社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－ARC－157 (19)
2004－HPC－ 97 (19)
2004／3／2

MATLAB記述のコンパイレーションにおける複数種類の疎行列データ構造への対応

川端 英之† 鈴木 睦†

MATLAB は数値計算コードを簡潔に記述できる言語および実行環境であり，広く利用されている．我々は，MATLAB コードの高速な実行環境の提供のため，MATLAB コードを静的解析により Fortran 90 記述に変換するコンパイラ CMC を開発している．CMC は疎行列計算にも対応しているが，これまでの実装では疎行列のデータ構造は CCS 形式しか扱えなかった．本稿では，CMC の CRS 形式や Multi-diagonal 形式によるデータ構造への対応について述べる．実測により，CMC にあらたに実装したデータ構造の自動変換機能が有効であることが分かった．

Compilation of MATLAB Scripts Utilizing Various Data Structures for Sparse Matrices

HIDEYUKI KAWABATA† and MUTSUMI SUZUKI †

MATLAB is a language and an execution environment for matrix computations, which is used in wide area. We have been developing the CMC, a compiler for matrix computations, which translates MATLAB-based scripts into Fortran 90 programs by static analyses. The CMC didn't have functionality for data structures for sparse matrices other than CCS form until the extension we show in this article was implemented. Newly supported data structures include CRS and Multi-diagonal forms. Experimental results show the importance to choose right data structures to make the most use of the computer at hand. In addition, automatic transformation of data structures is proved to be useful.

1. はじめに

MATLAB は数値計算コードを簡潔に記述できる言語および実行環境である[2]．MATLAB は，行列演算をプリミティブとして持つこと，変数の型宣言が不要なこと，インタプリタ実行に基づき記憶管理が動的になされること，などの特徴を持っており，プロトタイピング言語として有効で，広く利用されている．行列積などの基本的な行列演算はチューニングされたライブラリによって処理されるので，簡単な行列計算コードであればインタプリタ環境での実行であっても比較的高速に実行でき，実用性も高い．

様々な MATLAB 記述が高速に実行できればよいのだが，MATLAB 実行環境は動的な型やデータ構造の変更を行なうので実行時のオーバヘッドは無視できない．これに対し，変数の型などを静的解析により決定して MATLAB 記述を Fortran などのコンパイル言語記述に変換し高速化を図る試みがあるが[3]，大規模数値計算に用いるためには不可欠な，疎行列データ構造への対応が行なわれたものは見られなかった．

これに対し我々は疎行列データ構造を扱える行列言語コンパイラ CMC☆を開発している[1]．CMC は次の機能を持つ：

- MATLAB 記述を Fortran90 記述に変換する．
- 疎行列データ構造も利用可能である．
- 行列の形状情報をコード生成に利用する．
- ユーザ指示は必要に応じて MATLAB のコメント形式の指示行を用いる．

