily, we have been developing a compiler which translates programs in MATLAB-based language, named CMC, into Fortran 90. The basic idea of our approach for speeding up executions is to adopt static typing while keeping the features of MATLAB as many as possible, and some experimental results have been confirmed that our approach is effective. In this paper, we show new features of CMC. We refined the language specification to make CMC more compatible with MATLAB without loosing the ability of generating fast codes. The modified version of CMC has a few new features. Implicit type declaration with assignment allows the user to deal with variables of different types by the same name. Call-by-value is supported. Dynamic array-bounds checking and enlargement of array will be done at explicitly specified points by the user. Experimental results on Householder QR factorization algorithm have exhibited the effectiveness of CMC's approach.

1. はじめに

大規模数値計算プログラムは煩雑なものになりがちで，特に大規模疎行列を扱うプログラムはその開発や保守が一般に容易ではない．これに対して MATLAB をはじめとする行列計算実行環境がプロトタイピング用途に広く用いられているが，プロダクションラン用コード開発には依然として Fortran などの低水準言語が頼られている．その主たる理由は従来の行列計算環境の処理速度の低さである．本論文は，MATLAB をベースとする静的型付け言語処理系の実装例を通して，行列計算向きの高水準言語による効率良い大規模数値計算コード開発環境の実現可能性を示すものである．

† 広島市立大学情報科学部情報工学科
Department of Computer Engineering, Hiroshima City University

MATLAB は MathWorks 社が開発している数値計算向けプログラミング環境で[23]，ユーザはこれを用いることによって一般的な行列計算アルゴリズム記述に似通った表現によってプログラムを記述することができる．また MATLAB 処理系は柔軟な動的処理をサポートしていて，ユーザは記憶管理の煩わしさを感じることなくベクトルや行列などのデータ構造を扱うことができるし，動的型付けであるため変数宣言も必要ない．しかしながら，MATLAB 実行環境の処理速度は大規模な計算を行うには十分とはいい難く，特に大規模な疎行列を扱う行列計算や頻繁な手続き呼び出しを含む計算には使いにくい．

MATLAB プログラムの高速処理に関連する研究は数多く行われている．たとえば FALCON[12] は，MATLAB コード記述を Fortran 90 記述に変換することによって高速化を図るコンパイラである．変換にあたっては，変数の型や形状などの属性値を可能な限り静的に推定し，動的処理の必要性を削減しようとする．しかし FALCON は大規模数値計算において必須の疎行列データ構造を扱う機能に対応していない．また，FALCON では関数☆呼び出しをすべてインライン展開するので，多数の関数からなる MATLAB プログラムの処理には向かない．

ソースコードの型の静的推定機能に加えてライブラリルーチンの「特殊化」による自動最適化機能を統合した Telescoping Languages と呼ばれる枠組み[9),11),22)]も提案されているが，仕様の詳細はまだ明らかとはいえず，疎行列の扱いにも具体的な言及がない．

MATLAB コードを C 言語記述に変換し，さらに既存の数値計算パッケージを組み合わせて高速化を図る研究もある[28),29)]．しかしこれらも疎行列データ構造を考慮していない．

MATLAB の開発元である MathWorks 社は，疎行列データ構造の扱いを含めた MATLAB 記述を C 言語などのコンパイル言語へ変換できる MATLAB Compiler を開発している[24]．しかしその最適化能力は高くなく，実際 MATLAB Compiler では loop-intensive なコードしか高速化が望めないことが指摘されている[12),27)]．

MATLAB プログラムをソースコードレベルで最適化して高速化する研究も行われているが[27)]，適用範囲が限られる．また，提示されているコード変換機能を実現した処理系は我々の知るところでは実装されていない．

これらをふまえ，我々は，行列計算を高速に処理できかつ記述のしやすい言語処理環境の実現を目的とし，MATLAB をベースとする記述から C や Fortran 90 などのコンパイル型言語によるプログラムを生成する言語処理系 CMC を開発している[20),21)]．CMC は次の特徴を持つ：

- 入力言語の構文は広く使用されている行列計算向け言語である MATLAB をベースとしている．
- コンパイル言語への変換に必要な変数の型などの属性値を静的解析により自動決定する．ユーザは関数の引数などの自動決定が困難な変数属性しか指示せずに済む．
- 大規模数値計算記述に必須の，疎行列のためのデータ構造と演算に対応している．
- 三角行列や対角行列などの行列の形状情報の解析機能を持ち，それをふまえたコードを出力できる．
- 行列言語記述のソースコードレベルでの最適化機能を持つ．古典的最適化機能に加え，行列の形状やサイズをふまえた行列演算の計算順序決定，連続した行列の乗算や行列の積和式などの最適化など，行列言語向けの最適化機能を有する．

CMC が目指すのは，大規模行列計算アプリケーションの記述が可能な静的型付け高水準言語とその処理系の実現である．言語の仕様を MATLAB に完全準拠させることは目的ではないが，CMC 処理系において正しい挙動をするプログラムは MATLAB の構文を満足するように保ちたいと考えている．またユーザの負担軽減のために変数の属性値宣言の必要性をできるだけ削減したいので，関数記述中で属性値の宣言が必要なものは，仮引数，および，内部で呼び出している非組み込みの関数のインタフェースのみとしたい（関数の仮引数の属性値に関する宣言をユーザに求めることは，その関数の可読性や保守性の向上に好都合であると思われるし，記述の静的なエラーチェック機構の実装の容易さの観点でも有用であると我々は考えている）．このような方針の下で開発した CMC 処理系は，ユーザ指示行によって部分的に変数宣言を挿入した MATLAB プログラムを Fortran 90 コードに変換するもので，疎行列データ構造も容易に扱うことができる．さらに詳細な行列の形状情報を最適化に利用する機能を持っている．本処理系は，SOR 法や共役勾配（CG）法などの基本的な疎行列計算アルゴリズムについて，その効果が実測で確認されている[20),21)]．しかしながら，扱える言語の仕様が，MATLAB の構文に対して単純に静的型付けの制約を施したものであ

☆ 本論文は，特に記さない限り，「関数」という用語を MATLAB における関数，すなわち，MATLAB プログラムを構成する個々の M-file で定義される処理単位を指すものとして用いる．

り，プログラム中の個々の変数（すなわち行列）のサイズをプログラム中で変更することができず，柔軟性は MATLAB に対して低かった．

これに対し我々は，CMC の仕様に改良を加え，より柔軟な，MATLAB 言語と親和性の高いプログラム記述を可能にした．本論文では，このたび新たに改良を加えた行列計算記述向け言語の仕様と実装について述べる．新仕様では，従来の設計における強い制約を取り払い，行列サイズの動的変更を可能にした．また MATLAB と同様，基本的にすべての代入文を暗黙の変数宣言に対応させるべく，ある条件の下で同一名の変数を同一手続き中の複数箇所で別の型で扱うことができるようにした．これらの機能拡張により，プログラムの実行速度を高く保ちながら，より柔軟かつ MATLAB との互換性の高い記述を可能にした．現在，Fortran 90 への変換系を実装中である．MATLAB によって記述されたハウスホルダ QR 分解プログラムを用いて予備評価を行ったところ，MATLAB よりも平均して 3 割程度高速化されることが確認できた．

2. 準備：MATLAB とその高速化について

2.1 MATLAB の概要

MATLAB プログラムの例を**図 1** に示す．図 1 (a) はべき乗法のアルゴリズムの一般的な表現であり，図 1 (b) はそれを MATLAB における関数として記述したものである．MATLAB では変数 1 つ 1 つがそれぞれ行列を表す．また，行列の積や転置などの操作が基本演算として言語仕様に取り込まれている．図 1 (a) と図 1 (b) の類似の様子からも分かるように，MATLAB では数値計算コードの記述が簡潔に行える．

input: $A \in \mathbf{R}^{n\times n}$, $x \in \mathbf{R}^n$, $tol \in \mathbf{R}$
output: $\lambda \in \mathbf{R}$, $i \in \mathbf{N}$
$i \leftarrow 0$
$\lambda \leftarrow 0$
while(true) begin
$i \leftarrow i+1$
$y \leftarrow Ax$
$\lambda_{\mathrm{new}} \leftarrow (y^T y)/(y^T x)$
exit if $|\lambda - \lambda_{\mathrm{new}}| \leq tol.$
$x \leftarrow y/||y||_2$
$\lambda \leftarrow \lambda_{\mathrm{new}}$
end

(a) The power method.

```
function [l,i] = powermethod(A, x, tol)
i = 0;
l = 0;
while 1
  i = i + 1;
  y = A * x;
  lnew = (y.'  * y) / (y.'  * x);
  if abs(l - lnew) <= tol, break, end
  x = y / norm(y);
  l = lnew;
end
```

(b) A MATLAB srcipt of the power method.

