疎行列に対応した行列言語コンパイラ CMC の開発

川端英之[†] 鈴木睦[†,☆]

数値計算プログラムの記述を容易にするために MATLAB をはじめとする行列言語が開発され，ラピッドプロトタイピング用途等に広く利用されている．行列言語プログラムはインタプリタで実行される場合が多いが，処理の高速化を目的とし，行列言語コンパイラを用いて静的解析により余分な動的処理を排除して Fortran などのコンパイル言語記述に変換する手法が検討されつつある．しかしながら，大規模数値計算コード記述を想定して疎行列の扱いを考慮した取り組みはなかった．これに対し我々は，密行列と疎行列を区別なく扱うことを可能にする行列言語コンパイラ CMC を開発した．CMC は MATLAB に基づくコードを Fortran 90 に変換することができ，対角，三角などの行列形状情報の検出機能とそれに基づく最適化機能も備える．試作した処理系を SOR 法や CG 法の MATLAB コードに適用して実測したところ，係数が疎行列の大規模コードにおいて，SOR 法で 7 倍以上，CG 法でも 3 倍以上，MATLAB 実行環境よりも高速に実行でき，行列言語をベースとした大規模数値計算コード開発の可能性が確認できた．

CMC: A Compiler for Sparse Matrix Computations

Hideyuki Kawabata[†] and Mutsumi Suzuki[†,☆]

Matrix languages such as MATLAB have been widely used for numerical computations, especially as rapid prototyping tools. Those systems are basically interpreted to support typeless and flexible programming environments. Due to this fact, unfortunately, execution speed of a program written in a matrix languages is limited compared to a code in a general-purpose compiled language like Fortran. Recently, studies have revealed that the translation of matrix language scripts into programs written in compiled language would be a promising approach for high-speed computation with matrix languages. However, none of existing compilation systems seems to be able to handle sparsity of matrices attaining high-performance of translated codes. In this paper, we propose a compiler for large-scale sparse matrix computations, named CMC. Distinguishing features of CMC include the functionality to utilize sparse data structures and the optimization facility based on the detailed information of shapes of matrices, e.g., triangular, diagonal. CMC translates annotated MATLAB scripts into Fortran 90 programs. Experimental results show that the translated SOR and CG programs by CMC run seven times and three times, respectively, as fast as MATLAB interpretation, which confirms our method's effectiveness.

1. はじめに

数値計算プログラムの記述を容易にするために MATLAB をはじめとしていくつもの行列言語およびその実行環境が開発され，広く利用されている[1)~4)]．行列言語は行列を基本的なデータ構造として用意し，行列演算を基本演算として言語仕様で定義しているため，数値計算プログラムの記述に際しアルゴリズムの数学的表現との対応がとりやすい．また汎用高級言語を使用する場合に比べて一般に記述量も大幅に少なくて済むため，行列言語はラピッドプロトタイピング用の言語としても有用である．

行列言語処理系の実行環境は，インタプリタによるものとコンパイラでネイティブコードに変換するものに大別できるが，MATLAB[1)]，Scilab[2)] などの多くの行列言語はインタプリタ形式の実行環境を備えている．行列言語では一般に単独の演算子で表現される計算処理が汎用言語の場合と比較して重く，プログラムの構文の解釈に要する時間は行列演算に要する時間に比べて微小である．このため，行列積などの個々の行列演算を高速に処理できるライブラリルーチンが用意してさえあれば，インタプリタによる実行でもプログラム全体の処理時間は短く抑えられる．また，インタプ

† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University
☆ 現在，日立エスケイソーシャルシステム株式会社
Presently with Hitachi SK Social System Co., Ltd.

リタ実行環境であれば，プログラム中の各変数のデータ型やデータ構造および演算子の処理内容を実行時に動的に決定できるので，変数の型宣言を排除することもでき，可搬性の高い柔軟なプログラムの簡潔な記述が可能になる．代表的な行列言語である MATLAB は，以上の特徴から，世界的規模で多くのユーザから支持されている．

しかしながら，MATLAB をはじめとする既存の行列言語およびその処理系は必ずしも大規模数値計算コード開発に直接的には使用できない．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の観点で，行列言語による実行は Fortran や C 言語などのコンパイル言語記述による場合に劣るからである．とりわけ疎行列データ構造を用いる大規模計算では動的記憶管理のオーバヘッドが顕著となる．

大規模数値計算においては巨大な疎行列を用いた計算が頻出するが，汎用のコンパイル言語による疎行列計算コード開発は容易ではない．まず，疎行列の格納方法に工夫が必要で，単純な 2 次元配列を用いるのではなく，非零要素のみを効率的に格納するためのデータ構造が求められる．また，行列計算記述は多重ループ中で間接参照により非零要素をアクセスする形態にする必要がある．結果として，疎行列コードは複雑で開発や保守が困難なものとなる．こうした背景から，MATLAB で実現されているような単純な記法による疎行列コード開発が行える環境が求められている．しかしながら，従来の行列言語コードの実行環境は大規模疎行列計算に用いるには十分であるとはいい難い．

行列言語プログラムの高速化に関連する研究は数多く行われている．たとえば FALCON[5] 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は対応していない．

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[6),7)]．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[8)]．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[5),9)]☆．

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[9)]，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．

以上をふまえ，我々は，行列言語入力を受けて高速な疎行列計算コードを出力する行列言語コンパイラ CMC（a Compiler for Matrix Computations）を提案する．CMC は，ユーザに，可読性や保守性の高い行列言語を用いた大規模疎行列計算コードの開発環境を提供する．CMC は次の特徴を持つ：

- 入力言語の構文は，世界的に広く使用されている行列言語，MATLAB 言語に基づく．出力コードはコンパイル言語である Fortran 90 のプログラムである．
- 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も持ち，それに対応した出力コードを生成できる．
- 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．汎用言語における最適化機能に加え，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．

本論文では，行列言語コンパイラ CMC の設計について述べる．また，基本的な行列演算の 1 つである行列積の計算や，SOR 法，CG 法を用いた大規模連立一次方程式の求解などの実測に基づき，CMC の性能評価を行う．

本論文の構成は次のとおりである．まず 2 章では，CMC の入力言語のベースである MATLAB 言語および実行環境の特徴と，行列言語コンパイラによる高速化のアプローチについて述べる．3 章では，我々が開発した行列言語コンパイラ CMC の設計方針をまとめ，4 章，5 章，6 章でそれぞれ CMC の変数属性値解析機能，コード出力機能，最適化機能について述べ