CMC を用いれば，繁雑になりがちな疎行列計算コードの開発を，汎用言語記述よりも保守性や可読性の高い行列言語記述で行なうことが可能になる．

我々は更に，複数のデータ構造を扱う機能を CMC に実装した．本稿では，CMC であらたに可能となった複数のデータ構造を扱う機能についてて述べる．

2. MATLAB およびその高速化について

2.1 MATLAB コードの概要

MATLAB コードにおける変数は行列を基本として

† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University

☆ Compiler for Matrix Computations

```
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.

いる．MATLAB では行列の積や転置などの操作が基本演算であり，数値計算コードの記述が簡潔に行なえる．MATLAB コードの例を図 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 言語により関数として記述したものである．両者の類似性は明らかである．

図 1(b) のように，MATLAB では変数の型宣言などは不要で，各演算子による計算処理内容はオペランドの型や形状 (ベクトルか行列かなど) に基づき動的に決定される．実際，図 1(b) の関数を引数の A や x にスカラ値を与えて呼び出し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が，行列計算コードの記述においては各行列が疎行列か否かを意識する必要はない．例えば図 1(b) の関数は仮引数の行列 A に対応する実引数を疎行列として呼び出しても，動的な対処により適切に実行される．

2.2 CMC による MATLAB コードの高速実行

MATLAB では各演算子による処理内容がオペランドの変数の型や形状，あるいはデータ構造に応じて動的に決められる．このインタプリタ実行に起因するオーバヘッドを削減するべく，Fortran90 などのコンパイル言語記述に変換して実行する方法の研究はあったが [3]，疎行列に対応したものはなかった．これに対し我々は，疎行列データ構造をユーザが意識すること無く行列演算を記述できることができる処理系，CMC

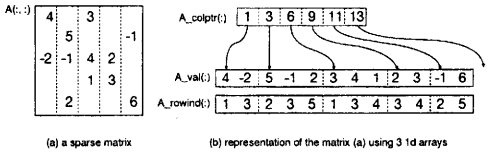

図 2 CCS 形式による疎行列の表現
Fig. 2 The structure of CCS form for sparse matrices.

を開発した [1]．CMC は，三角行列や対角行列などの行列の形状の詳細情報もプログラム記述から自動抽出し，それに応じた出力コードを生成できる [1]．

3. CMC における複数種類の疎行列データ構造への対応

疎行列のためのデータ構造は多数考えられる．データ構造はアルゴリズムの効率と密接に関連するので目的に応じてデータ構造を使い分けることが望ましい．

MATLAB は，「各行列演算が疎行列の非零要素数におよそ比例する計算時間および所要記憶容量で抑えられればよい」との考え方で [4]，単一のデータ構造にしか対応していない．我々は，より高速に実行できる行列言語処理系の開発を目指し，いくつかの試行を行なった．結論として，次の機能を CMC に実装した：

- 疎行列データ構造として，CCS 形式に加え，CRS 形式，MD 形式に対応．
- 行列演算は，同じデータ構造どうしのみで行なう．
- 異なるデータ構造どうしの演算がユーザから要求される場合は，一方のデータ構造を変換して同一構造に揃えてから計算する．

本章では，典型的な疎行列データ構造について触れ，CMC で実装した疎行列への対応について述べる．

3.1 疎行列データ構造について

基本的な疎行列データ構造の一つに，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 Compressed Column Storage (CCS) 形式 [5] がある．これは Compressed Sparse Column 形式 [6] とも呼ばれ，図 2 に示すように 3 本の一次元配列で一つの行列を表す．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．MATLAB における疎行列のデータ構造は，CCS 形式と同等である [4]．なお CCS 形式における行と列を入れ換えた形式は Compressed Row Storage (CRS) 形式 [5] あるいは Compressed Sparse Row 形式 [6] と呼ばれ，所要記憶容量などの特性は CCS 形式と同等である．

帯行列は Compressed Diagonal Storage (CDS) で扱われることも多い．CDS 形式 [5] は図 3 のように行列の要素の位置を規則的にずらして二次元配列に格納するので，要素参照に間接参照を用いる必要がない．しかし，帯内にある零要素は保持しなくてはならず，いくつかの無駄な (使用されない) 領域も生じるので，

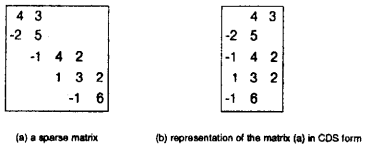

図 3 CDS 形式による疎行列の表現
Fig. 3 The structure of CDS form for sparse matrices.

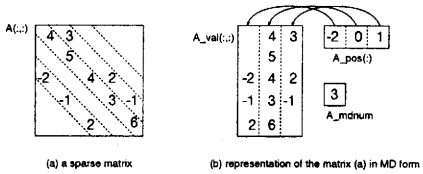

図 4 Multi-diagonal 形式による疎行列の表現
Fig. 4 The structure of multi-diagonal form for sparse matrices.