図 1 MATLAB のコードの例
Fig. 1 An example of MATLAB coding.

MATLAB のプログラム記述に際しては変数の属性値の宣言は不要で，各演算子による計算処理内容はオペランドの型や形状に基づき動的に決定される．実際，図 1 (b) の関数 `powermethod()` を呼び出す際に引数の `A` や `x` にスカラ値を与えても破綻なく計算される．

MATLAB 実行環境には疎行列を扱うためのデータ構造も用意されている．MATLAB の疎行列データ構造は，行列の各列の非零要素を圧縮して行列全体を一次元配列により表現する方法で[15]，Compressed Column Storage（CCS）形式[4] と実質的に同等であり，疎行列の保持に要する記憶容量は非零要素の個数に比例する量で抑えられる．

MATLAB における行列計算コードの記述においてはユーザは各行列が疎行列か否かを意識する必要はない．たとえば，図 1 (b) の関数を呼び出す際に仮引数の行列 `A` に対応する実引数が密行列であるか疎行列であるかによらず適切に計算が行われる．

MATLAB の構文については，付録を参照されたい．

2.2 MATLAB プログラム実行の高速化のための要件

MATLAB はプロトタイピング用途で広く利用されてはいるが，大規模数値計算コード開発に直接的に用いることは一般には難しい．その理由は，大規模数値計算コードに対する最もクリティカルな性能指標である処理速度や所要記憶容量，手続き呼び出しの速度などの観点で，C や Fortran などでの記述による場合に大幅に劣ることが多いからである．

MATLAB プログラムの高速実行のためには，動的処理に起因するオーバヘッドを排除する機能，すなわち，各変数の属性値や各演算子の処理内容を静的解析によって決定できることが求められる．また，大規模行列計算プログラム開発での利用のためには，疎行列データ構造の効率的な扱いも不可欠である．MATLAB プログラムの高速処理のための取り組みである FALCON [12] は静的解析によるコード生成機能を持っているが，疎行列計算に未対応であることから，大規模数値計算への適用が難しいといわざるをえない．しかも構文の拡張を避けているため適用範囲だけでなく動的処理を排除できる範囲までが限定されている．

我々は，処理系への入力記述としてユーザ指示行付きの MATLAB プログラムを仮定することにより

MATLAB プログラムの構文規則を維持しつつ静的型付けとした処理系を開発している[20),21)]．本処理系は，限定的なユーザ指示のみを受けて静的解析により実行時のオーバヘッドを排除して高速化を図るという方針で設計されており，いくつかの実測により効果が確認されている．しかしながら，従来の言語仕様は柔軟性に欠けるもので，MATLAB との互換性や適用可能性という観点で，改良の余地が多分にあった．

3. MATLAB ベースの柔軟性の高い静的型付け言語の設計

本章では，我々が新たに改良を加えた設計した MATLAB ベースの静的型付け言語の設計について述べる．

3.1 設計方針

本節では，従来版と区別するため，従来版[20),21)] を旧 CMC，本論文で述べる処理系を新 CMC と呼ぶ（次節以降では，新 CMC を単に CMC と表現する）．

旧 CMC はもともと，MATLAB の言語仕様との互換性を維持しつつ，高速実行のための制約を課した言語であり，次のような特徴を持つ：

- MATLAB の構文規則をベースとし，型宣言などの新規に追加する構文はユーザ指示行の形式（MATLAB 処理系にとってのコメント行形式）である．
- 変数の型は，形状，要素の基本型，および構造の組として扱われる．
  - 形状：スカラ，列ベクトル，行ベクトル，行列，のいずれか．
  - 要素の基本型：文字型，論理型，整数型，実数型，複素数型．
  - 構造：密あるいは疎（疎は行列の場合のみ）．
- 静的型である．型宣言は仮引数などの一部の変数について行えばよい．

これらを踏襲しつつ，より柔軟で MATLAB との互換性の高い言語処理環境の実現を目的とし，新 CMC では次の変更を新たに加えた．

- 変数のサイズ（行列やベクトルの次元数）を実行時に変更可能にした．
- 関数の型宣言を可能にした．
- 1 つの変数名を複数の型で利用可能にした．
- 関数呼び出しの際の引数授受の方式を MATLAB と同様にした．

以下，それぞれについて述べる．

3.1.1 変数のサイズの動的変更について

旧 CMC では，変数の属性について，形状だけでなくサイズも動的に変更することを許していなかった．また，疎行列データ構造を用いるにあたってはユーザは次元数だけでなく非ゼロ要素数がどの程度になるのかも把握しておかなければならなかった．これらの制約は実行速度の高さを追求するためには有効であったが，一方で，言語処理系の適用範囲を狭める原因となっていた．これらの制約を緩和するため，新 CMC では行列サイズの動的変更に対応することにした．各変数のための領域確保は基本的には値の代入の直前に行うことにしている．

行列サイズの動的な変更を可能にすると，プログラム実行時の頻繁な領域境界のチェックや大量のデータコピーが必要となりうるが，不必要なチェックを自動あるいは手動で抑える仕組みを導入すれば，旧 CMC で得ていた高速性を失うことなく柔軟性の高い処理系を実現できるものと期待される．

3.1.2 関数の型宣言について

旧 CMC では関数の型宣言をすることができなかったため，ユーザは，内部で呼び出される関数の戻り値を受ける変数の型を明記する必要があった．また，関数の引数の型チェックを静的に行うこともできなかった．これに対し新 CMC では関数の型宣言に対応した．これにより，関数の実引数の型の静的なチェックや処理系による型変換処理の挿入ができるようになり，また，インタフェースブロック（Fortran 90）やプロトタイプ宣言（C 言語）の自動出力も可能となった．

3.1.3 同一変数名の複数の型での利用について

新 CMC では，互いに依存関係にない同名の変数の参照を別々の変数の参照に置き換える機能（リネーミング）に新たに対応した．この機能はプログラムの実行速度向上の効果は持たないが，動的型の導入を避けつつできるだけ MATLAB との互換性を高めるためには有効であろうと判断し，導入した．

3.1.4 引数の授受の方式について

旧 CMC では，Fortran の流儀に従い，戻り値を格納する領域を呼び手側が用意しておくことを期待するコードを出力するようにしていた．しかしながら MATLAB では関数の戻り値のサイズが静的に決まるとは限らないため，この方法では対応できるプログラミングスタイルが大幅に制限されてしまう．これに対し新 CMC では，MATLAB と同様なプログラミングを可能にすべく，戻り値の格納領域は基本的に呼ばれた側が確保するようにした．

3.2 指示行の構文

CMC は，MATLAB の構文規則をそのまま受け継ぎ，型宣言などについての指示を MATLAB におけるコメント行の形式で与える．MATLAB の構文は行指

annotation → `%cmc` *type_desc* `::` *id_list* `\n`
type_desc → *attr_list* | `id` | *ftype* | *note*
attr_list → *attr* { `,` *attr* }
attr → *itype* | *shape* | *structure*
itype → `character` | `logical` | `integer` | `real` | `complex`
shape → `scalar` | `colvec` | `rowvec` | `full`
structure → `dense` | `ccs` | `crs`
ftype → `[` *id_list* `] -> [` *id_list* `]`
note → `nocheck`
id_list → `id` { `,` `id` }

図 2 CMC の指示行の構文規則の一部
Fig. 2 Syntax of annotations for CMC.

```
%cmc integer,scalar :: n          % 整数のスカラ
%cmc real,colvec :: v1, v2  % 実数の列ベクトル
```

(a) 変数の宣言

```
%cmc real,colvec :: t                    % 型名定義
%cmc t :: rv1, rv2                       % 型名引用
```

(b) 型名の宣言

```
%cmc integer,scalar :: ts                % 型名定義
%cmc real,colvec :: trv                  % 型名定義
%cmc [ts,trv]->[trv] :: f1,f2            % 関数宣言
```

(c) 関数の宣言

```
    ...
%cmc nocheck :: m              % チェック排除指示
   m(i1, i2) = n(i1) + v(i2);          % 代入文
    ...
%cmc nocheck :: r              % チェック排除指示
   r = rorg - A*x;                     % 代入文
    ...
```

(d) 代入文における境界チェックの排除の指示

図 3 CMC における指示行構文の使用例
Fig. 3 Examples of annotations for CMC.

