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

2006－HPC－108 (4)
2006／10／5

MATLABプログラムの実行時特殊化機能の実現

網本 真夏美† 川端 英之† 北村 俊明†

MATLABは動的に型付けされる言語である．プログラム記述の際に変数宣言が不要であるなどの特徴があり，ユーザにとって使い易いと言われる．しかし，一つの“正しい”MATLABプログラムが，与えられる入力の属性に応じて多様な振る舞いをし得るため，高速処理のための最適化を静的に施すことは必ずしも容易ではない．これに対し，いくつかの変数の属性を固定することによってプログラムを特殊化して高速化を図る方法があるが，変数宣言の導入はMATLABユーザにとって敷居が高い可能性がある．そこで我々は，ユーザに負担をかけることなく特殊化による高速化を施せるよう，MATLABの対話環境からプログラムを実行させる時点で特殊化のために必要な情報を得てプログラムのコンパイルおよび実行を行うシステムを検討し，プロトタイプを実装した．本稿では，本処理系の設計について，簡単な実測結果を示しつつ述べる．

Runtime Specialization of MATLAB Programs

Manami Amimoto,† Hideyuki Kawabata† and Kitamura Toshiaki†

MATLAB is a widely-used dynamically-typed language which is known to be useful especially for prototyping. However, due to its dynamic nature, a well-written MATLAB program can essentially retain many functionality applicable for a lot of contexts, which in tern causes the difficulty of static optimization for high-speed execution. One of major approaches for translation of MATLAB programs is *specialization*: generation of a tailored program assuming a limited execution environment. This approach, although known to be effective in terms of the execution speed of translated programs, might require the user's assistance. In this article, we introduce a dynamic compilation system that works without user's support. Optimization is carried out at the invocation time of the program using the actual parameters' attributes. The system is intended to be used with MATLAB interpreter so that the user can enjoy handy built-in routines of MATLAB with dynamically translated programs. Experimental results show that the system operates fairly efficiently.

1. はじめに

MATLABは行列計算プログラム記述を簡潔に行なうことができる言語である[1]．MATLABプログラムでは変数宣言は不要で，変数や式の型は動的に決定される．変数は行列を表し，また各種の演算子には数値アルゴリズム記述での慣例を踏襲した多相性が与えられているので，ユーザは記憶領域の手間やデータ構造の詳細を意識することなく，密行列／疎行列を交えた計算処理を手短に記述することができる．その他，個々の行列計算はチューニングされたライブラリルーチンで高速に処理される，高機能な組み込み手続きが豊富である，などの特徴があり，MATLABは数値シミュレーション等においてラピッドプロトタイピング用途を中心に広く用いられている．

MATLABはユーザにとって使い易いと言われる．しかしながら，一つの“正しい”MATLABプログラムが，与えられる入力の属性に応じて多様な振る舞いをし得るため，高速処理のための最適化を静的に施すことは必ずしも容易ではない．これに対する基本的なアプローチの一つとしては，各手続きについて，使用する文脈を特定して手続きを特殊化(specialize)する方法が挙げられる[2),8)]．我々の開発しているCMCも特殊化による高速化を支援する処理系である[4)]．これらはいずれも，MATLABインタプリタとは独立な環境で実行させることを想定しており，各手続きに対して用途を絞って機能的な制約を課すことにより，動的処理が排除できる可能性を高めて静的解析による高速化を得ようとするものである．文脈の指定には，ユーザに入力データのサンプルを用意させたり，プログラム中のいくつかの変数の属性をユーザに宣言させる，あるいはより抽象的な制約条件をユーザに記述させる，などの方法がある．しかしながらこれらはいずれも，プログラム変換に際して(1)ユーザになにがしかの負担を課すものであるし，また(2)特殊化したプログラ

† 広島市立大学
Hiroshima City University

ムの管理の手間も必ずしも解消されているとは言えない．加えて，(3)MATLAB 対話環境との連携も十分には考慮されていないので，MATLAB の持つ高機能のライブラリルーチンをプログラム中で利用するなども多くの場合困難である．