```
y = 0
do Xj = 1, 2500
  do Xiptr = A_colptr(Xj), A_colptr(Xj+1)-1
    Xi = A_rowind(Xiptr)
    y(Xi) = y(Xi) + A_val(Xiptr) * x(Xj)
  enddo
enddo
```

(a) CCS

```
do Xi = 1, 2500
  Xs = 0
  do Xjptr = A_rowptr(Xi), A_rowptr(Xi+1)-1
    Xj = A_colind(Xjptr)
    Xs = Xs + A_val(Xjptr) * x(Xj)
  enddo
  y(Xi) = Xs
enddo
```

(b) CRS

```
do Xk = 1, A_mdnum
  Xj = A_pos(Xk)
  do Xi = max(1,Xj), 2500+min(0,Xj)
    y(Xi-Xj) = y(Xi-Xj) + A_val(Xi-Xj,Xk) * x(Xi)
  enddo
enddo
```

(c) MD

図 5 y=A*x に対応する Fortran 90 記述
Fig. 5 Fortran 90 code corresponding to y=A*x.

帯幅が狭い場合でなければ効率が悪い．

CDS 形式では帯内に全ての要素がゼロの sub-/superdiagonals があってもその事実を記憶領域削減に利用できないが，図 4 のようにすれば可能となる；ここでは多重対角（MD; Multi-Diagonal）形式と呼ぶ．MD 形式では，部分的に間接参照が必要となるが，偏微分方程式の自然な変数順序付けによる差分近似などで頻出の疎行列を効率良く扱える．

3.2 疎行列構造を用いた演算：行列とベクトルの積

行列 A を疎行列とし，x, y を列ベクトルとしたときに，MATLAB における行列計算記述 y=A*x が CMC によって Fortran 90 記述に変換された様子を図 5 に

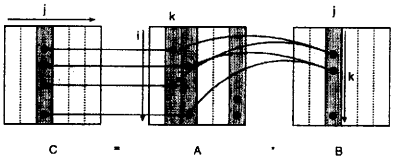

図 6 CCS 形式の行列の乗算 (C=A*B)
Fig. 6 Sparse matrix multiplication using CCS form.(C=A*B)

示す．図 5(a)，(b)，(c) はそれぞれ行列 A が CCS 形式，CRS 形式，MD 形式の場合である．CCS 形式と CRS 形式では間接参照が必要である．一方 MD 形式では非零要素以外の要素についても計算が必要である．

CCS 形式と CRS 形式の違いは内側ループ中で回帰的参照が行なわれるか否かである．ベクトル計算機では CCS 形式が有利と考えられるが，スカラ計算機上ではキャッシュ容量やストアの回数から考えて CRS 形式が有利である．

図 5(c) の MD 形式によるコードは，非零要素以外の格納量が少ない場合には，間接参照が全く無い上に最内側ループの繰返し回数が多く，ベクトル計算機に最も適する．スカラ計算機も有効である．

3.3 疎行列構造を用いた演算：行列の和

CMC では，CCS 形式どうし，CRS 形式どうし，あるいは MD 形式どうしの和は，特に指定しない限りそれぞれ同じ形式で格納する．

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程度の加算である．ただし CCS（CRS）形式の場合には値の格納の際に列（行）方向に詰める必要がある．CMC では，加えるべき列（行）どうしの非零要素を同時に順に調べて詰めながら加算する．

3.4 疎行列構造を用いた演算：行列積

行列積 C=A*B を，CCS 形式どうし，CRS 形式どうし，MD 形式 どうしで行なう場合について述べる．なお，CMC では，同じ構造の疎行列どうしの積は特に指定しない限りそのデータ構造に格納する．

まず CCS 形式どうしの積における行列要素の参照の様子を図 6 に示す．CCS 形式への格納は第 1 列から最終列まで順に行な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．オペランドの両行列が次元数 n の正方行列であるとし，列当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，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しがちである．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ので，CMC では各列の計算後に列方向の圧縮を行なうアルゴリズムとして次の方法を用意して選択できるようにしている：

(1) 非零要素の位置をフラグで把握し，列／行毎に値を詰める方法

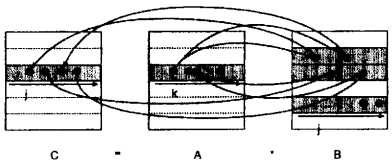

図 7 CRS 形式の行列の乗算 (C=A*B)
Fig. 7 Sparse matrix multiplication using CRS form.(C=A*B)

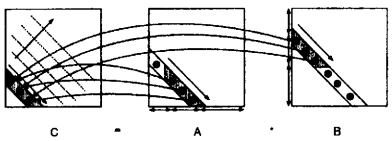

図 8 MD 形式の行列の乗算 (C=A*B)
Fig. 8 Sparse matrix multiplication using MD form.(C=A*B)