☆ MATLAB Compiler は，2.3 節で述べるとおり，少なくとも現時点では，コード実行の高速化を目的として開発されているわけではない．

```
input: A ∈ R^{n×n}, x ∈ R^n, tol ∈ R
output: λ ∈ R, i ∈ N
  i ← 0
  λ ← 0
  while(true) begin
    i ← i + 1
    y ← Ax
    λ_new ← (y^T y)/(y^T x)
    exit if |λ − λ_new| ≤ tol.
    x ← y/||y||_2
    λ ← λ_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.

る．7 章では CMC の実行環境について触れ，8 章で実測に基づく CMC の性能評価を行う．9 章では，既存の研究との関連について述べる．

2. 行列言語 MATLAB とその高速化

2.1 MATLAB コードの概要

MATLAB コード記述の例を**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（function M-file）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 (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．

MATLAB の変数の属性値には次のものがある：☆

型 汎用言語における変数と同様で，その変数が保持する値の型は，論理型，整数型，実数型，複素数型に区別される．

サイズ MATLAB の変数は行列であり，複数の要素の集合であるので，保持する行列の次元数（サイズ）が属性の 1 つとなる．

構造 密行列か疎行列かの区別がある（次節参照）．

これらのほかに，派生的な変数属性として，スカラ，ベクトル，行列などの区別，すなわち**形状**を考えることができる．MATLAB では 1×1 の行列でスカラを表現し，また $1 \times n$ および $n \times 1$ の行列で n 次元列ベクトルと n 次元行ベクトルをそれぞれ表すが，本論文では便宜上，サイズとは独立に形状情報を変数属性の 1 つと見なす．

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．

2.2 MATLAB における疎行列の扱い

MATLAB 実行環境には疎行列を扱うためのデータ構造が用意されている．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[10]，Compressed Column Storage（CCS）形式[11] あるいは Compressed Sparse Column 形式[12] と呼ばれるものと実質的に同等である．CCS 形式では**図 2** に示すように 3 本の一次元配列で 1 つの行列を表す．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．

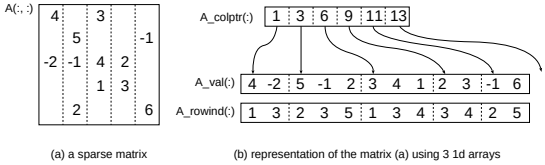

図 2 CCS 形式による疎行列の表現
Fig. 2 The structure of CCS format for sparse matrices.

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` に対応する実引数を疎行列にしても，適切に計算が行われる．

2.3 行列言語コンパイラによる変換と高速化

MATLAB のように型宣言のないインタプリタ方式の行列言語コードの高速化を考えるとき，“行列言語コンパイラ” によってコードをコンパイル言語記述に変換して動的処理に起因するオーバヘッドを排除しようとするのはきわめて自然な考え方である．ただし，インタプリタ環境あるいはランタイムライブラリの強力な支援なしで実行できるコードを生成するためには，行列言語で記述されたソースコードの汎用性を幾分か

☆ バージョン 5 以降の MATLAB には多次元配列やユーザ定義型（構造体）を扱う機能があるが，ここではそれらには触れない．

削って “専用化” したコードを出力せざるをえない．その専用化の程度によって行列言語コンパイラの実現方式は多様になりうるし，またコード変換の適用が必ずしも高速化につながるとは限らない．

MathWorks 社による MATLAB Compiler[8] は，スタンドアロンコード生成機能やコードのバイナリ化によるアルゴリズム隠蔽機能の提供を目的として開発されており[1),8)]，MATLAB コード記述の持つ機能をすべて保ったまま C 言語などのコンパイル言語記述に変換する☆．また変数の属性値の高度な静的解析機能も持たず，定数の畳み込みやループの上下限式の整数化などの限られた最適化機能しか持たない．結果として，MATLAB Compiler によるコード変換はコードの実行速度向上にはつながりにくく，実際，行列演算を中心とするコードはほとんど高速化できないことが報告されている[5),9)]．

これに対して FALCON[5] はコードの高速化を主眼として開発されている行列言語コンパイラである．ただし，コード中で使用するデータがコンパイル時に参照可能でなければ，必ずしも効率的なコード生成は行えない．いい替えれば，FALCON は，入力データの属性値をある程度限定して “専用化” したコードを生成することにより高速化を図るコンパイラであるといえる．なお FALCON は，いくつかのアプリケーションで MATLAB コードを高速化できるものの，疎行列データ構造を扱う機能には対応しておらず，大規模数値計算への応用は困難である．

3. 行列言語コンパイラ CMC

我々は，行列言語記述を汎用言語記述に変換するコンパイラ CMC を設計，開発した．CMC の開発目的は，行列言語記述の可読性や保守性，機能の高さと，コンパイル言語による静的記述の高速性の両者を生かした，行列言語記述をベースとした大規模数値計算コード開発環境の実現である．

本章では，CMC の設計方針について，FALCON[5] などの既存のシステムと比較しつつ述べる．

3.1 簡潔なプログラム記述からコンパイル言語記述を生成

CMC の入力言語は，広く使用されている行列言語である MATLAB をベースとする．そして静的解析によりプログラム中の変数の属性や演算内容を判別し，動的処理を削減して Fortran 90 で記述されたコードを出力する．

MATLAB のように変数の型宣言がない言語であっても，各変数の型や構造は，プログラム中で使用されている様子からある程度は静的に判別できる[5)]．プログラムの静的解析によって各変数の型が完全に決定できない状況としては，関数の引数やファイル入力など外部から与えられるデータによる場合，あるいは，実行時に動的に型や形状の変換がなされる場合が考えられる．CMC の入力記述としては，前者に対しては指示行によりユーザからの指示を仰ぐことにし，後者は CMC の入力言語仕様としては認めない．すなわち，CMC の対象とするコードは，すべての変数の属性値が静的に（必要最小限のユーザ指示を受けたうえで）決定可能なものとする．

FALCON は，入力変数の属性値を指示行ではなく具体的に用いるデータから直接得るため，少なくともサンプルの入力ファイルを用意して各入力変数の型や形状を外部から指示する必要がある．これに対し CMC は，必要最小限の変数属性値に関する指示行があれば，入力データがコンパイル時に利用可能でなくても Fortran 90 コードを生成できる．

CMC がユーザに要求する変数属性指示は，もともとそのコードを使用するうえでユーザが知っておくべき事項を改めてコード中に記載するにすぎないため，ドキュメントとしての効果，すなわちプログラムの可読性を高める効果もある．加えて，属性指示を用いればコンパイラによる静的な型チェックが可能になるので，コードの保守性向上にも効果がある．

3.2 疎行列データ構造への対応

大規模数値計算プログラム記述を行うためには疎行列を扱う機能が必須である．また，変数の構造が疎行列であるか否かによらない数式記述への対応が望まれる．CMC ではまず，CCS 形式の疎行列データ構造を用いてこれらに対応する．汎用言語で疎行列データ構造を扱う処理を記述するのは繁雑であるが，CMC を用いれば，簡便な行列言語記述を基にして，CCS 形式の疎行列構造に対応したコード記述を生成することができるようになる．

疎行列データ構造への対応は，大規模数値計算のためには必須であるにもかかわらず，FALCON をはじめ多くの行列言語コンパイラに欠けている機能である．

なお我々は，将来的な CMC の拡張案として，CCS 形式だけでなく複数の疎行列データ構造を利用できるようにすることも考慮中である．

☆ 古いバージョンの MATLAB Compiler は “配列境界のランタイムチェックの無視” や “複素数を扱えないコードの生成” などの “unsafe” な変換機能を備えており，これによってコードの実行速度向上を図ることも一応は可能であった．しかし現在のバージョン（Ver.3）ではこれらの機能は用意されていない．