向であるので，指示行も行単位で記述する．

指示行の構文規則は **図 2** に示すとおりである（MATLAB の構文では各行の '%' 以降の文字はコメントと見なされる）．図中，*type_desc* の 4 とおりの意味は次のとおりである：

- 変数の宣言（**図 3** (a)）
  プログラムすなわち関数定義の中で用いる変数のうち，仮引数の型の宣言を行う．
- 型名の参照（図 3 (b)）
  変数宣言の便宜を図ったり，関数宣言に利用したりするために，型名を宣言することができる．
  型名の宣言は一般の変数の宣言と同じ構文である．宣言されている識別子が型変数か否かはプログラムの本体すなわち変数宣言以外の部分で参照されているかどうかで決まり，本体中で参照されていない変数は型名として扱われる．型名に対しては実行時の領域割当てなどは行われない．
- 関数の宣言（図 3 (c)）
  プログラム中で呼び出している関数の型，すなわちその引数および戻り値の型を宣言する．引数および戻り値の型は，型名の列挙により行う．
- 代入文における境界チェック排除の指示（図 3 (d)）
  代入文の処理中には，ベクトルや行列の添字付き要素参照では代入する要素が現在確保されている領域内のものか否かが実行時にチェックされ，必要ならその変数のための記憶領域の拡張（解放，コピー，および再確保）が行われる．添字式のない参照の場合には，すでに左辺変数に対応する領域が確保されているか否か，確保されているならば大きさが十分か否かが動的にチェックされ，必要に応じて解放と確保が行われる．これに対し，図 3 (d) に示す指示を行うと，その直後の代入文における指定した変数代入に関してのそれらのチェックが抑制される☆．

3.3 MATLAB との違い

CMC は MATLAB と互換性を保つべく設計されているが，いくつかの点で異なっている．現在の CMC と MATLAB との相違点は以下のとおりである．

- CMC では変数としてスカラ，ベクトル，および行列を仮定しており，多次元配列やセル配列は扱えない．またサイズが $0 \times n$ や $n \times 0$ の行列も扱えない（将来的にはこれらには対応したいと考えている）．
- CMC では動的な型変更に対応していない．代入による暗黙の宣言によってプログラムの字面上で同名の変数を異なる型として扱うことはできるが，プログラム中の各点における変数参照において参照される値の型は静的に決まる必要がある．
- CMC では MATLAB にない整数型を基本型として用意している．
- MATLAB では変数に「関数ハンドラ」を代入することもでき，ある名前が変数と関数のいずれを表すかは実行時にしか決まらない場合があるが，CMC では，関数呼び出しには型宣言を要求し，

☆ 動的なチェックが必要である場面，たとえば `nocheck` 指示が付加された当該変数定義文が逆依存の終点となっていない場合には，`nocheck` 指示は無視される．

関数呼び出しが行われる点が静的に決定されるようにしている．

- MATLAB では関数境界を越えて参照できる大域変数があるが，CMC にはない．

4. CMC プログラムの解析

入力プログラムに対する構文解析の結果である中間表現は，MATLAB コードの制御構造および式表現をほぼそのまま保った構文木である．この構文木に対して，依存関係解析，変数の別名化，変数の属性解析，変数の共通化，が順に行われる．これらの処理によってプログラム中で参照される各変数の型情報はすべて静的に決定される（ソースコード中で同一変数名が別の型で使用されている場合にはその変数は別名化されて内部的に別々の変数に置き換えられる）．

これらの解析後，無用命令の除去やループ不変式の移動などの古典的な最適化，および，行列計算に関する強さの軽減処理が行われる[20]が，本論文では触れない．

4.1 依存関係解析

構文木に対して依存関係解析が行われ，各文の間のフロー依存，逆依存，出力依存の情報がまとめられる．現在の実装ではデータフロー方程式の反復解法[1]が用いられている．

4.2 変数の別名化

代入文ごとに左辺変数が別名化される．ただし，互いに依存関係のある変数参照が行われる変数どうしは同名に保たれる．具体的には，コード中のある点における変数の参照（使用）が，複数箇所での参照（定義）に対して「定義–使用の連鎖」の関係にある場合（**図 4**）には，それら複数の定義点において定義される変数は別名化されない☆．変数の部分的な定義（行列の要素定義）は，新たな変数の定義とは見なされない．

別名化の処理は「代入による暗黙の型宣言」の機能を導入するためのものである．別名化は，依存関係解析によって得られるデータフローグラフを用いて，次の手順により行われる：

(1) プログラム中の代入文ごとに以下を繰り返す．
  (a) その文で値が代入される変数がすでに別名化されていなければ，新たに別名を用意し，その変数への代入に置き換える．
  (b) その代入文を始点とするフロー依存エッジの終点における当該変数の参照も，別名への参照に置き換える．このとき，その変数参照がすでに別名化されたものであれば，その別名を再度別名化することを示すリンクを変数表に付加する．
(2) 別名化された変数参照が再度別名化されているものについて，すべて最終的に確定した別名への参照に変更する．

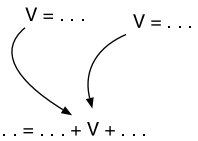

図 4 フロー依存エッジの終点が合流している例
Fig. 4 Flow-dependence edges that share their ends.

4.3 属性解析

別名化後，各変数参照に対して型チェックが行われる．代入文の左辺変数については，代入による暗黙的な型宣言に対応する型が決定される．右辺式の値の型は式中で引用されている変数の型と演算子ごとの規則から合成して決定される．左辺変数の型は基本的に右辺式の値の型とされるが，複数箇所で同一変数が定義されている場合には，それらの中で最も一般的な型☆☆とされる．

4.4 変数の共通化

各変数の型が決定された時点で，別名化前の変数名が共通であってかつ型が等しい変数どうしは同一の変数名に再度変更される．この処理は，変数のための領域確保の量を削減することなどを意図してのものである．

5. コード出力

本章では，4 章で述べた解析によって得られる情報に基づいて入力プログラムを Fortran 90 での記述に変換して出力する手順について述べる．開発中の処理系では，1 つの関数が 1 つの Fortran 90 のサブルーチンとして出力される．

5.1 CMC と Fortran 90 の変数の対応づけ

4 章で述べた処理が済んだ段階では，中間表現における変数名はすべて単一の型（基本型，形状，構造の組）を持っている．Fortran 90 でのコード出力にあたっては，各変数名と Fortran 90 での表現は次のように対応づけられる．

- CMC における行列要素の基本型の対応：
  文字型 →`character(len=1)`,

☆ ここで述べる別名化適用後のコードは，「SSA 形式への変換後に ϕ 関数の引数と戻り値をすべて同一名に修正して ϕ 関数を除去したもの」と見ることもできる．

☆☆ たとえば変数 v へ代入される右辺値が整数であったり実数であったりする場合には変数 v の型は実数型とする，ということ．

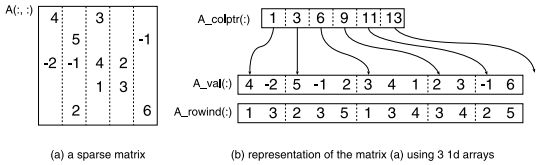

図 5 CCS 形式による疎行列の表現
Fig. 5 Sparse storage scheme of CCS form.

論理型 →logical，整数型 →integer，
実数型 →double precision，
複素数型 →complex double.

- CMC における変数形状と構造の対応：
  スカラ →dimension 属性無し，
  （列/行）ベクトル →dimension(:)，
  密行列 →dimension(:,:).
- 疎行列は，Fortran 90 において複数の変数の組で扱う．たとえば CCS 形式であれば，**図 5** に示すように 3 本の 1 次元配列で 1 つの行列を表す．実数型の変数 A が CCS 形式であれば，double precision の A_val，integer の A_rowind，A_colptr を用いる．複素数型の変数 B が CRS 形式であれば，double complex の B_val，integer の B_rowptr，B_colind を用いる．密行列の場合は行列のサイズを特別に保持する必要がないが，疎行列の場合には別途保持しておく（たとえば行列 A についてスカラ整数 A_d1，A_d2 を用意）．また，非ゼロ要素数を意味するスカラ整数 A_nzmax も用意する．
- スカラ変数以外はすべて pointer 属性とする．これは 3.1.4 項で述べた機能を実現するために必要な処置である．

5.2 動的なサイズ変更に対応するための Fortran 90 ライブラリルーチン

変数の動的なサイズ変更に対応するためには，確保されている領域のサイズの確認や再確保，ポインタの付け替えなどの操作が頻繁に必要になる．本節では，Fortran 90 でのコード出力を補助するために別に用意したライブラリルーチン群をまとめておく．次節以降で示すプログラムの出力例ではこれらの存在を仮定している．

- reallocT1(V,s)
  1 次元配列 V にサイズ s の領域を割り当てる．違うサイズの領域が割り当てられていた場合はいったん解放して再確保する．末尾の文字 T は型を表す（R なら double precision，I なら integer など）．reallocR1() の実装例を**図 6** に示す．
- reallocT2(M,s1,s2)

```
subroutine reallocR1(v, s)
integer :: s
double precision,dimension(:),pointer :: v
if (associated(v)) then
  if (size(v,1).ne.s) then
    deallocate(v)
    allocate(v(s))
  endif
else
  allocate(v(s))
endif
end
```

図 6 reallocR1() の実装例
Fig. 6 Implementation example of reallocR1().