```
allocate(Xflag(dim2))
Xflag = 0
do Xj = 1, dim2
  do Xiptr = A_colptr(Xj), A_colptr(Xj+1)-1
    Xi = A_rowind(Xiptr)
    Xflag(Xi) = Xflag(Xi) + 1
  enddo
enddo
B_rowptr(1) = 1
do Xj = 1, dim2
  B_rowptr(Xj+1) = B_rowptr(Xj) + Xflag(Xj)
  Xflag(Xj) = B_rowptr(Xj)
enddo
do Xj = 1, dim2
  do Xiptr = A_colptr(Xj), A_colptr(Xj+1)-1
    Xi = A_rowind(Xiptr)
    B_val(Xflag(Xi)) = A_val(Xiptr)
    B_colind(Xflag(Xi)) = Xj
    Xflag(Xi) = Xflag(Xi) + 1
  enddo
enddo
deallocate(Xflag)
```

図 9 CCS 形式から CRS 形式へのデータ構造変換
Fig. 9 Transformation of sparse matrix structures.

(2) 非零要素の出現順に配列に詰めておいてからソートする方法

CRS 形式どうしの行列積の要素参照の様子を図 7 に示す．CRS 形式と CCS 形式は互いに転置なので CCS 形式の場合とほとんど同様である．

MD 形式どうしの場合の要素参照は図 8 のように対角方向に行なえば間接参照を排除できる．非零要素の割合が高い場合には，ベクトル計算機ではもちろん，スカラ計算機においても高速な実行が期待できる．

3.5 疎行列データ構造の変換

3.5.1 CCS 形式と CRS 形式

CCS 形式から CRS 形式への構造変換は図 9 のようにすればよい．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，CRS 形式から CCS 形式へのデータ構造変換も全く同様に行なうことができる．

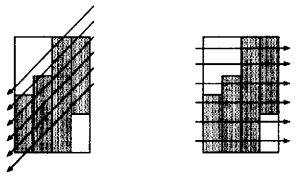

図 10 MD 形式から CCS 形式，CRS 形式への変換
Fig. 10 Transformation from MD form to CCS or CRS form.

3.5.2 CCS 形式，CRS 形式から MD 形式へ

CCS 形式から MD 形式への構造変換手順を示す：

(1) 行列の非零要素を全て調べ，MD 形式にしたときに「何列分」の領域が必要になるかを調べる．
(2) 何本目の対角方向の要素を何列目に格納すればよいかを決定する．
(3) 行列の全ての要素について，一つずつ，データをコピーする．必要に応じて零も格納する．

非零要素数にほぼ比例した回数の処理が必要である．CRS 形式の場合も同様である．

3.5.3 MD 形式から CCS 形式，CRS 形式へ

MD 形式で格納された行列を CCS 形式あるいは CRS 形式に変換するのは容易である．図 10 に矢印で示した方向に疎って非零要素をスキャンしながら，直接的に要素をコピーすることができる．

3.6 異なるデータ構造どうしの演算について

異なるデータ構造どうしの演算は，同一のデータ構造どうしの演算と比較して，効率が悪いと思われる．例えば CRS 形式と CCS 形式の乗算は内積形式のアルゴリズムが採用できるが，非零要素どうしの積を最内側ループ中で検索する必要が生じる．MD 形式も，図 4 のデータ構造のままでは，「i 行 j 列の要素」を検索無しで参照することはできない☆．疎行列データ構造は，要素の配置順にアクセスすることができるようなアルゴリズムと組にして使用することが重要である．

なお，4.3 節で述べたように，データ構造の変換はいずれの形式でも非零要素数におよそ比例するコストで行なえる．そこで CMC は，両者のデータ構造を統一してから行列演算を行なうことにする．