3.3 余分な動的処理の排除による高速化

コンパイル言語によるプログラム記述と比較してMATLABなどの行列言語の実行環境の処理速度が遅い原因は，動的な記憶管理のオーバヘッドが大きいことである．CMCでは，コンパイル後のコードの実行の高速性を優先し，プログラム実行時の変数の構造や形状の動的変更には対応しない．この点においては，CMCによるコードの“専用化”はFALCONのそれよりも程度が高い．しかしながら我々は，大規模数値計算プログラムにおいて変数の型や形状を動的に変更する必要性はわずかであると考える．

ただし，CMCの出力コードも若干の動的処理は行う．たとえば，多数の項を持つ長大な式の行列演算を実行する際には中間結果を保持する記憶領域が大量に必要となる場合もあるので，CMCにおいても，アロケートやデアロケートなどの動的なメモリ管理には対応する．また，行列演算において実際に採用する計算アルゴリズムをプログラム実行中に選択するなどの処理も行う（5.2節参照）．

3.4 行列言語であることを生かした最適化機能

行列演算はスカラ演算に比べてソースコードにおける演算子あたりの計算量が多く，共通部分式の削除やループ不変部分式の移動などのいわゆる古典的最適化がプログラムの実行速度の大幅短縮につながる場合も多い．我々は，行列言語コンパイラにとってはこのようなソースレベル最適化は必須の機能であると考え，CMCに最適化機能を持たせる．

FALCONがソースレベル最適化に対応している旨の記述は見られない．またMATLABシステムでも，MATLAB Compilerを含め，ソースレベル最適化には対応していない．

4. CMCにおけるプログラム解析

4.1 CMCによるコンパイル処理の流れ

図3に，CMCを用いたコンパイル処理の流れを示す．CMCは入力としてMATLABで記述された関数（function M-file）を受け，Fortran 90で記述されたサブルーチンを出力する☆．出力コードはFortran 90で記述されるので，必要に応じてユーザは別に用意したルーチンとリンクしてロードモジュールを作成することもできる．

なお，CMCに実装されているソースレベル最適化機能のみを適用した結果をMATLAB言語のままで出力することもできる．

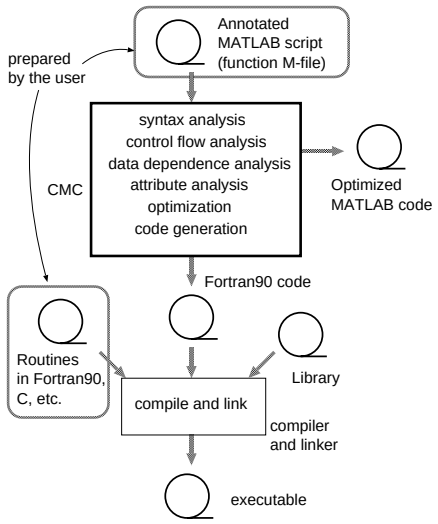

図3 CMCによる処理の流れ
Fig. 3 Building a sparse code using CMC.

4.2 変数の属性の解析

CMCは，入力コード記述（関数）の仮引数の変数について，それぞれの属性値，すなわち型，サイズ，構造，形状の情報をユーザ指示行により受付ける．それ以外の変数の属性値については，次の情報をソースとして用いることにより，反復アルゴリズムによる前進型データフロー解析[13]により静的に決定する．

- ユーザからの指示．
- コード中の定数．
- 組み込み関数の返り値．

組み込み関数の返り値の属性は多くの場合既知である．たとえば関数`length()`の返り値は整数型であり，関数`triu()`は返り値が行列の場合の形状は上三角である．関数`diag()`は引数の形状に応じて対角行列あるいは列ベクトルを返す．

以下の各節では，それぞれの属性の検出手順について述べる．いずれの属性値の検出も，行列演算におけるオペランドの変数の属性値と演算子との関係から演算結果の属性値を判別する．なお，反復処理による静的解析中，オペランドの属性値が判明していない時点では演算結果の属性値も決定できないが，前進型データフロー解析の反復によって最終的にはすべての属性値を決定できる（3.1節参照）．

なお型およびサイズの決定手順は文献5)の方法とほぼ同様である．

4.2.1 変数の型の決定

変数の型は，次の順序関係に基づいて判別する．

論理型 $\prec$ 整数型 $\prec$ 実数型 $\prec$ 複素数型

☆ 入力ファイルがscript M-fileであれば，Fortran 90のメインルーチンの形で出力する．

表 1 データの形状の変換規則
Table 1 Data structure translation rules.

(a) `A*B` の計算結果の形状

`A` の形状	`B` の形状						
	S	R	C	U	L	D	F
Scalar	S	R	C	U	L	D	F
Row vec.	R	—	S	R	R	R	R
Column vec.	C	F	—	—	—	—	—
Upper tri. mat.	U	—	C	U	F	U	F
Lower tri. mat.	L	—	C	F	L	L	F
Diagonal mat.	D	—	C	U	L	D	F
Full mat.	F	—	C	F	F	F	F

(b) `A+B`, `A-B`, `A./B`, `A.*B` の計算結果の形状

`A` の形状	`B` の形状						
	S	R	C	U	L	D	F
Scalar	S	R	C	U†	L†	D†	F
Row vec.	R	R	—	—	—	—	—
Column vec.	C	—	C	—	—	—	—
Upper tri. mat.	U†	—	—	U	F	U	F
Lower tri. mat.	L†	—	—	F	L	L	F
Diagonal mat.	D†	—	—	U	L	D	F
Full mat.	F	—	—	F	F	F	F

†：+/-の場合は，スカラ値がゼロでなければ F にする．

たとえば整数型と実数型の変数の積は実数型と見なす．ただし，整数型どうしの除算結果は実数型にするなどの特別扱いが必要である．

4.2.2 変数の形状の決定

MATLAB では形状情報は派生的な属性値であるが，CMC では形状情報を独立した変数属性と見なす．これは，より効率的な Fortran 90 コード出力，すなわち，スカラをスカラ変数で，ベクトルを一次元配列で扱うなどを行いやすくするためである．また，特殊な行列形状に応じて最適化したコード出力を可能にするために，三角行列や対角行列などのより詳細な形状情報も把握する．

各演算子の計算結果の形状はオペランドの形状によって決まる．CMC では，各演算について，**表 1** に示す規則を適用して演算結果の形状を決定する．たとえば式 `C=A*B` において行列 `A`，`B` がそれぞれ対角行列，上三角行列ならば，行列 `C` の形状は上三角行列と決定される．

4.2.3 変数のサイズの決定

形状の場合と同様，演算子とオペランドとから計算結果のサイズが決定できる．たとえば式 `C=A*B` における行列 `C` のサイズは，`A`，`B` の一方がスカラなら他方のサイズと同じで，両方ともスカラでないなら `A` の行数と `B` の列数との積となる．

4.2.4 変数の構造の決定

ベクトルやスカラは疎行列データ構造にはせず，行列の場合のみ疎行列データ構造の利用を考慮する．演算結果が行列である場合は，その構造を次のようにオペランドの構造に基づいて決定する．

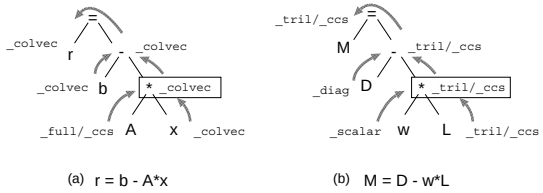

図 4 式中の中間変数と左辺の型などの決定
Fig. 4 Type inference for intermediate variables.

- 疎行列どうしの演算：算結果は疎行列.
- 疎行列とスカラ：減算であれば演算結果は密行列☆，それ以外であれば疎行列.
- 密行列と疎行列/密行列どうし：算結果は密行列.
- 組み込み関数の出力が行列の場合は，基本的には入力のデータ構造と同じにする.

図 4 に，式中の中間変数の型や形状などの決定例を示す．図では式を木構造で表現している．図に示しているように，葉に相当する変数の情報がすでに得られているならば，式の木を根の方向にたどり，式全体の計算結果の属性を決めることができる．図 4 (a) および (b) の式では，右辺式の計算結果が変数 `r` および `M` に代入されていることから，(変数 `r`，`M` について個別のユーザ指示がなかった場合は）`r` は列ベクトル，`M` は下三角の疎行列と決定される．なお図中の枠をつけた部分では中間の値を作業領域へ保存する必要があることもある．CMC は必要に応じて，検出した属性値に従って作業領域の確保と解放を行う（5.1 節参照）．

4.3 ユーザ指示行について

CMC への入力となる MATLAB の関数記述における仮引数の属性値の情報は，ユーザからの指示を受ける．**図 5** に，SOR 法の MATLAB コードに仮引数の属性を指定する指示行を加えた例を示す．指示行は `%cmc` で始まる行で，記述は Fortran 90 の変数属性指定に類似の形式である．なお MATLAB においては各行の '`%`' 以降の文字列はコメントと見なされるので，CMC 用の指示行を加えたコードでも MATLAB インタプリタで実行できる．

現状の CMC の実装で指定できる情報は次のとおりである：

- 型宣言：複素数型（`complex`），実数型（`real`），整数型（`integer`），論理型（`logical`）．それぞれ，Fortran における `COMPLEX*16`，`REAL*8`，

☆ MATLAB ではスカラと行列の和/差は，行列の各要素についてそのスカラとの和/差をとって得られる行列として定義される．