2 次元配列 M にサイズ s1×s2 の領域を割り当てる．違うサイズの領域が割り当てられていた場合はいったん解放して再確保する．末尾の文字 T は型を表す（R なら double precision，I なら integer など）．

- expandT1(V,s)
  1 次元配列 V のサイズが s 以上になるように拡張する．小さいサイズの領域が割り当てられていた場合は，新規にサイズ s の領域を確保してコピーする．末尾の文字 T は型を表す（R なら double precision，I なら integer など）．
- expandT2(M,s1,s2)
  2 次元配列 M のサイズが s1×s2 以上になるように拡張する．小さいサイズの領域が割り当てられていた場合は，新規にサイズ s1×s2 の領域を確保してコピーする．末尾の文字 T は型を表す（R なら double precision，I なら integer など）．
- reallocAndCopyT1(V,V2)
  V2 に 1 次元配列 V の値をコピーする．次と同じ：
  call reallocT1(V2,size(V,1)); V2=V
- reallocAndCopyT2(M,M2)
  M2 に 2 次元配列 M の値をコピーする．次と同じ：
  call reallocT2(M2,size(M,1),size(M,2)); M2=M
- expandByValueT1(V,V1)
  V は 1 次元配列．1 次元配列 V1 の要素の中の最大値 s を得たうえで expandT1(V,s) を実行する．
- expandByValueT2(M,V1,V2)
  M は 2 次元配列．1 次元配列 V1 の要素の中の最大値 s1 および 1 次元配列 V2 の要素の中の最大値 s2 を得たうえで expandT2(M,s1,s2) を実行する．
- linearize(M,V)
  2 次元配列 M を，行優先アクセスで 1 次元配列 V

```
lhs = id_r ( expr_1 , ... , expr_n )
⟹ T_1 = expr_1 ; ... ; T_n = expr_n ;
     lhs = id_r ( T_1 , ... , T_n )

lhs = expr_1 : expr_2
⟹ T_1 = expr_1 ; T_2 = expr_2 ; lhs = T_1 : T_2

id_l ( expr_1 , ... , expr_n ) = rhs
⟹ T_1 = expr_1 ; ... ; T_n = expr_n ;
     id_l ( T_1 , ... , T_n ) = rhs

[ var_1 , ... , var_n ] = rhs
⟹ T_1 = var_1 ; ... ; T_n = var_n ;
     [ T_1 , ... , T_n ] = rhs

id_l ( id_1 , ... , id_n ) = id_r ( id_1 , ... , id_n )
⟹ T_t = id_r ( id_1 , ... , id_n ) ;
     id_l ( id_1 , ... , id_n ) = T_t
```

図 7 CMC における代入文の書き換え（単項および二項演算は省略）
Fig. 7 Rewriting rules for assignment statements in CMC (unary and binary operations omitted).

```
lhs = [ expr_list_1 ; expr_list_2 ; ... ; expr_list_n ]
⟹  lhs = mat_cons([ expr_list_1 ] ,
                  [ expr_list_2 ; ... ; expr_list_n ])

lhs = [ expr_1 , expr_2 , ... , expr_n ]
⟹  lhs = row_cons( expr_1 ,
                   [ expr_2 , ... , expr_n ])

lhs = [ expr ] ⟹ lhs = expr
```

図 8 CMC における行列構成構文の書き換え
Fig. 8 Rewriting rules for matrix constructors in CMC.

に詰め直す．

5.3 コード変換の基本手順

本節では，入力プログラム（CMC プログラムと呼ぶ）を Fortran 90 へ変換する手順を具体的に示す．

5.3.1 式の変形と出力

CMC プログラムにおける代入文は，**図 7** および **図 8** に示す規則を用いて適宜中間的な作業用変数を導入することにより，**図 9** 中のいずれかの形の代入文の系列に変形できる（図 8 は，2 引数関数 `mat_cons` および `row_cons` を導入するもので，要素の列挙による行列構成構文を排除する規則である）．以下では，図 9 の各代入文に対応した，CMC プログラムから Fortran 記述への変換規則について述べる．

まず図 9 の各代入文の右辺は関数呼び出しかそれ以外かに分類して変換する．

関数呼び出しの場合

図 9 の中では，(10) と (11) は関数呼び出しの式であり，(2)，(3)，(9) も関数呼び出しでありうる．関数呼び出しである場合には，**図 10** のように `call` 文が出力される（戻り値のための領域確保は呼び出されたサブルーチン側で行われることが仮定される）．引数と戻り値に同名の変数が列挙されている場合は（MATLAB ではこれが可能），別の変数を媒介にさせる．戻り値を受けるポインタ属性変数は，状況に応じて，`call` 文に先んじて領域を割り付けておくかあるいは `nullify` しておく（5.3.2 項参照）．

なお (9) については，`T=id_r; id_l(id_list)=T` のように中間変数を媒介にした関数呼び出し文に変換される．また，疎行列が引数に現れる場合は，対応する変数の組が列挙される（**図 11**）．

単項，二項演算などによる行列計算についても，それぞれを処理するライブラリルーチンを用意しておけばここで述べたとおりの関数呼び出しへの変換処理で対応できる．疎行列が絡む場合の計算手順については 5.4 節で述べる．

関数呼び出し以外の場合

行列の添字付き参照，すなわち (3)，(4)，(5)，(6)，(7) の右辺，および，(8)，(9) の左辺については，添字式の数は 1 個あるいは 2 個に限定される．これに対し，

```
id_l=num                          (1)
id_l=id_r                         (2)
id_l=id_r(id_list)                (3)
id_l=id_r(:)                      (4)
id_l=id_r(id_1,:)                 (5)
id_l=id_r(:,id_2)                 (6)
id_l=id_r(:,:)                    (7)
id_l(id_list)=num                 (8)
id_l(id_list)=id_r                (9)
[id_list_ret]=id_r                (10)
[id_list_ret]=id_r(id_list_arg)   (11)
```

図 9 図 7 および図 8 の規則に関して既約な式
Fig. 9 Irreducible expressions under rules in Fig. 7 and Fig. 8.

```
[x,n,z]=f(x,n,w)
        ⇓Fortran
call f(x,n,w,tmpx,tmpn,z)
deallocate(x)
x => tmpx
nullify(tmpx)
n = tmpn
```

図 10 関数呼び出し（`x` が密の非スカラ，`n` がスカラの場合）
Fig. 10 Function call statement where `x` is a dense nonscalar and `n` is a scalar.

```
[x]=f(x)
      ⇓Fortran
call f(x_val,x_colptr,x_rowind,
&       x_d1,x_d2,x_nzmax,
&       xtmp_val,xtmp_colptr,xtmp_rowind,
&       xtmp_d1,xtmp_d2,xtmp_nzmax)
 deallocate(x_val)
 deallocate(x_colptr)
 deallocate(x_rowind)
 x_val => xtmp_val
 x_colptr => xtmp_colptr
 x_rowind => xtmp_rowind
 nullify(xtmp_val)
 nullify(xtmp_colptr)
 nullify(xtmp_rowind)
 x_nzmax = xtmp_nzmax
 x_d1 = xtmp_d1
 x_d2 = xtmp_d2
```

図 11 関数呼び出し（x が疎行列の場合）
Fig. 11 Function call statement where x is a sparse matrix.

```
call reallocR2(id_l, size(v_1,1),size(v_2,1))
do itmp2 = 1, size(v_2,1)
  do itmp1 = 1, size(v_1,1)
    id_l(itmp1,itmp2) = id_r(v_1(itmp1),v_2(itmp2))
  enddo
enddo
```

図 12 代入文 id_l=id_r(v1,v2) において v_1, v_2 がベクトルである場合
Fig. 12 An assignment statement id_l=id_r(v1,v2) where v_1 and v_2 are vectors.