4. 実測と評価

CMC に対する複数のデータ構造への対応の効果について評価を行なうために，いくつかの実測結果を示す．用いた計算機やソフトウェア環境は次の通り☆☆：

☆ CCS における colptr のように，列番号を保持するポインタを別の一次元配列に持たせれば検索は不要になる．ただし n 行 m 列の行列に対して要素数は $m+n-1$ 個必要である．

☆☆ 京都大学学術情報メディアセンターのシステムを使用．

表 1 SPP 上での行列積
Table 1 Matrix multiplication on SPP.

(a) 5 点差分 S の場合

次元数	MATLAB[sec]	CCS 形式 [sec]	MD 形式 [sec]
10000	0.06(1.7)	0.0356(1.0)	0.0137(.38)
40000	0.28(2.0)	0.141(1.0)	0.0540(.38)
90000	0.75(2.3)	0.320(1.0)	0.138(.43)
160000	1.25(2.1)	0.587(1.0)	0.264(.45)
250000	2.10(2.2)	0.944(1.0)	0.428(.45)

(b) 7 点差分 T の場合

次元数	MATLAB[sec]	CCS 形式 [sec]	MD 形式 [sec]
1000	0.01(2.4)	0.00410(1.0)	0.00189(.46)
8000	0.12(1.6)	0.0743(1.0)	0.0200(.27)
27000	0.42(1.6)	0.262(1.0)	0.0690(.26)
64000	1.01(1.6)	0.628(1.0)	0.207(.33)
125000	2.06(1.7)	1.23(1.0)	0.408(.33)

表 2 VPP 上での行列積
Table 2 Matrix multiplication on VPP.

(a) 5 点差分 S の場合

次元数	CCS 形式 [sec]	MD 形式 [sec]
10000	0.222(1.0)	0.000318(.0014)
40000	0.900(1.0)	0.00101(.0011)
90000	2.03(1.0)	0.00214(.0011)
160000	3.61(1.0)	0.00389(.0011)
250000	5.65(1.0)	0.00576(.0010)

(b) 7 点差分 T の場合

次元数	CCS 形式 [sec]	MD 形式 [sec]
1000	0.00699(1.0)	0.0000214(.0031)
8000	0.276(1.0)	0.000519(.0019)
27000	0.960(1.0)	0.00138(.0014)
64000	2.31(1.0)	0.00303(.0013)
125000	4.55(1.0)	0.00582(.0013)

(1) スカラ計算機（以降では SPP と呼ぶ）
- 富士通 GP7000F (SPARC64-GP, 主記憶 24GB) を 1 CPU のみ使用.
- Fortran90 コンパイラ：`frt` (Fujitsu Fortran Compiler Driver Version 5.5)
- `frt` オプション：`-Kfast_GP=2 -X9`

(2) ベクトル計算機（以降，VPP と呼ぶ）
- 富士通 VPP800/63（ピーク性能 8GFlops）を 1 ノードのみ使用.
- Fortran90 コンパイラ：`frt` (Fujitsu UXP/V Fortran V20L20 Driver Version L02091)
- `frt` オプション：`-Wv,-m3 -Pao -Eiuep -X9`

なお SPP 上での SOR 法，CG 法の実測については MATLAB （Ver6.5.1）での実行結果も併せて示す.

計時は，MATLAB では組み込み関数 `cputime` を利用した．Fortran90 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.

4.1 行 列 積

Laplace 方程式