```
function [x,i] = sor0s(A,x0,b,tol)
%cmc integer,parameter ::  s=50*50
%cmc real,_ccs(5) ::  A(s,s)
%cmc real,_colvec ::  x0(s), b(s)
%cmc real,_scalar ::  tol
w = 1.8;
D = diag(diag(A)); L = -tril(A,-1); U = -triu(A,1);
r0 = norm(b-A*x0);
x = x0; i = 0;
while 1
  i = i+1;
  x = (D-w*L)\(((1-w)*D+w*U)*x+w*b);
  if norm(b-A*x)/r0 <= tol, break, end
end
```

図 5 MATLAB で記述した SOR 法のコード
Fig. 5 The SOR method in MATLAB.

INTEGER，LOGICAL に対応．

- 形状指定：長方形行列（_full；デフォルト），行ベクトル（_rowvec），列ベクトル（_colvec），上三角行列（_triu），下三角行列（_tril），対角行列（_diag），スカラ（_scalar）．
- 行列の構造指定：密行列（_dense；デフォルト）あるいは疎行列（_ccs）．疎行列であれば，各列の非零要素数の平均値 n を_ccs(n) のように指定．n を省略した場合は別に定めるデフォルト値となる．
- 行列やベクトルのサイズの指定（次元数の指定）．

図 5 のコードの指示行は，たとえば変数 A が s×s の疎行列で非零要素数が s×5 以下であることを示している☆．

5. CMC によるコード出力

5.1 MATLAB から Fortran 90 への変換

Fortran 90 の演算子や組み込み関数のいくつかは，MATLAB 同様，行列演算に対応しているため，密行列に対する計算に限っては単純な演算子の置換えで Fortran 90 記述に変換できる．一方，疎行列の行列演算は CCS 形式での計算に書きあらためる必要がある．

CMC による行列計算式の変換例を**図 6** に示す．図 6 (a) の MATLAB 記述は，行列 A が密行列か疎行列かによって図 6 (b) あるいは図 6 (c) のように変換される．疎行列は CCS 形式で扱い，図に示すように，たとえば行列 A を 3 本の一次元配列 A_val，A_colptr，A_rowind で表現する（CCS 形式については図 2 も参照）．

なお，図 6 (c) は図 4 (a) の構造をそのまま出力にマッピングしたものであるが，実際には CMC はコード出力時に積和式の合成を行うので，出力コードは