- 添字式が 2 個なら，その変数は行列であり，それぞれの添字を 1 次元化して 2 重ループ中でスキャンするコードを出力する．
- 添字式が 1 個なら，添字式の形状に応じたループネストを出力する．id はベクトルか行列であり，ベクトルの場合には 1 次元化して参照するようにする．

例 1 id_l=id_r(v1,v2) において v_1, v_2 がともにベクトルである場合には，**図 12** のように変換される．

例 2 id_l(m)=id_r において id_l がベクトル，m と id_r が行列である場合には，**図 13** のように変換される．m と id_r のサイズの整合性の動的チェックの導入も可能である．

例 3 id_l(M1,M2)=num は，右辺のスカラ値を左辺の各要素に格納する代入文であり，**図 14** のように変換できる．

例 4 r=row_cons(x,y)（すなわち r=[x,y]）において x および y が行列である時は**図 15** のようになる．変数の形状に応じてコード記述の詳細は変更する．

```
call linearize(m, vtmp)
call expandByValueR1(id_l, vtmp)
do itmp2 = 1, size(m,2)
  do itmp1 = 1, size(m,1)
    id_l(m(itmp1, itmp2)) = id_r(itmp1, itmp2)
  enddo
enddo
```

図 13 代入文 id_l(m)=id_r において id_l がベクトル，m と id_r が行列である場合
Fig. 13 Assignment statement id_l(m)=id_r where id_l is a vector and m and id_r are matrices.

```
call linearize(M_1, v_1)
call linearize(M_2, v_2)
call expandByValueR2(id_l, v_1, v_2)
do itmp2 = 1, size(v_2,1)
  do itmp1 = 1, size(v_1,1)
    id_l(v_1(itmp1),v_2(itmp2)) = num
  enddo
enddo
```

図 14 代入文 id_l(M1,M2)=num で M_1, M_2 が行列の場合
Fig. 14 Assignment statement id_l(M1,M2)=num where M_1 and M_2 are matrices.

```
reallocR2(r,size(x,1,size(x,2)+size(y,2)))
r(1:size(x,1), 1:size(x,2)) = x
r(1:size(x,1),size(x,2)+1:size(x,2)+size(y,2))=y
```

図 15 r=[x,y] において x, y が行列の場合
Fig. 15 r=[x,y] where x and y are matrices.

```
function [r,x]=f(a,x)
      ⇓Fortran
subroutine f(aorg,xorg,r,x)
... ! 変数宣言など
nullify(a)
call reallocAndCopyR2(aorg, a)
call reallocAndCopyR2(xorg, x)
...
if (associated(a)) deallocate(a)
end
```

図 16 サブルーチン文
Fig. 16 Subroutine statiment.

5.3.2 サブルーチン文

Fortran 90 によるサブルーチン文の出力の様子を**図 16** に示す．図に示すように引数は局所変数に複写してから使用する（図 16 では，引数は aorg と xorg で受け，値は a と x とに複写している．aorg と xorg はサブルーチン内では値を変更しない）．変数 a は呼び出し元からは見えない局所変数で，先頭で領域確保し，また戻る前に解放する（reallocAndCopyR2 の中で associated かどうか調べられるので先頭で初期化（nullify）が必要）．一方，変数 x は戻り値であり，

呼び出し元によって associated とされている可能性があるので，nullify せずに reallocAndCopyR2 を呼び出す．

なお，仮引数の値が関数内で引用されるだけであって定義されることがない場合は，コピーの作成は省く．

5.3.3 インタフェース構文

変換後のプログラムにおいて呼び出されるユーザ定義手続きがある場合は，対応するインタフェースブロックが出力される（**図 17**）．

5.4 行列計算について

行列計算の変換処理については基本的には従来の CMC[20),21)] の場合と同様であるが，疎行列が絡む場合については若干変更を加えている．

行列計算において計算結果が疎行列となる場合には，結果の非ゼロ要素の個数を計算を行う前に正確に決めることはできないので，計算結果を格納する領域を必ずしも計算に先んじて確保することができない．従来の CMC では各疎行列が持つ非ゼロ要素数をユーザが指示行で与えなくてはならず，しかも領域境界チェックを省いていたためにランタイムエラーの可能性が避けられなかった．これに対し現在の実装では，行列サイズの動的拡張に対応するにあたり，計算処理中に領域チェックを行うようにしている．

たとえば計算結果が CCS 形式であれば列ごとに計算を進めるので（**図 18** に CCS 形式の疎行列どうしの和を求める様子を示す）列単位での確保量のチェックが行われる．CRS 形式であれば行単位の処理となる．

```
%cmc real,scalar :: t % 型名宣言
%cmc [t]->[t] :: func % 関数宣言
        ⇓Fortran
interface
   subroutine func(arg1, ret1)
       real,scalar :: arg1
       real,scalar :: ret1
   end
end interface
```

図 17 インタフェースブロック
Fig. 17 Interface block.

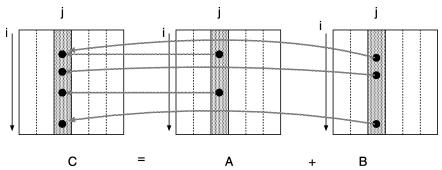

図 18 CCS 形式による行列の和の計算の様子
Fig. 18 Sparse matrix addition in CCS from.

6. 性能評価

開発中の処理系の実行性能の予備的評価のために，数値計算プログラムを題材として実測を行った．実測に用いたプログラムは，ハウスホルダ変換による QR 分解アルゴリズム[16)] を MATLAB で記述したものである．プログラムの全体を**図 19** に示す．このプログラムは，ループの反復中において作業領域の大きさが変化したり（図 19 (a)）関数の戻り値のサイズが引数の値によって決定されたりする（図 19 (b)）などの特徴があり，これまでに開発した処理系[20)] では扱うことができなかったものである．このプログラムに対し，CMC で Fortran 90 に変換した場合と MATLAB で実行した場合の速度を比較する．

実測に用いるプログラムは疎行列データ構造をいっさい扱わずに済むものである．疎行列計算に関する性能はこれまでの報告結果[20),21)] がほぼそのままあてはまるのでここでは省略する．

なお（新仕様の）CMC による変換機能はいまだ不完全な部分があるので，部分的に手作業でコードを変

```
function [A] = houseQR(A)
%cmc real,full :: A
%cmc real,colvec :: tv
%cmc real,scalar :: ts
%cmc [tv]->[tv,ts] :: house
m = size(A,1);
n = size(A,2);
for j = 1:n
  [v, beta] = house(A(j:m,j));
  A(j:m,j:n)=A(j:m,j:n)-(beta*v)*(v'*A(j:m,j:n));
  if j < m
    A(j+1:m, j) = v(2:m-j+1);
  end
end
```

(a) Householder QR：内部で (b) の house() を呼ぶ

```
function [v, beta] = house(x)
%cmc real,colvec :: x
n = length(x);
if (n==1)
  v = 1;
  beta = x;
else
  x = x / norm(x);
  sigma = x(2:n)' * x(2:n);
  v = [1; x(2:n)];
  if (sigma <= 1e-10)
    beta = 0;
  else
    mu = sqrt(x(1)^ 2 + sigma);
    if (x(1) <= 0)
      v(1) = x(1) - mu;
    else
      v(1) = -sigma/(x(1) + mu);
    end
    beta = 2*v(1)^ 2/(sigma + v(1)^ 2);
    v = v/v(1);
  end
end
```

(b) Householder Vector を求める

図 19 ハウスホルダ変換による QR 分解アルゴリズム
Fig. 19 QR factorization by Householder transformation.

換した．ただし本論文で述べている基本手順に従って単純に変換した（領域確保量抑制の `nocheck` 指示などもしていない）．

6.1 実測環境

実測に用いた計算機やソフトウェア環境は次のとおりである．

- 富士通 HPC2500☆（SPARC64V 2.08 GHz，主記憶 512 GB，Solaris 8；SMP だが 1 CPU のみ使用）：以下 SPARC と呼ぶ．
  - MATLAB：7.0.1.24704 R14 （SP1）
  - Fortran：`frt`（Version 5.6），コンパイルオプション：`-Kfast_GP=3 -X9`
- Apple PowerBook G4（PowerPC G4 1.5 GHz，主記憶 1.5 GB，Mac OS X 10.4.2）： 以下 PowerPC と呼ぶ．
  - MATLAB：7.0.1.24704 R14 （SP1）
  - Fortran：`g95` （gcc version 4.0.0），コンパイルオプション：`-O3`
- 富士通 FMV（Pentium-M 1.5 GHz，主記憶 768 MB，Linux 2.6.9）：以下 Pentium と呼ぶ．
  - MATLAB：7.0.1.24704 R14 （SP1）
  - Fortran：`g95` （gcc version 4.0.1），コンパイルオプション`-O3`

計時は，MATLAB および MCC では組み込み関数 `tic` および `toc` を利用した．Fortran 90 コードの実行においては，標準ライブラリ関数 `gettimeofday()` をリンクして用いた．

数値実験で用いた密行列は次のとおりである：

$F = (f_{i,j}) \in \mathbf{R}^{n\times n}$，$v \in \mathbf{R}^n$，

$$f_{i,j} = \begin{cases} 1 & (i = j) \\ -3^{-|i-j|} & (i \neq j) \end{cases}$$

6.2 実測結果

表 1 は，4 種類の行列サイズの問題に対し，SPARC，PowerPC，Pentium のそれぞれについて，MATLAB インタプリタでの実行に要した時間と Fortran 90 に変換して実行した時間をまとめたものである．表 1 より，CMC で記述して変換したプログラムはほとんどの場合に MATLAB よりも高速であることが分かる．

高速化の度合いは PowerPC および Pentium ではおよそ 3 割程度で，しかも問題サイズが大きくなるにつれて比率が小さくなっている．問題サイズの変化にともなう高速化率の変化はアプリケーション実行における計算コストとメモリ管理コストの割合などに強く依存するので，表 1 の結果だけを見て説明することは難しい．しかしながら，MATLAB と同等な行列サイズの動的変更処理を Fortran 90 によるシンプルな実装により実現できることが確認できた．

☆ 京都大学学術情報メディアセンター設置．

表 1 ハウスホルダ QR 分解の実行に要した時間（単位は秒）
Table 1 Execution times of Householder QR decomposition in seconds.