```
function [x,i] = sor0s(A,x0,b,tol)
%CMC integer,parameter :: s=50*50
%CMC real*8,_CCS(5) :: A(s,s)
%CMC real*8,_COLVEC :: x0(s), b(s)
%CMC real*8,_SCALAR :: tol
w = 1.8;
D = diag(diag(A)); L = -tril(A,-1); U = -triu(A,1);
r0 = norm(b-A*x0);
M = D-w*L; N = (1-w)*D+w*U; b2 = w*b;
x = x0; i = 0;
while 1
  i = i+1;
  x = M\(N*x+b2);
  if norm(b-A*x)/r0 <= tol, break, end
end
```

図 11 MATLAB で記述した SOR 法のコード
Fig. 11 The SOR method in MATLAB.

$$\nabla^2 u = 0 \qquad (1)$$

について，二次元正方格子上の自然な順序付けの 5 点差分近似を行なう際に現れる係数行列 S，および，三次元の場合の 7 点差分近似を行なう際に現れる係数行列 T を用いて，S^2 および T^2 を計算した．なお，行列 S や T は，MD 形式に適した非零要素配置である.

行列積の実測にあたっては，計算後のデータ圧縮は 3.4 節で述べた選択肢のうち (2) を用いた.

SPP 上での実測結果を**表 1** に示す．表中の括弧内の数値は，各行の実測値の相対的な大きさを示している．MD 形式を用いると，CCS 形式よりも，行列 S では 2.2~2.6 倍，T では 2.2~3.8 倍，高速化されることが分かる．なお，MATLAB インタプリタでの実行と比較すると 5 倍程度高速である.

表 2 は VPP 上での行列積の実行結果である．MD 形式を用いた場合は CCS 形式を用いるよりも 100 倍近く高速に実行できることが分かる.

4.2 連立一次方程式の求解

式 (1) について，二次元正方領域上でディリクレ境界条件の下で 5 点差分近似を行なった結果を**表 3**，**表 4**，**表 5** に示す．求解には SOR 法（図 11）と CG 法を用いた．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．SOR 法における加速係数は 1.8 とした.

SPP 上での結果（表 3, 表 4）を見ると，疎行列（係数行列）のデータ構造は CCS 形式よりも CRS 形式のほうが高速である．これは行列とベクトルの積の最内側ループにおいてストア回数が大幅に減ることに依ると思われる．また CRS 形式よりも MD 形式のほうが高速であるが，これは MD 形式のデータ参照が直接参照であるためであろう．なお，いずれのデータ構造の場合も，MATLAB より数倍高速である.

表 5 は VPP 上での結果である．CRS 形式での CG 法では，最も重い処理である行列と列ベクトルの積がベクトル計算機に向かないデータ参照パターンとなるので，VPP 上では CRS 形式では CCS 形式の場合よりも遅い．一方，MD 形式では，ループ回転数の多い

表 3 SPP 上での SOR 法の実測結果
Table 3 Results of SOR solver execution on SPP.

次元数/反復回数	MATLAB[sec]	CCS 形式 [sec]	CRS 形式 [sec]	MD 形式 [sec]	CCS to CRS [sec]	CCS to MD [sec]
1600/158	0.78(7.6)	0.103(1.0)	0.0915(.89)	0.0561(.54)	0.0832(.81)	0.0584(.57)
6400/606	12.3(7.1)	1.74(1.0)	1.53(.88)	1.08(.62)	1.53(.88)	1.08(.62)
14400/1282	61.9(7.1)	8.72(1.0)	7.44(.85)	5.42(.62)	7.49(.86)	5.43(.62)
22500/1926	168(7.6)	22.2(1.0)	19.9(.90)	12.3(.55)	20.1(.91)	12.8(.58)

表 4 SPP 上での CG 法の実測結果
Table 4 Results of CG solver execution on SPP.

次元数/反復回数	MATLAB[sec]	CCS 形式 [sec]	CRS 形式 [sec]	MD 形式 [sec]	CCS to CRS [sec]	CCS to MD [sec]
1600/95	0.18(3.9)	0.0460(1.0)	0.0448(.97)	0.0316(.69)	0.0464(1.0)	0.0338(.73)