```
r = b - A*x;
```

(a) An expression in MATLAB (A: matrix, r,b,x: column vectors).

```
r = b - MATMUL(A,x)
```

(b) Corresponding Fortran 90 code with dense A.

```
allocate(Xr1(2500))
Xr1 = 0
do Xk = 1, 2500
  do Xiptr = A_colptr(Xk), A_colptr(Xk+1)-1
    Xi = A_rowind(Xiptr)
    Xr1(Xi) = Xr1(Xi) + (A_val(Xiptr)) * (x(Xk))
  enddo
enddo
r = (b) - (Xr1)
deallocate(Xr1)
```

(c) Corresponding Fortran 90 code with sparse A using CCS format.

```
r = b
do Xk = 1, 2500
  do Xiptr = A_colptr(Xk), A_colptr(Xk+1)-1
    Xi = A_rowind(Xiptr)
    r(Xi) = r(Xi) - A_val(Xiptr) * (x(Xk))
  enddo
enddo
```

(d) Optimized Fortran 90 code with sparse A using CCS format.

図 6 MATLAB から Fortran 90 への変換例
Fig. 6 Examples of MATLAB-to-Fortran 90 translation.

図 6 (d) のようになる（6.2.2 項参照）．

5.2 CMC における疎行列計算のアルゴリズム

5.2.1 疎行列とベクトルとの積

疎行列とベクトルとの積は図 6 (d) のコードの 2 行目以降の部分に示されている．疎行列の全要素を二重ループ中で参照し，その数だけ乗算および減算が行われるので，疎行列の要素数に比例する量の計算が必要である．

5.2.2 疎行列どうしの和

CMC では，疎行列どうしの和は特に指定しない限り疎行列に格納する．和の計算 C=A+B の実行時の各行列の要素の参照の様子を**図 7** に示す．

行列の加算に要する計算は，オペランドの行列の非零要素数回とほぼ同数の加算である．しかし，疎行列の計算においては計算結果を CCS 形式の疎行列データ構造に詰めて格納する操作のコストが比較的大きい．どのような手順でデータを詰めるべきかは，非零要素の割合や配置によって変わる．CCS 形式どうしの行列の和は図 7 に示すように列ごとに値が順に計算されるので，次の 2 通りが考えられる：

(1) 列中の非零要素をすべて計算してから，1 列分を圧縮して疎行列データ構造に詰める（作業領

☆ CMC の出力コードは行列の領域境界の動的なチェックを行わないので，指示行での宣言範囲を越える領域の参照が行われた場合にはランタイムエラーが発生する．

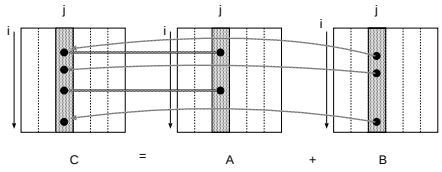

図 7 CCS 形式の行列の和 (C=A+B)
Fig. 7 Sparse matrix addition (C=A+B) using CCS format.

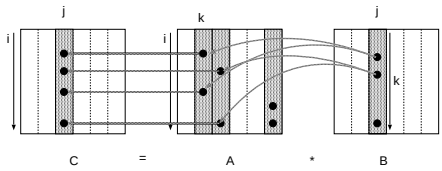

図 8 CCS 形式の行列の乗算 (C=A*B)
Fig. 8 Sparse matrix multiplication (C=A*B) using CCS format.

```
allocate(Xflag(10000))
Xflag = 0
do Xj = 1, 10000
  do Xiptr = A_colptr(Xj), A_colptr(Xj+1)-1
    Xi = A_rowind(Xiptr)
    Xflag(Xi) = Xflag(Xi) + 1
  enddo
enddo
B_colptr(1) = 1
do Xj = 1, 10000
  B_colptr(Xj+1) = B_colptr(Xj) + Xflag(Xj)
  Xflag(Xj) = B_colptr(Xj)
enddo
do Xj = 1, 10000
  do Xiptr = A_colptr(Xj), A_colptr(Xj+1)-1
    Xi = A_rowind(Xiptr)
    B_val(Xflag(Xi)) = DCONJG(A_val(Xiptr))
    B_rowind(Xflag(Xi)) = Xj
    Xflag(Xi) = Xflag(Xi) + 1
  enddo
enddo
deallocate(Xflag)
```

図 9 CCS 形式の行列の共役転置 (B=A')
Fig. 9 Complex conjugation of a sparse matrix (B=A') using CCS format.

域大/次元数に比例する量の処理が必要).

(2) 各非零要素を作業領域に詰めつつ計算し, 1 列分を並べ換えて疎行列データ構造に格納する (作業領域小/非零要素のソート処理が必要).

計算結果の行列の非零要素の割合が大きい場合は (1) が, 小さい場合は (2) が有効だと考えられる. CMC は, 第 1 列目の計算結果の非零要素数を元にしてどちらの手順を採用するかを動的に決定するコードを生成する.

5.2.3 疎行列どうしの積

CMC では, 疎行列どうしの積は特に指定しない限り疎行列に格納する. CCS 形式への格納は第 1 列から最終列まで順に行うのが都合がよいので, 乗算アルゴリズムはいわゆる JKI 型としている. 行列積 C=A*B に対する行列要素の参照の様子を**図 8** に示す.

オペランドの両行列が次元数 n の正方行列であるとし, 列あたりの非零要素の平均個数を d とすると, CCS 形式の乗算にはおよそ nd^2 回の乗算および加算が必要となる. ただし, 計算結果の非零要素が少ない場合 ($d \ll n$) には, 和の場合と同様, 計算結果を疎行列データ構造に詰め込む作業の手順が実行時間を支配する. CMC は, 疎行列の和の場合と同様に, データの計算および格納手順を動的に選択するコードを生成する.

5.2.4 疎行列の転置

CMC では, 疎行列の転置および共役転置結果は特に指定しない限り疎行列に格納される. 共役転置操作 B=A' に対応する出力コードを**図 9** に示す. 図中, 一次元配列 Xflag は作業用であり, 処理の前後でアロケートやデアロケートを行う.

転置操作は実数演算をともなわないが, 疎行列の非零要素数に比例する回数の量の間接参照によるデータ移動を要する処理である.

なお, 入力記述が B=A' であっても, 変数 A が複素数型でない場合には, CMC は単なる転置操作 B=A.' と等価と見なし, 図 9 において DCONJG() を省いたコードを出力する.

5.2.5 疎行列用の領域確保について

計算結果の行列を CCS 形式でメモリに格納する場合にどの程度の容量を確保すればよいかは単純には決定できない. CMC では関数の引数が疎行列である場合は指示行による指定に頼り (4.3 節参照), 疎行列どうしの和/差の保持する記憶領域の大きさは, 双方の疎行列の大きさの合計値にする. しかし疎行列どうしの積などは非零要素数をあらかじめ静的に見積もることが難しいので, そのような場合は (作業領域用変数をユーザが明記したうえで) _ccs(n) によりユーザが大きさを宣言するものとする.

なお MATLAB インタプリタによる実行では, 確保してある領域を越えた位置への代入が試みられた時点で動的に記憶領域が拡張される. しかし添字の境界のチェックや記憶領域の動的管理のオーバヘッドは比較的大きい. CMC ではこのような動的な記憶領域の拡張には対応しない.

5.3 ライブラリルーチンの利用

CMC は, ライブラリルーチンを利用する形式でのコード出力にも対応し, 実行時オプションで出力形式を選択できるようにしている. たとえば図 6 (a) のコードに対応する疎行列コード出力は, 図 6 (c), (d) のような記述以外に, **図 10** に示すように行列演算をライ