	SPARC		PowerPC		Pentium	
問題サイズ	MATLAB	CMC	MATLAB	CMC	MATLAB	CMC
300×300	0.207	0.276	2.34	1.60	1.21	0.709
400×400	2.48	1.94	5.61	4.04	2.00	1.40
500×500	12.7	4.95	10.9	8.02	4.47	3.42
600×600	31.6	9.56	18.9	14.4	8.89	7.23

下線は，各計算機について，各問題サイズに対して高速に処理が行われた方を示す．

7. 関連研究

本章では，行列計算プログラム開発に関する研究を CMC と比較しつつまとめる．

7.1 高水準言語記述からのコード生成

MATLAB コード実行の高速化に関しては，1 章でも述べたとおり，変数の型の推定を行って Fortran コードへの変換を行うもの[12]，並列処理機能を持つライブラリの導入により並列処理を可能にしようとするもの[28),29)]，ソースレベル変換による最適化の検討[27)] などの研究がある．また MATLAB の対話的実行環境の機能を保持したままで部分的に just-in-time コンパイルを行う方法も研究されている[2)]．注釈付き MATLAB プログラムから組み込み環境向けコードやハードウェア記述言語プログラムを生成するアプローチもある[3)]．これらの多くはプログラム中の変数の属性値の推定を行う機能を有する．しかしながらいずれも疎行列計算についての言及はない．

変換後のプログラムの処理速度を重視して静的型言語として設計されている MaTX[25)] などの行列言語もあるが，疎行列データ構造に対応したものは見られない．また基本的に使用するすべての変数の属性宣言をユーザが記述する必要があり，CMC と比較するとプログラム記述は繁雑になる．

7.2 疎行列計算のためのプログラミング言語

Fortran 90 における配列計算プリミティブを疎行列データ構造に対応させる研究[7),8)] もある．しかしながら Fortran 90 では行列計算の数式記述に直接的にコード表現を対応させることができないので，プログラムを記述しやすいとは必ずしもいえない．たとえばスカラどうしの積と行列どうしの積が同じ演算子 ‘*’ で表現できない．

疎行列計算コード開発を支援するシステムとしては，関数型言語記述から疎行列コードを生成する試みもある[14]．また，Fortran 77 で記述された密行列用コードをもとにして疎行列コードを生成する研究もある[5),6)]．これらはいずれも疎行列コード記述の繁雑さを低減させることを主眼としているが，それらのソースとなる関数型言語による記述や Fortran 77 による密行列コード記述が可読性や保守性に優れているかどうかは意見が分かれるところであろう．自動変換系の実装の容易さも問題で，たとえば Fortran 77 での密行列コード（すなわち多重ループ中での要素計算の反復）から行列の転置操作を自動検出するのも容易ではないだろう．我々は，少なくとも，知名度の高い MATLAB 記述に基づく方法の実現は有意義であると考える．

7.3 ライブラリの利用とその最適化

高性能な疎行列コード開発支援のために，疎行列演算ライブラリの仕様共通化の提案も行われている[13)]．一般に多数の引数の指定を必要とするライブラリルーチンを用いるコードを一般ユーザが直接記述することは必ずしも容易ではないし，結果として得られるコードの可読性や保守性も高いとはいい難い．しかしながら，標準化されたライブラリを効果的に使用するコードを MATLAB などの高水準言語記述から生成できれば，保守性と性能可搬性が両立できる．CMC はこの方向でも貢献できる．

なお，汎用的に利用可能なライブラリルーチンのチューニング技法としては，ライブラリのインストール時に環境に合わせた自動チューニングが可能な ATLAS-tuned BLAS[31)] が有名である．最近では，実行時に自動チューニングする疎行列対応ライブラリも開発されている[30)]．

ライブラリによる行列計算処理とその制御ルーチンを分離したクライアント・サーバ型のシステムの例に SILC がある[18)]．SILC は，制御プログラムと計算処理の実体を完全に分離し，またライブラリ利用のためのインタフェースを簡潔にまとめることによってあらゆるプログラミング言語を用いてほぼ同一の手続きでプログラムを構成することを可能にするなど，ユーザの負担削減を意識したシステムである．しかしながら，サーバに任せる計算処理記述が行列計算式の並びに限定されているために，現時点では反復処理に基づくアルゴリズム記述が効率的に行えるとはいえない[19)]．サーバ側での計算処理の最適化の機会が少ない点などについても改善の余地があるといえる．CMC はこのようなシステム向けのスクリプト言語としても利用可能であろう．

7.4 統合的アプローチ： Telescoping Languages[22)]

大規模数値計算プログラムの開発環境に関する研究は数多く行われている．近年では特に，抽象度の高いデータ型や演算子を備えた言語によるプログラムを入力として受け，高度に最適化されたオブジェクトコードを出力するというシステムのスタイルが注目されている．プログラミングに要求されるユーザの負担の軽さおよび実行コードの性能の高さの両者がともに重要視され，トレードオフの関係としてとらえることは必ずしも十分とはいえない状況となっている．そこでは，ソースコードだけではなくライブラリ（関数呼び出し関係の下層部分）をいかに効果的に利用するかがポイントとなる．

代表的なアプローチとして，Rice 大学のグループによって研究されている Telescoping Languages プロジェクト[9),11),22),26)] があげられる．この研究は，高度に最適化されたライブラリを簡潔なスクリプトで連結するプログラミング環境の実現を目的とするもので，キーとなるのはライブラリ開発者による詳細な注釈（文脈）記述に基づくライブラリの最適化（特殊化）機能，および，ソースコード中のライブラリルーチン呼び出し箇所における文脈抽出機能である．特殊化（specialization）とは，1 つのライブラリルーチン記述から生成可能な variant（たとえば引数の型を固定化した複数のバージョンなど）を自動的に抽出してそれぞれ個別のルーチンとして最適化して用意することを指す．特殊化の際には個々のルーチンが呼び出される文脈から適切なルーチンを選択してリンクするための “variants database” も作成（更新）する．

ライブラリ開発者の知見をコンパイラに対して注釈の形で与えて最適化のために利用するという考え方は Broadway コンパイラシステム[17)] でも提案されているが，Telescoping Languages ではオブジェクトコードの生成にライブラリのソースを必要としないので，短いスクリプトのコンパイルに長大なコンパイル時間を要するようなことがない．また MATLAB などの高水準言語でライブラリからアプリケーションまですべてを記述可能な枠組みを用意している．たとえば，動的型言語によるライブラリ（およびスクリプト）記述に対する変数の属性値推定機能を備えている．この属性値推定機能は FALCON プロジェクト[12)] のアプローチとは異なりデータフロー解析に依っていない．また 1 つの記述から生成可能なライブラリの variant を導出する機能を持っている[10),11),26)]．FALCON の開発グループによる just-in-time コンパイル処理系

MaJIC[2] でもライブラリの特殊化を投機的に試みているが，Telescoping Languages では最適化の効果向上のためにライブラリ開発者の注釈を用いようとしている．

Telescoping Languages のグループは MATLAB プログラムに対して行列サイズを推定する手法として slice-hoisting を提案している[9),10)]．この方法では，処理系はプログラム中で使用される行列のサイズ決定に関係する処理を計算処理部分から分離し，計算処理に先んじて（可能ならば静的に）所要領域を確保するようなコードを出力する．この方法は他のコード最適化手法に対して独立性が高く，CMC 処理系でも直接的に利用可能である．