ユーザが MATLAB インタプリタから呼び出し可能なプログラムを用意する方法は，MATLAB 言語スクリプトである M-file 形式以外に，MEX-file 形式がある．MEX-file 形式を用いれば，MATLAB インタプリタから呼び出すことのできるプログラムを C や Fortran 等で記述することができる．すなわち，M-file に対して動的処理を排除して特殊化した MEX-file を構成する仕組みを用意できれば，MATLAB 対話環境の機能を享受しつつ，処理に時間のかかる記述を高速化することが可能になり，ユーザにとってたいへん都合がよい．我々はこれまで，CMC に対して指示行付 M-file から MEX-file のソースを生成する機能を組み込んだ [6]．この機能により上述の (3) の問題点は緩和されたが，特殊化のための指示はユーザが M-file に記述しておく必要があり，依然として上記 (1) および (2) の問題は残されていた．

本稿では，ユーザに負担をかけることなく MATLAB 対話環境から起動されるプログラムを特殊化により高速化する方式を提案する．本方式は，M-file として記述されたプログラムに対し，実行時（実行開始時）に，与えられる引数の属性値をもとにして変換（特殊化）を施すものである．プログラム変換には CMC を用いる．変換後の MEX-file は，引数の属性値集合と対応づけて管理し，同一の属性値集合の引数を伴う呼び出しの際には再利用する．開発したプロトタイプでは変換処理を高速に行うことには特に注意を払ってはいないが，コンパイル処理時間を含めても処理時間が大幅に短縮される場合も確認されている．

以下，2 章で MATLAB プログラムの実行方式に関して概説し，3 章では我々の設計した実行時特殊化の枠組みとその実現方式について述べる．いくつかの実測結果は 4 章で示し，5 章で現状の問題点などを整理する．6 章で関連研究に触れる．7 章はまとめである．

2. MATLAB：言語とその実行方式

2.1 M-file

MATLAB 対話環境においてユーザが用意するプログラムは function M-file と呼ばれるテキストファイルである（以下，単に M-file と呼ぶ）．1 つの M-file は 1 つの関数から成っており，ユーザは適切な引数および返り値を受ける変数を指定してその関数を呼び出すことが出来る．対話処理中あるいは他の関数記述の解釈実行中に，任意の関数を呼び出すことができる．

M-file 記述における変数は行列を表す．MATLAB における演算子は行列演算をサポートする多相的なもので，数値計算コードの記述が簡潔に行なえる．M-file 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) はそれに対応する MATLAB 記述である．両者の類似は明らかでる．

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 ture
  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) An M-file implementation of the power method

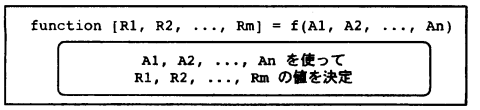
図 1 MATLAB のコードの例

MATLAB では変数の型宣言は不要で，各演算子による計算処理内容はオペランドの型や形状（ベクトルか行列か，密か疎か，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．

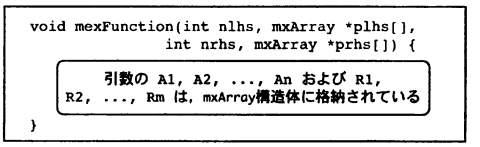
2.2 MEX-file 形式の利用：C によるコード記述

ユーザが MATLAB インタプリタから呼び出し可能な関数を用意する方法は，M-file 以外に，C や Fortran を用いて MEX-file を用意する方法がある．

M-file と MEX-file の構造の違いを図 **2** に示す．図 2 (a) は，引数が `A1`, `A2`, ..., `An` ，戻り値が `R1`, `R2`, ..., `Rm` で名前が `f` である M-file，図 2 (b) はそれに対応する MEX-file ソースを C で記述した例を示している．図 2 (b) から分かるように，MEX-file のエントリポイントは `mexFunction` に固定されていて，対応する M-file の手続き名や引数名とは無関係である．MATLAB における変数の実体は `mxArray` 構造体のインスタンスであり，`mexFunction` に渡される引数は `mxArray` 構造体を指すポインタ配列である．`mexFunction` の中での計算処理は，`mxArray` 構造体（ユーザが create/destroy することもできる）のフィールドを `mxGetPr()` や `mxSetPr()` などのセレクタルーチンを用いて直接操作したり，`mxArray` どうしの演算を（`mexCallMATLAB()` 呼び出しにより）他の M-



(a) M-file の概要



(b) MEX-file ソース (C 版)

図 2 M-file と MEX-file ソースの外観

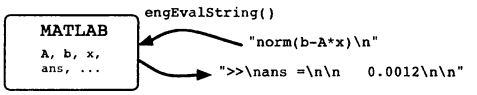


図 3 MATLAB Engine ライブラリの利用

file/MEX-file/組み込み手続きの起動によって行うことで，実現できる．ただし，M-file スクリプトの記述と比較すると MEX-file ソースの記述は大幅に煩雑である上に，MEX-file を用いることが必ずしも高速化につながるとは限らない．