```
  allocate(Xr1(2500))
  call ccs_mat_mul_vec(
 &     A_val, A_colptr, A_rowind,x, Xr1, 2500)
  r = (b) - (Xr1)
  deallocate(Xr1)
```

図 10 ライブラリルーチン呼び出しの形式での出力
Fig. 10 Output code utilising library routines.

```
SUBROUTINE sor0s(A_val,A_colptr,A_rowind,x0,b,tol,x,i)
IMPLICIT NONE
INTEGER,PARAMETER ::  s = 50*50
REAL*8 A_val(s*5)
INTEGER A_colptr(s+1), A_rowind(s*5)
...
```

図 11 図 5 に対して CMC が生成したコードの一部
Fig. 11 A part of the generated code from Fig. 5 by CMC.

ブラリルーチンに任せる形式にもできる．

ライブラリルーチンを利用する出力形式では，計算機ごとにチューニングしたライブラリを用意しておくことにより，変換後のコードの性能可搬性を高いレベルで保ちやすいと考えられる．ただし現状の CMC 用行列計算ライブラリには，行列積や行列とベクトル積などの簡単な処理を行うルーチンしか実装されていないため，複数の行列演算が連続する場合には図 10 のように中間変数が必要となる．一方，ライブラリを用いない形式では図 6 (d) のように作業領域を必要としないコード出力が可能である（6.2.2 項参照）．

5.4 変換後のコードの実行

CMC の出力は Fortran 90 のサブルーチンであり，他の Fortran 90 ルーチンなどとリンクして実行することができる．出力されるサブルーチンの仮引数は，MATLAB コードにおける仮引数と返り値である．

疎行列に指定した仮引数の行列は，CCS 形式に基づく 3 本の一次元配列で与える．**図 11** に，図 5 を CMC で変換して得られるコードの先頭部分を示す．行列 `A` がベクトル `A_val`，`A_colptr`，`A_rowind` で表されている．

6. CMC の最適化機能

6.1 MATLAB ソースコードレベルでの最適化

6.1.1 古典的最適化機能

CMC が対応している古典的最適化機能[13] は次のとおりである．

- 共通部分式の削除
- ループ不変部分式の切り出しと移動
- 複写伝播
- 無用命令の削除

これらの最適化を行うためには，プログラム中の変数の定義・引用関係の情報の解析，すなわちデータ依存関係解析が必要となる．CMC はデータフロー方程式の反復解法に基づいた依存解析[13] を行う．

なお，古典的最適化や行列乗算の優先順位の最適化は MATLAB のソースコードレベルでのプログラム変換であり，CMC はこの最適化を適用後のコード出力機能も持つ．これにより，CMC を用いてソースレベルでの最適化を施したコードを MATLAB インタプリタで実行することもできる．

6.1.2 行列計算の計算順序決定

行列の乗算はスカラの乗算と同様に結合法則を満たすので，`A*B*C` のような複数の乗算を含む項は演算実行順序に任意性がある．行列やベクトルの積では，スカラ変数どうしの積の場合と比較してその演算順序がプログラムの実行時間に与える影響が大きいので，CMC では，連続的な乗算に関してどの順序で積を計算するかを見積もって最適化する．

たとえば変数 `s`，`A`，`r` がそれぞれスカラ，行列，列ベクトルである場合，`s*A*r` の計算は，`(s*A)*r` よりも `s*(A*r)` とすべきである．より詳細には，各変数のサイズから計算コストを評価できる．なお，疎行列がからむ積では厳密な計算量や所要記憶容量を静的に決定することはできないが，CMC の現在の実装ではすべての行列を密行列と見なした場合のコスト評価に基づいて優先順位を決めている．

ここで，図 5 で示した SOR 法のコードに古典的最適化を適用した結果を**図 12** に示す．ループ不変部分式が検出され，ループ外に移動されていることが分かる．また**図 13** (a) の CG 法のコードにソースレベル最適化を適用すると図 13 (b) となる．ループの反復をまたがる部分式の移動により，ループ中で 3 回行われていた内積計算（`r.'*r` や `rnew.'*rnew`）が，1 回にまとめられているのが分かる．

なお，共通部分式の削除とループ不変部分式の移動は互いに他を妨げる場合があるが，行列計算の重さを考慮し，CMC では共通部分式の削除を優先する．たとえば図 5 のコードに対しては，ループ不変部分式 `alpha*A` をループ外に移動するのではなく，（ループ不変ではない）共通部分式 `A*p` の括り出しを行う．

6.2 コード出力時の最適化

6.2.1 行列の形状に応じたコード出力

CMC は対角行列や三角行列などの形状を解析する機能を有する（4.2 節参照）．たとえば図 12 中の `TMPM00` や `TMPM01` がそれぞれ下三角，上三角行列であることを判定できる．このような詳細な形状の把握には次の利点がある：

- 対角行列の値の保持は一次元配列で済ませられる．

```
function [x, i] = sor0s(A, x0, b, tol)
%cmc integer,parameter ::  s=50*50
%cmc real,_ccs(5) ::  A(s,s)
%cmc real,_colvec ::  x0(s), b(s)
%cmc real,_scalar ::  tol
w = 1.8;
D = diag(diag(A)); L = -tril(A,-1); U = -triu(A,1);
r0 = norm(b-A*x0);
x = x0;
i = 0;
TMPLM00 = D-w*L;
TMPLM01 = (1-w)*D+w*U;
TMPLV02 = w*b;
while 1
  i = i+1;
  x = TMPLM00\(TMPLM01*x+TMPLV02);
  if (norm(b-A*x)/r0<=tol), break, end
end
```

図 12 ソースレベル最適化の適用後の SOR 法のコード
Fig. 12 The optimized SOR code in MATLAB.

- 三角行列や対角行列が演算のオペランドになっていれば，零要素の参照を省くことができる．
- 代入文の左辺変数が三角行列や対角行列であれば，零要素に帰する値のための計算を省ける．
- 演算子 '\' の左側に対角行列や三角行列があれば，MATLAB のような動的な形状検査（コストは次元数 n に対して $O(n^2)$）を省き，前進代入や後退代入などを直接適用することができる．

以上のように，三角行列や対角行列の検出は記憶領域，計算量の削減に寄与しうる．ただし，三角行列への対処はそれが密行列でないと効果が少ないので，現状の CMC の実装では，三角行列については密行列の計算の場合に限り形状情報をコード生成に反映させている．

6.2.2 複数の項の加算をまとめたコード出力

CMC では，積や和などの個々の行列演算処理を単純にライブラリルーチンに任せるのではなく，任意の積和形で表される式による演算を総合して Fortran 90 のコード記述に変換する．この変更を行っても計算量は変化しないが，中間変数を保持する記憶領域が不要になり，その分だけメモリへのロード・ストアの量を削減することができる（図 6 (c) と (d) を比較されたい）．また，中間の結果が疎行列である場合には，CCS 形式にするためのデータ圧縮の回数を削減することができる．

7. CMC の開発環境および実行環境

CMC システムは，Perl スクリプトのフィルタやドライバと，lex/yacc および C++言語で記述されているコア部分とからなる．CMC の開発においては，プラットフォーム依存性は極力排除している．

CMC は，コマンドラインでの実行ができるほか，