Telescoping Languages の枠組み自体は非常に抽象的であり[22)]，プログラム開発環境としてある意味で理想的で，原理的にはあらゆるコンパイラ技術を取り込むことができるものと思われる．たとえば，現時点での CMC に対する取り組みは Telescoping Languages の枠組みにおけるライブラリ開発言語設計に応用することができる．

Telesoping Languages に関する懸念としては，ライブラリ開発の際に「正しく動くコード」しか解析対象として想定していないためにプロトタイピングに使用しにくい可能性があるという点があげられる．また，ライブラリの特殊化時に不必要な variant の生成を排除する機能の詳細な検討も必要であろう．たとえば MATLAB プログラムは，仕様上，静的にすべての変数の型を唯一に決定することが必ずしもできないので，多段のライブラリ階層で構成されるアプリケーションのコンパイル時には膨大な数のライブラリルーチンの variant が生成されてしまう可能性が否めない．さらに，1 つのアプリケーションをコンパイルする際には基本的に関数呼び出し階層の最下層からコンパイラで処理する必要があり，また現実的には（不必要な variant の増大を排除するために）すべてのライブラリルーチンに注釈を付ける必要がある．結果として，小規模な関数を多数組み合わせてアプリケーションを構築したい場合など（これはごく一般的に見られる状況である）にはユーザの負担は少なくはない．また，Telescoping Languages の枠組みではライブラリ記述者がいかに効果的な注釈を与えられるか（どの程度与える必要があるか）がその性能を大きく左右すると考えられるが，現時点ではライブラリの variants database の詳細構造や注釈記述の言語仕様などは具体的に述べられていない[9)]．性能評価についても DSP アプリケーションなどの特定分野における応用以外への言及も少ない．その他，ライブラリの粒度の制御やインライン展開への対応など，実装方式が具体化されていない点が多い．

8. おわりに

本論文では，行列計算プログラムの開発を容易にすることを目的として開発中の MATLAB ベース静的型付け言語処理系 CMC について述べた．CMC の入力言語記述は，静的解析によって C 言語や Fortran 90 などの汎用コンパイル言語記述に変換することができる．本論文では，従来の CMC の仕様に対して MATLAB との互換性をより向上させるための改良点と，Fortran 90 への変換系の実装手順について，具体的に述べた．仕様の改良によって新たに加えられた機能の性能評価のためにハウスホルダ QR 分解プログラムを題材として複数の計算機上で実測したところ，MATLAB と比較して 3 割程度高速に実行され，CMC の有効性が確認できた．

現時点での CMC の仕様上の問題点としては，(1) 複数の関数を階層的に呼び出すプログラム全体の解析機能や最適化機能がないこと，(2) 同一関数に対する variant 生成および最適化機能がないこと，および (3) 並列処理環境についての検討がないことがあげられる．(1) は，比較的小さな関数を多数個組み合わせて 1 つのプログラムを記述する場合にかかるユーザの負荷軽減のために改善を検討すべき点の 1 つである．現状の CMC は関数ごとにユーザが指示行を与える必要があるが，ユーザによって関数階層が記述される場合などには各関数が呼ばれる文脈は唯一に特定できることが多いと考えられ，関数呼び出し階層における最上位のルーチンに対して若干のユーザ指示があればすべてのルーチンを変換できるであろう．このような，トップダウン的な関数間解析によるエラーチェックやコード生成機能の開発が今後の課題の 1 つである．(2) は Telescoping Languages の枠組み[11)] が自動的に行っているとされる処理の再検討を意識したものである．MATLAB では 1 つの関数が呼ばれる文脈によっては大きく異なる挙動となる場合がある．現状の CMC ではこのような関数記述には対応していないが，いくつかのパターンを想定した変数属性値指定を行えるようにするなど，何らかの限定を加えた形での対応が望ましいと考えられる．

実装上の課題としては，BLAS をはじめとする標準ライブラリの組み込みや，多次元配列，構造体などへの対応，C 言語による出力機能の実装などがあげられる．また，より規模の大きいアプリケーションを用い

た性能評価のほか，不必要な領域境界チェックを自動的に省く機能の検討（slice-hoisting [9),10)] の評価）なども行いたい．

謝辞 本研究の一部は広島市立大学特定研究費（一般研究費，課題番号 4111）および科学研究費補助金若手研究（B)（課題番号 17700037）の助成による．

参 考 文 献

1) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers — Principles, Techniques and Tools*, Addison Wesley (1986).
2) Almasi, G. and Padua, D.: MaJIC: Compiling MATLAB for Speed and Responsiveness, *Proc. PLDI'02*, pp.294–303 (2002).
3) Banerjee. P., et al.: A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable Computing Systems, *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines* (2000).
4) Barrett, R., et al.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM (1994).
5) Bik, A.J.C. and Wijshoff, H.A.G.: Compilation Techniques for Sparse Matrix Computations, *Proc. ICS'93*, pp.416–424 (1993).
6) Bik, A.J.C., et al.: The Automatic Generation of Sparse Primitives, *ACM Trans. Math. Softw.*, Vol.24, No.2, pp.190–225 (1998).
7) Chang, R.-G., Chuang, T.-R. and Lee, J.K.: Efficient Support of Parallel Sparse Computation for Array Intrinsic Functions of Fortran 90, *Proc. ICS'98*, pp.45–52 (1998).
8) Chang, R.-G., Chuang, T.-R. and Lee, J.K.: Compiler Optimizations for Parallel Sparse Programs with Array Intrinsics of Fortran 90, *Proc. Intl. Conf. Parallel Processing*, pp.103–110 (1999).
9) Chauhan, A.: Automatic Type-Driven Library Generation for Telescoping Languages, Ph.D. Thesis, Tech. Rep. TR03-428, Rice University (2003).
10) Chauhan, A. and Kennedy, K.: Slice-hoisting for Array-size Inference in MATLAB, *Proc. 16th International Workshop on Languages and Compilers for Computing* (*LCPC*), pp.495–508, Springer-Verlag (2003).
11) Chauhan, A., McCosh, C., Kennedy, K. and Hanson, R.: Automatic Type-Driven Library Generation for Telescoping Languages, *Proc. SC'03* (2003).
12) De Rose, L. and Padua, D.: Techniques for the translation of MATLAB programs into Fortran 90, *ACM Trans. Prog. Lang. Syst.*, Vol.21, No.2, pp.286–323 (1999).
13) Duff, I.S., et al.: Level 3 Basic Linear Algebra Subprograms for Sparse Matrices: A User-Level Interface, *ACM Trans. Math. Softw.*, Vol.23, No.3, pp.379–401 (1997).
14) Fitzpatrick, S., Clint, M. and Kilpatrick, P.: The Automated Derivation of Sparse Implementations of Numerical Algorithms through Program Transformation, Tech. Rep. 1995/Apr-SF.MC.PLK, Dept. Comput. Sci., The Queen's University of Belfast (1995).
15) Gilbert, J.R., Moler, C. and Schreiber, R.: Sparse Matrices in MATLAB: Design and Implementation, *SIAM J. Matrix Anal. Appl.*, Vol.13, No.1, pp.333–356 (1992).
16) Golub, G.H. and van Loan, C.F.: *Matrix Computations*, 3rd edition, Johns Hopkins University Press (1996).
17) Guyer, S.Z. and Lin, C.: An Annotation Language for Optimizing Software Libraries, *Proc. ACM SIGPLAN/USENIX Workshop on Domain Specific Languages* (1999).
18) 長谷川秀彦，須田礼仁，額田 彰，梶山民人，中島研吾，高橋大介，小武守恒，藤井昭宏，西田 晃： 計算環境に依存しない行列計算ライブラリインタフェース SILC，情報処理学会研究報告，2004-HPC-100, pp.37–42 (2004).
19) 梶山民人，額田 彰，須田礼仁，長谷川秀彦，西田 晃： 共有メモリ並列環境における SILC の実現と利用，第 34 回数値解析シンポジウム講演予稿集，pp.49–52 (2005).
20) 川端英之，鈴木 睦： 疎行列に対応した行列言語コンパイラ CMC の開発，情報処理学会論文誌：コンピューティングシステム，Vol.45, No.SIG11(ACS7), pp.378–392 (2004).
21) Kawabata, H., Suzuki, M. and Kitamura, T.: A MATLAB-Based Code Generator for Sparse Matrix Computations, *Proc. APLAS2004*, LNCS, Vol.3302, pp.280–295 (2004).
22) Kennedy, K., et al.: Telescoping Languages: A Strategy for Automatic Generation of Scientific Problem-Solving Systems from Annotated Libraries, *Journal of Parallel and Distributed Computing*, Vol.61, pp.1803–1826 (2001).
23) MathWorks Inc. homepage. http://www.mathworks.com/
24) The MathWorks, Inc.: *MATLAB Compiler Version 3 User's Guide* (2002).
25) http://www.matx.org/
26) McCosh, C.: Type-Based Specialization in a Telescoping Compiler for MATLAB, Master's Thesis, Tech. Rep. TR03-412, Rice University (2003).
27) Menon, V. and Pingali, K.: A Case for Source-

Level Transformations in MATLAB, *Proc. DSL'99*, pp.53–65 (1999).
28) Ramaswamy, S., et al.: Compiling MATLAB Programsto ScaLAPACK: Exploiting Task and Data Parallelism, *Proc. IPPS'96*, pp.613–619 (1996).
29) Quinn, M.J., Malishevsky, A. and Seelam, N.: Otter: Bridging the Gap between MATLAB and ScaLAPACK, *Proc. 8th IEEE Intl. Symp. High Performance Distributed Computing* (1998).
30) Vuduc, R., Demmel, J.W. and Yelick, K.A.: OSKI: A Library of Automatically Tuned Sparse Matrix Kernels, *Journal of Physics: Conference Series*, Vol.16, pp.521–530 (2005).
31) Whaley, R.C. and Dongarra, J.J.: Automatically Tuned Linear Algebra Software, *Proc. SC: High Performance Networking and Computing Conference* (1998).