6400/184	1.52(3.4)	0.452(1.0)	0.427(.94)	0.304(.67)	0.420(.93)	0.300(.66)
14400/270	4.86(3.1)	1.59(1.0)	1.46(.92)	1.10(.69)	1.47(.92)	1.11(.70)
22500/334	9.74(3.1)	3.17(1.0)	2.83(.89)	2.17(.68)	2.97(.94)	2.20(.69)

表 5 VPP 上での CG 法の実測結果
Table 5 Results of CG solver execution on VPP.

次元数/反復回数	CCS 形式 [sec]	CRS 形式 [sec]	MD 形式 [sec]	CCS to MD [sec]
1600/95	0.127(1.0)	0.223(1.8)	0.00372(.029)	0.00549(.043)
6400/184	0.975(1.0)	1.71(1.8)	0.0215(.022)	0.0287(.029)
14400/270	3.22(1.0)	5.65(1.8)	0.0649(.020)	0.0815(.025)
40000/437	15.2(1.0)	25.4(1.7)	0.276(.018)	0.322(.021)

最内側ループが間接参照もなくベクトル化できるため，ベクトル計算機の能力を十分に引き出せている．

4.3 データ構造の変換機能について

データ構造の選択が計算の実行速度に大きな影響を与えることが前節の実測結果で示されているが，ユーザが手元に用意しているデータやリンクしようとしているプログラムは必ずしも CRS 形式や MD 形式であるとは限らない．そこで，CMC を拡張し，CCS 形式で受けた疎行列を節で述べた処理により CRS 形式あるいは MD 形式に変換して計算を続けるようなコード出力機能を加えた．この機能に基づいた実測結果を，表 3，表 4，表 5 の "CCS to CRS" あるいは "CCS to MD" の欄に示す．実測結果の数値は，データ構造の変換に要する時間を含んだものである．

SPP 上では，CCS 形式のままで実行するよりも，CRS 形式に変換して計算することで 1 割前後の高速化につながっている．また，MD 形式に変換した場合は 5 割 ~7 割高速化できている．VPP 上での実行では，CCS 形式での実行よりも 5 倍近く高速である．表 3，表 4，表 5 の結果は，用いる計算機やアルゴリズムの特性に応じてデータ構造の選択を適切に行なうことの重要性を示している．

5. ま と め

本稿では，行列言語コンパイラ CMC の複数の疎行列データ構造への対応について述べた．CCS 形式に加え，CRS 形式および MD 形式に対応することにより，Laplace 方程式の差分近似などで得られる疎行列をより効率的に扱うことができることが分かった．また，各々の疎行列データ構造間の変換機能を CMC に実装することにより，プログラムに入力する疎行列のデータ構造をユーザが変更することなく使用する計算機やアルゴリズムの特性を生かした実行を可能にした．

今後の課題としては，より多くのアプリケーションに対する CMC の適用と性能評価が挙げられる．

謝辞 本研究の遂行にあたり，広島市立大学津田孝夫名誉教授に有益な御助言を賜った．

参 考 文 献

1) 川端英之, 鈴木睦: MATLAB に基づく疎行列計算向けコード生成, 情報処理学会ハイパフォーマンスコンピューティング研究会研究報告, 2003-HPC-96 (2003).
2) Mathworks Inc. homepage. `www.mathworks.com`.
3) De Rose, L., Padua, D.: Techniques for the translation of MATLAB programs into Fortran 90, *ACM Trans. Programming Languages and Systems*, Vol.21, No.2, pp.286–323 (1999).
4) Gilbert, J. R., Moler, C., Schreiber, R.: Sparse Matrices in MATLAB: Design and Implementation, *SIAM J. Matrix Anal. Appl.*, Vol. 13, No. 1, pp.333–356 (1992).
5) Barrett, R., et al.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM (1994).
6) 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).