```
function [x,i] = cg0s(A,x0,b,tol)
%cmc integer,parameter ::  s=50*50
%cmc real,_ccs(5) ::  A(s,s)
%cmc real,_colvec ::  x0(s), b(s)
%cmc real,_scalar ::  tol
r = b-A*x0;
rn = norm(r);
x = x0;
p = r;
i = 0;
while 1
  i = i + 1;
  alpha = (r.'  * r) / (p.'  * A * p);
  x = x + alpha * p;
  rnew = r - alpha * A * p;
  if norm(rnew)/rn <= tol, break, end
  beta = (rnew.'  * rnew) / (r.'  * r);
  p = rnew + beta * p;
  r = rnew;
end
```

(a) The CG method.

```
function [x,i] = cg0s(A,x0,b,tol)
%cmc integer,parameter ::  s=50*50
%cmc real,_ccs(5) ::  A(s,s)
%cmc real,_colvec ::  x0(s), b(s)
%cmc real,_scalar ::  tol
r = b-A*x0;
rn = norm(r);
x = x0;
p = r;
i = 0;
TMPX00 = A * r ;
TMPS01 = r.'  * r;
while 1
  i = i + 1;
  alpha = TMPS01 / (p.'  * TMPX00);
  x = x + alpha * p;
  rnew = r - alpha * TMPX00;
  if norm(rnew)/rn <= tol, break, end
  TMPS02 = rnew.'  * rnew;
  beta = TMPS02 / TMPS01;
  p = rnew + beta * p;
  r = rnew;
  TMPX00 = A * p;
  TMPS01 = TMPS02;
end
```

(b) The optimized CG code.

図 13 CG 法のコードへのソースレベル最適化の適用
Fig. 13 Source-level optimization of the CG method in MATLAB.

Perl/Tk で記述した GUI による実行環境も備えている．ソースコードエディタとの連携機能のほか，コントロールフローグラフやデータフローグラフの描画機能など，ユーザによるプログラム開発を支援する機能も備えている．

GUI による各種フローグラフの描画機能は，プログラム開発や保守において，コードの挙動の把握などに有用であると考えられる．一例として，図 13 (b) に対応するコントロールフローグラフおよびデータフローグラフの描画例を**図 14** に示す．

8. 実測と評価

本章では，開発した行列言語コンパイラ CMC の有効性の評価のために行った実測結果について述べる．

まず，基本的な行列演算の 1 つである行列積につい

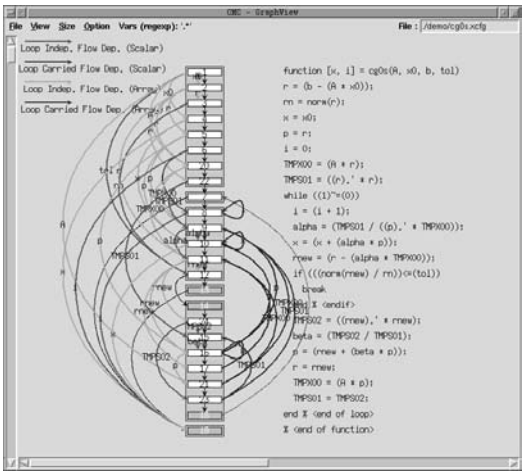

図 14 CMC によるフローグラフの描画例
Fig. 14 The flowgraph viewer of CMC.

て，密行列どうしの積および疎行列どうしの積のそれぞれの場合の実行結果を MATLAB インタプリタでの実行結果と比較しつつ示す．次に，SOR 法および CG 法による連立一次方程式の求解コードを MATLAB 言語で記述して実測した結果を示す．加えて，CMC のソースレベル最適化機能の効果についても実測結果を元に示す．

SOR 法および CG 法のコードの実測にあたっては，MATLAB Compiler（以降では MCC と呼ぶ）を用いて C 言語によるスタンドアロンコードに変換した場合の測定結果も比較のために示す．

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 GP7000F（SPARC64-GP，主記憶 24 GB）を使用．SMP システムだが 1 CPU のみ使用．
- MATLAB のバージョン：6.5.1.199709 R13 (SP1)
- Fortran 90 コンパイラ：`frt`（Fujitsu Fortran Compiler Driver Version 5.5）
- `frt` コンパイルオプション：`-Kfast_GP=2 -X9`

MCC での実測において用いたコンパイラは次のとおりである：

- MCC：`mcc` バージョン 3.0
- `mcc` のオプション：`-m` 相当☆☆

☆ 京都大学学術情報メディアセンター（京大センター）のシステムを使用．

☆☆ 京大センターの MATLAB には MCC がインストールされていなかったため，C 言語コードの生成とリンクは広島市立大学設置の MATLAB（同一バージョン）で行った．C 言語コードのコンパイルは京大センターの `fcc` で行なったほか，MATLAB のダイナミックリンクライブラリも京大センターのものを用いた．

表 2 行列積に要する実行時間
Table 2 Execution times for matrix multiplication.

(a) $F \times F$ (F: dense)

次元数	MATLAB[sec]	CMC[sec]
200	0.03(1.0)	0.0334(1.1)
400	0.23(1.0)	0.185(0.80)
800	1.80(1.0)	1.32(0.73)

(b) $S \times S$ (S: sparse)

次元数	MATLAB[sec]	CMC[sec]
40000	0.27(1.0)	0.161(0.60)
160000	1.03(1.0)	0.675(0.66)
640000	4.98(1.0)	2.73(0.55)

- C コンパイラ：`fcc`（Fujitsu C Compiler Driver Version 5.4）
- `fcc` コンパイルオプション：`-Kfast_GP=2`

CMC での実測は，5.3 節で述べたライブラリルーチンの組み込みを行わない出力コードで行った．

計時は，MATLAB および MCC では組み込み関数 `cputime` 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．

数値実験で用いたデータは次のとおりである．

- 密行列 F と列ベクトル v：
  $F = (f_{i,j}) \in \mathbf{R}^{n \times n}$，$v \in \mathbf{R}^n$，
  $$f_{i,j} = \begin{cases} 1 & (i = j) \\ -3^{-|i-j|} & (i \neq j) \end{cases}$$
  $v = (1, 0, \ldots, 0)^T$
- 疎行列 S と列ベクトル u：
  二次元正方領域において Laplace 方程式
  $\partial^2 P/\partial x^2 + \partial^2 P/\partial y^2 = 0$
  を正方格子上の自然な順序付けの 5 点差分近似により解く際に現れる係数行列 S と，ディリクレ境界条件として一辺で $P = 1$，他の 3 辺で $P = 0$ とする場合に対応する右辺ベクトル u

行列 F，S とも実正定値対称である．

8.1 基本的な演算性能の評価：行列積

行列積 $F \times F$ および $S \times S$ を計算した．S および F はそれぞれ疎行列データ構造（CCS 形式），密行列データ構造（二次元配列）に格納して計算した．それぞれの実行に要した計算時間を**表 2** に示す．表中，実行結果に添えて括弧内に示す数値は，MATLAB での所要時間に対する比である．なお，密行列データ構造での行列積コードの CMC による変換結果は，Fortran 90 の組み込み関数 `MATMUL (F,F)` を実行するだけのコードである．

表 2 (a)，(b) より，密行列，疎行列のいずれの場合

表 3 SOR 法の実測結果（疎行列 S に対して $Sx = u$ を解いた場合）
Table 3 Experimental results on the SOR method (solving $Sx = u$ where S is sparse).

	MATLAB [sec]		MCC [sec]	CMC/dense [sec]		CMC/sparse [sec]	
次元数/反復回数	dense	sparse	sparse	noshape	shape	noshape	shape
400/64	0.38(4.8)	0.08(1.0)	0.07(0.88)	0.347(4.3)	0.268(3.4)	0.0105(0.13)	0.00932(0.12)
900/81	8.75(40)	0.22(1.0)	0.20(0.91)	4.90(22)	2.87(13)	0.0312(0.14)	0.0286(0.13)
1600/158	54.5(71)	0.77(1.0)	0.74(0.96)	31.9(41)	18.9(25)	0.107(0.14)	0.0980(0.13)
2500/247	203(109)	1.87(1.0)	1.87(1.0)	115(61)	66.5(36)	0.271(0.14)	0.252(0.13)
4900/472	—	7.27(1.0)	7.22(0.99)	—	—	1.10(0.15)	1.01(0.14)
8100/755	—	19.9(1.0)	19.4(0.97)	—	—	3.13(0.16)	2.68(0.13)

反復回数は行ごとの各実行についてすべて同じであった．括弧内の数値は各行における数値の相対比で，下線部は行ごとの最速値である．

表 4 SOR 法の実測結果（密行列 F に対して $Fx = v$ を解いた場合）
Table 4 Experimental results on the SOR method (solving $Fx = v$ where F is dense).

	MATLAB [sec]		MCC [sec]	CMC/dense [sec]		CMC/sparse [sec]	
次元数/反復回数	dense	sparse	dense	noshape	shape	noshape	shape
100/322	0.13(1.0)	0.49(3.8)	0.14(1.1)	0.0960(0.74)	0.0718(0.55)	0.115(0.88)	0.115(0.88)
200/1156	1.62(1.0)	6.79(4.2)	1.63(1.0)	1.30(0.80)	0.923(0.57)	1.54(0.95)	1.55(0.96)
300/2404	7.53(1.0)	31.8(4.2)	7.47(0.99)	5.97(0.79)	4.19(0.56)	7.17(0.95)	7.15(0.95)
400/4020	22.5(1.0)	127(5.6)	43.6(1.9)	17.4(0.77)	12.3(0.55)	25.6(1.1)	23.2(1.0)

反復回数は行ごとの各実行についてすべて同じであった．括弧内の数値は各行における数値の相対比で，下線部は行ごとの最速値である．

も，ごく小さい密行列の積の場合を除き，MATLAB インタプリタによる実行よりも Fortran 90 への変換後のコードの方が高速であることが分かる．CMC では行列の積を格納する代入先の疎行列のための記憶領域の大きさをユーザが指定しなくてはならないが，動的な領域確保や添字の境界チェックなどのオーバヘッドの削減により，高速実行が行われる．

なお，ほとんど自明であるが，表 2 (b) の実測を密行列データ構造にしか対応していない環境で行うことは所要記憶用量および計算量のいずれの観点からも非現実的である．

8.2 SOR 法の実測結果

連立一次方程式 $Fx = v$ および $Sx = u$ を SOR 法により解いた．用いたコードは図 12 である．初期解はゼロとし，残差ノルムの比が $tol = 10^{-6}$ 以下になった時点で反復を終了した．加速係数は 1.8 とした．

実測結果を**表 3** および**表 4** に示す．MATLAB インタプリタでは 2 通り，CMC では 4 通りの実行方法で実測した．表中，dense および sparse は，係数行列を保持するデータ構造を密行列（二次元配列），疎行列（CCS 形式）のいずれにしたかを表す．CMC についてはさらに，対角行列や三角行列の形状を検出して出力コードに反映させた場合（shape）としなかった場合（noshape）について実測した．なお，CMC/dense+noshape で得られるコードは FALCON[5] の出力コードと実質的に同等であると考えられる．

8.2.1 問題 $Sx = u$ の場合（表 3）

係数行列が疎の場合，dense はいずれも低速である．MATLAB か否かによらず，疎行列を密行列データ構造で扱うのは効率が悪いということが確認できる．

MCC を適用しての実測は疎行列データ構造の場合について行ったが，表 3 からは MATLAB/sparse の場合との有意な差異は認められない．

CMC/sparse+shape は MATLAB/sparse より 7 倍以上高速である．CMC/sparse の noshape と shape との違いは，図 12 における対角行列 `D` を一次元配列で扱うか CCS 形式で扱うかの違いにすぎないため，実行時間に大きな差はない．

8.2.2 問題 $Fx = v$ の場合（表 4）

sparse はいずれも低速である．MATLAB では特に，密行列を疎行列データ構造で扱う際のオーバヘッドが顕著である．

MCC を適用しての実測は密行列データ構造の場合について行ったが，MATLAB/sparse に対して高速化できてはいない．なお，F の次元数が 400 の場合の大幅な速度低下の原因の詳細は不明である．

CMC/dense+shape は CMC/dense+noshape よりも 1.4 倍程度高速である．CMC/dense における shape と noshape の差は，問題 $Sx = u$ の場合の CMC/sparse における shape と noshape の差よりも大きい．密行列計算では三角行列などの形状情報の把握の有無が実行速度に与える影響が大きいといえる．

8.3 CG 法の実測結果

CG 法により方程式 $Sx = u$ と $Fx = v$ を解いた結果を**表 5** に示す．用いたコードは図 13 (b) である．反復の停止基準や初期解は SOR 法の場合と同じにし

表 6 ソースレベル最適化適用の効果
Table 6 The effects of source-level optimization on the SOR and CG method.

(a) the SOR method