付　　録

A.1 MATLAB プログラミング

MATLAB のプログラミング言語としての特徴をまとめておく．

A.1.1 変数と演算子について

- 変数は行列である．スカラ変数はサイズが 1×1 の行列であり，列ベクトル（サイズが $1 \times n$ の行列）と行ベクトル（サイズが $n \times 1$ の行列）は区別される☆．
- 変数の持つ属性値には，サイズ以外に，個々の要素の基本型（文字型/論理型/実数型/複素数型），およびデータ構造（密行列/疎行列）がある．データ構造はプログラミングにおいてはユーザが必ずしも意識する必要はないが，ユーザが制御することができる．疎行列構造は CCS 形式である．
- 変数の型付けは動的に行われる．変数の型（属性）宣言はない．代入文が実行された時点で，左辺変数が宣言されその領域が確保される．
- + や * などの組み込みの演算子は行列演算である．

A.1.2 制御構造について

- if 文や switch 文による条件判断，while 文や for 文による処理の反復が記述できる．
- いわゆる goto 文はなく，制御構造はブロックが明示される．ただし，字面上のブロックは変数のスコープを意味しない（次項を参照）．

☆ MATLAB バージョン 6 以降では，多次元配列や構造体，セル配列なども使用できる．またサイズが $0 \times n$ あるいは $n \times 0$ の行列も扱える．

A.1.3 プログラムとその実行について

- MATLAB のプログラムは，関数の集合である．1 つの関数は 0 個以上の仮引数と 0 個以上の戻り値を持つ．任意個数の仮引数および戻り値の関数を作ることもできる．なお関数は入れ子にはできない．
- 関数呼び出しの際の引数の授受は値呼びで行われる．
- プログラムの実行とは，各時点で参照できるワークスペース（変数と関数の情報の集合）に対する操作の系列である．
- 代入文では，右辺式中で参照される変数および関数の値をワークスペースを参照することによって得て，それをもとに右辺式を計算し，値を左辺変数に代入する．このとき，
  - 左辺変数が添字付き参照でない場合，ワークスペース中でその変数が未定義であれば新規に登録し，すでに登録されていれば上書きする．
  - 左辺が行列の要素参照の場合は，ワークスペース中でその変数が未定義であれば新規に登録し，すでに登録されていれば，そのサイズに応じて単なる値の更新かあるいは領域拡張を行う．
- 関数呼び出しの際には，実引数の情報がコピーされた局所的なワークスペースが作られ，対応する仮引数名で参照される．呼び出された関数内部では呼び出し元のワークスペースは参照できない．呼び出された関数から復帰する際には，戻り値の変数に対応する情報が呼び出し元のワークスペースにコピーされる．
- 変数をグローバル宣言しておくことにより，関数の呼び出し関係を越えた大域的な変数参照もできる．

A.1.4 MATLAB の特徴的な構文

MATLAB における代入文の構文は**図 20** に示す．MATLAB における変数は行列であり，行列要素の参照や領域拡張のための構文が用意されている．以下に例をあげつつ特徴をまとめる．

- `A(:,n)`
  行列 `A` の第 n 列が列ベクトルとしてまとめて参照される．':' は単独では *expr* ではなく，行列添字としてのみ出現できる．
- `A(:)`
  行列 `A` の全要素が 1 本の列ベクトルとして（列優先アクセスで）参照される．

assign_stmt → *var* `=` *expr* | `[` *var_list* `]` `=` *var*
var_list → *var* { `,` *var* }
var → `id` | `id` `(` *expr_or_colon_list* `)`
expr_or_colon_list → *expr_or_colon* { `,` *expr_or_colon* }
expr_or_colon → *expr* | `:`
expr → `num` | *var* | `u_op` *expr* | *expr* `b_op` *expr* | *expr* `:` *expr* | `(` *expr* `)` | `[` *row_list* `]`
row_list → *expr_list* { `;` *expr_list* }
expr_list → *expr* { `,` *expr* }

図 20 CMC における代入文の構文規則の概略☆
Fig. 20 Assignment statements in CMC.☆

- `[` e_1 `,` e_2 `,` ... `,` e_n `]`
  括弧中に列挙されている各式の値を横方向に連結した行列を構成する．‘,’ ではなく空白で式が区切られていてもいい．
- `[` e_1 `;` e_2 `;` ... `;` e_n `]`
  括弧中に列挙されている各式の値を縦方向に連結した行列を構成する．‘;’ と ‘,’ では，‘,’ の方が優先順位が高い．
- e_1 `:` e_2 `:` e_3
  式 e_1，e_2，e_3 の値の先頭の要素の値（すなわち行列の (1,1) 要素)) を s_1，s_2，s_3 とするとき，下のような行ベクトルが構成される．
  $$(s_1, s_1 + s_2, s_1 + 2 \cdot s_2, \ldots, s_1 + n \cdot s_2)$$
  末尾の要素における n は $s_1 + n \cdot s_2 \leq s_3$ を満足する最大の整数である．e_2 を省略して e_1 `:` e_3 と記述した場合は e_1 `:` 1 `:` e_3 の意味となる．
- `[` r_1 `,` r_2 `]` `=` `func(` e_1 `,` e_2 `)`
  関数呼び出しの一般的記法である．右辺の括弧中の式のリストは，個々の値を計算した後にその値が引数として関数に渡される．左辺の括弧中に列挙されているのは戻り値を受ける変数である．関数呼び出しは値渡しで行われるので，実引数リストに列挙された変数の値は関数呼び出しの実行によって影響を受けることはない．実引数リストに列挙された変数を仮引数リストに含めた場合は，その変数の内容は関数呼び出しから戻った時点で変更される．
  なお，戻り値が 1 つの関数呼び出しは `r=func(`e_1`,`e_2`)` と記述することもでき，`r=`e_1`+func(`e_2`)*`e_3 のように式中に含めることもできる．
- MATLAB の for 文は次に示す構造である．

```
for v = expr
    ...
end
```

第 i 回目の反復では制御変数 `v` の値は式 *expr* の値の第 i 列目が入る．すなわち，上の for ループは次のように書き換えることができる．

```
t = expr;
for i = 1 : size(t,2)
    v = t(:,i);
    ...
end
```

(平成 17 年 9 月 22 日受付)
(平成 17 年 12 月 26 日採録)

川端　英之（正会員）
1992 年京都大学工学部情報工学科卒業．1994 年同大学大学院工学研究科修士課程修了．同年より広島市立大学情報科学部助手．博士（工学）．高性能計算，数値処理ソフトウェアに関する研究に従事．ACM，IEEE-CS，SIAM，電子情報通信学会，日本ソフトウェア科学会，日本応用数理学会各会員．

北村　俊明（正会員）
1978 年京都大学工学部情報工学科卒業．1983 年同大学大学院工学研究科博士課程情報工学専攻研究指導認定退学．同年富士通（株）入社．汎用コンピュータ，スーパコンピュータ VPP シリーズの VLIW 型 CPU，M アーキテクチャ・命令エミュレーション，米国 HAL 社において SPARC プロセッサ等の研究開発に従事．博士（工学）．2000 年京都大学総合情報メディアセンター助教授．2002 年広島市立大学情報科学部教授．計算機アーキテクチャに興味を持つ．電子情報通信学会，IEEE，ACM 各会員．

☆ 実際は，2 つ以上 ‘;’ や ‘,’ が並んでいるときには 2 つ目以降を無視したり，列挙による行列構成構文中では ‘,’ と空白が同義と見なしたりするなど，細かい構文規則がある．