	MATLAB [sec]		MCC [sec]		CMC [sec]	
次元数/反復回数	noopt	opt	noopt	opt	noopt	opt
400/64	0.15(1.0)	0.08(0.53)	0.14(0.93)	0.07(0.47)	0.0272(0.18)	0.00932(0.062)
900/81	0.41(1.0)	0.22(0.54)	0.40(0.98)	0.20(0.49)	0.0902(0.22)	0.0286(0.070)
1600/158	1.46(1.0)	0.77(0.53)	1.47(1.0)	0.74(0.51)	0.319(0.22)	0.0980(0.067)
2500/247	3.65(1.0)	1.87(0.51)	3.66(1.0)	1.87(0.51)	0.878(0.24)	0.252(0.069)
4900/472	14.0(1.0)	7.27(0.52)	14.5(1.0)	7.22(0.52)	3.53(0.25)	1.01(0.072)
8100/755	37.9(1.0)	19.9(0.53)	37.2(0.98)	19.4(0.51)	10.3(0.27)	2.68(0.071)

opt の欄の数値はそれぞれ表 3 の MATLAB/sparse，MCC/sparse，CMC/sparse+shape の欄の再掲である．
括弧内の数値は各行における数値の相対比で，下線部は行ごとの最速値である．

(b) the CG method

	MATLAB [sec]		MCC [sec]		CMC [sec]	
次元数/反復回数	noopt	opt	noopt	opt	noopt	opt
900/72	0.13(1.0)	0.04(0.31)	0.13(1.0)	0.04(0.31)	0.0221(0.17)	0.00758(0.058)
2500/117	0.62(1.0)	0.20(0.32)	0.64(1.0)	0.20(0.32)	0.142(0.23)	0.0501(0.081)
4900/162	1.66(1.0)	0.55(0.33)	1.75(1.1)	0.56(0.34)	0.406(0.24)	0.162(0.098)
8100/205	3.58(1.0)	1.19(0.33)	3.59(1.0)	1.17(0.33)	0.880(0.25)	0.385(0.11)
12100/249	6.66(1.0)	2.31(0.35)	6.51(0.98)	2.19(0.33)	1.66(0.25)	0.716(0.11)

opt の欄の数値は表 5 (a) の再掲．括弧内の数値は各行における数値の相対比で，下線部は行ごとの最速値である．

表 5 CG 法の実測結果
Table 5 Experimental results on the CG method.

(a) $Sx = u$ (S: sparse matrix)

	MATLAB	MCC	CMC
次元数/反復	sparse [sec]	sparse [sec]	sparse [sec]
900/72	0.04(1.0)	0.04(1.0)	0.00758(0.19)
2500/117	0.20(1.0)	0.20(1.0)	0.0501(0.25)
4900/162	0.55(1.0)	0.56(1.0)	0.162(0.29)
8100/205	1.19(1.0)	1.17(0.98)	0.385(0.32)
12100/249	2.31(1.0)	2.19(0.95)	0.716(0.31)

(b) $Fx = v$ (F: dense matrix)

	MATLAB	MCC	CMC
次元数/反復	dense [sec]	dense [sec]	dense [sec]
200/112	0.06(1.0)	0.07(1.2)	0.0396(0.66)
400/214	0.41(1.0)	0.41(1.0)	0.288(0.70)
600/316	1.33(1.0)	1.34(1.0)	0.966(0.73)
800/417	3.28(1.0)	3.11(0.95)	2.26(0.69)

た．表 5 中の括弧内の数値は MATLAB における所要時間との比を示している．なお，CG 法のコードでは shape か noshape かによる違いは現れない．また CMC/dense で得られるコードは FALCON [5] の出力コードと実質的に同等であると考えられる．

CG 法は基本的な行列ベクトル演算から構成されるので MATLAB に都合のよいアプリケーションの 1 つであるが，表 5 (a) では CMC は MATLAB よりも 3 倍以上高速である．

MATLAB に対する CMC の高速化率は，係数行列が密行列の場合（表 5 (b)）よりも疎行列の場合（表 5 (a)）のほうが大きい．MATLAB における疎行列データ構造の扱いにともなうオーバヘッドの大きさが現れているといえる．

なお，表 5 には MCC での実行結果と MATLAB の場合との違いはほとんど現れていない．

8.4 ソースレベル最適化の効果について

CMC は，図 5 のコードを図 12 に，また図 13 (a) のコードを図 13 (b) に，MATLAB ソースコードレベルで変換する機能を持つ．ここではこのソースコードレベルの最適化機能の評価のため，最適化前と最適化後のそれぞれのソースコードによる連立一次方程式 $Sx = u$ の求解の実測結果を**表 6** に示す．noopt は図 5 あるいは図 13 (a) のコード，opt は図 12 あるいは図 13 (b) のコードの実行であることを示す．CMC での実行結果は形状情報をコード出力に反映させたもの（表 3 における shape+sparse）である．CMC/noopt の測定では CMC のソースコードレベル最適化機能はオフにした．なお opt の欄の値は表 3 および表 5 (a) のデータの再掲である．CMC は表中の noopt のコードを opt のコードに変換できるが，MATLAB 処理系は，MCC を含め，この類の最適化は行わない．

表 6 より，CMC の持つソースレベル最適化機能が実行速度に大きな影響を与えることが分かる．MATLAB/noopt に対して MATLAB/opt は，SOR 法，CG 法ともに 2 倍程度高速である．すなわちソースコードレベルでの最適化だけで大幅に高速化できている．また MATLAB/noopt と CMC/opt との比較から，ナイーブなアルゴリズム記述に対して CMC は

MATLAB 処理系よりも 10 倍前後高速化できる場合があるということも分かる．

9. 関 連 研 究

MATLAB コードの高速化に関しては，1 章でも述べたとおり，いくつかのアプローチがある[5)~7),9)]．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[14)]．

MaTX[3)] など，高速性を重要視して最初からコンパイル言語として設計されている行列言語もあるが，疎行列データ構造に対応したものは見られない．また既存のコンパイル型言語では，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，MATLAB などと比較するとプログラム記述は繁雑になるので，可読性や可搬性の高いコード開発は必ずしも容易ではない．

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はあるが[15),16)]，ユーザが Fortran 90 でコードを記述することを前提としており，行列言語に対する取り組みではない．Fortran 90 における “行列計算” は演算子の整合性に難があり，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．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．

疎行列計算コード記述を支援するシステムという観点から CMC に関連するものとしては，関数型言語記述から疎行列コードを生成する試みがあげられる[17)]．また，Fortran 77 で記述された密行列用コードを元にして疎行列コードを生成する研究もある[18),19)]．これらはいずれも疎行列コード記述の繁雑さを低減することが主眼であるが，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 の密行列コード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．

高性能な疎行列コード開発支援のために，疎行列演算をライブラリルーチン化する手法も検討されている[12)]．これは疎行列コードの性能だけに注目すれば有効なアプローチである．しかしながら，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．CMC は，文献 6) や 7) の方法と同様，シンプルなコード記述の可能な行列言語ベースのアプローチであることをその存在意義の 1 つとして主張するものであるが，5.3 節で述べたように CMC もライブラリルーチンを呼び出す形式の出力コード生成は可能であり，ライブラリベースの手法と対立するものではない．なお，文献 6) や 7) で述べられているのは ScaLAPACK を用いて高速化する方法であるが，疎行列データ構造の扱いは考慮されていない．CMC はすでに CCS 形式による疎行列データ構造を扱う機能を有するので，将来的に疎行列プリミティブが標準化された場合にはそれに対応したコード生成を行うように拡張することも容易であると考えられる．

10. ま と め

本論文では，MATLAB で記述された行列計算コードを Fortran 90 に変換して高速化する処理系 CMC の設計と実現方式について述べた．また，CMC の性能評価を実測に基づいて行った．結論として次がいえる：

- 行列言語を用いて疎行列を扱う高速な数値計算コードの生成が可能であることが確認できた．MATLAB インタプリタに対し，CMC によって SOR 法で 7 倍以上，CG 法で 3 倍以上，高速化できた．
- 行列の形状を詳細に把握することにより，密行列コードも高速化できることが分かった．SOR 法における実測では，MATLAB インタプリタでの実行に対し，CMC により 1.4 倍高速化できた．
- CMC のソースレベルでの最適化機能の有効性が確認できた．SOR 法や CG 法のナイーブなコード記述に対する実測では，MATLAB インタプリタでの実行に対し，CMC では 10 倍前後高速化できた．

今後の課題としては，MATLAB の組み込み関数への対応などによる MATLAB との互換性のさらなる向上，CCS 形式以外の疎行列データ構造への対応，外部の行列計算ライブラリとの連携機能の強化，などがあげられる．

謝辞 本研究の遂行にあたり，京都大学および広島市立大学津田孝夫名誉教授には，有益なご助言を賜った．なお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）の助成による．

参 考 文 献

1) http://www.mathworks.com/
2) http://www.scilab.org/
3) http://www.matx.org/
4) Doornik, J.A.: *Ox: Object-Oriented Matrix Language*, 4th edition, Timberlake Consultants Press (2001).
5) 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).
6) Ramaswamy, S., et al.: Compiling MATLAB Programsto ScaLAPACK: Exploiting Task and Data Parallelism, *Proc. IPPS'96*, pp.613–619 (1996).
7) 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).
8) The MathWorks, Inc.: *MATLAB Compiler Version 3 User's Guide* (2002).
9) Menon, V. and Pingali, K.: A Case for Source-Level Transformations in MATLAB, *Proc. DSL'99*, pp.53–65 (1999).
10) 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).
11) Barrett, R., et al.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM (1994).
12) 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).
13) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers — Principles, Techniques, and Tools*, Addison Wesley (1986).
14) Almasi, G. and Padua, D.: MaJIC: Compiling MATLAB for Speed and Responsiveness, *Proc. PLDI'02*, pp.294–303 (2002).
15) 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).
16) 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).
17) 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).
18) Bik, A.J.C. and Wijshoff, H.A.G.: Compilation Techniques for Sparse Matrix Computations, *Proc. ICS'93*, pp.416–424 (1993).
19) Bik, A.J.C., et al.: The Automatic Generation of Sparse Primitives, *ACM Trans. Math. Softw.*, Vol.24, No.2, pp.190–225 (1998).

(平成 16 年 1 月 31 日受付)
(平成 16 年 5 月 29 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．高性能計算，自動並列化技術，数値処理ソフトウェアに関する研究に従事．ACM，IEEE-CS，SIAM 各会員．

鈴木 睦

2004 年広島市立大学情報科学部情報工学科卒業．現在日立エスケイソーシャルシステム株式会社勤務．大規模データベースシステムの開発に従事．