なお，呼び出す側のコードは，呼び出されるルーチンがどのような形式で用意されているかを意識することなく記述できる．例えば図 1 (b) のプログラムは，関数 `abs()` や `norm()` が M-file，MEX-file，組み込みルーチンのいずれであるかによらず実行できる．

2.3 MATLAB Engine ライブラリ

MATLAB インタプリタは，C や Fortran で記述されたプログラムの計算エンジンとして利用することも出来る．例えば C プログラムであれば次のライブラリルーチンにより，文字列を媒介とした MATLAB インタプリタとの対話が実現できる（図 3）．

- `engOpen()`, `engClose()`
  子プロセスとしての MATLAB の起動および終了
- `engEvalString()`
  MATLAB への命令列の送信
- `engOutputBuffer()`
  MATLAB の出力を受けるバッファの用意

なお，Engine ライブラリを用いた場合でも，グラフィックスルーチンなどを利用することもできる．

Engine ライブラリを用いて MATLAB 対話環境をバックエンドに持つシェルプログラムを実現すれば，ユーザ入力に対してシステム側で加工を加えた命令列を MATLAB に与えることができる．第 3 章で述べる我々の処理系は，この機能を用いてユーザと MATLAB との間での処理を実現している．

3. MATLAB プログラムの実行時特殊化

我々は，静的解析により MATLAB プログラムを C/Fortran に変換できる CMC を用い，MATLAB 対話環境において実行時特殊化機能を実現する仕組みを試作した．本章では，我々のシステムの構成方式について述べる．なおここでは，“実行時特殊化” は，“実行開始時”，すなわち，MATLAB 対話環境においてユーザがプログラムの実行開始を指令した時点で，プログラムに特殊化を加えることを指している．

3.1 概　　観

ここで想定している実行時特殊化は，MATLAB 実行環境で対話的に起動されるプログラムの実行開始時に，それへの引数の情報を取得して，実行されるプログラムを書き換えることによって行う．対象とするのは M-file として参照できるプログラムで，内部で CMC を利用することによって適宜 MEX-file を生成してそちらを呼び出す．ユーザが予めプログラムに特殊な指示を書き加える必要はないし，どのデータを利用するのかを指定する必要もない．

特殊化に利用する文脈情報とは，各プログラムの実行開始時点における，実引数の変数属性情報である．具体的には，各変数（一般には行列）について，次の情報を収集して利用する．

- プリミティブ型 (real, complex, ...)
- 形状 (scalar, colvec, rowvec, full, ...)
- 構造 (dense, sparse)

これらはいずれも，個々の変数の実体である `mxArray` 構造体のフィールドを参照して得ることが出来る．

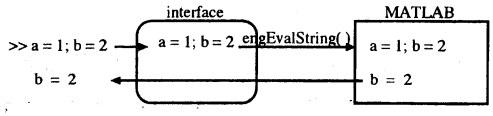
一旦特殊化したオブジェクト (MEX-file) は，同一の文脈における起動が再度行われた場合には再利用する．すなわち，引数の属性値に応じて特殊化した MEX-file のバリアントを扱うことができる．どのバリアントを用いるか，および，(M-file が更新されたなどの理由で) 再特殊化を行う必要があるかどうかの判断などは，ユーザ指示に頼ることなく行う．

3.2 構　　成

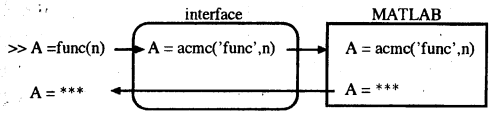
我々は実行時特殊化を次のコンポーネントを組み合わせることで実現した．

(1) ユーザからの入力文字列を解析するインターフェース部
(2) 引数情報を得て CMC や mex コンパイラ等を起動するドライバ
(3) M-file から MEX-file を構成するためのツール群：CMC，C コンパイラ，mex コンパイラ

ユーザと MATLAB インタプリタとの間に位置する (1) は，MATLAB をバックエンドで起動して Engine ルーチンにより MATLAB と通信する独立したシェルである．M-file に特殊化を施して MEX-file ソース



(a) acmc の起動が不必要である例



(b) acmc により特殊化を施した場合

図 4　ユーザインターフェース部の動作

を生成する処理自体は CMC により行うが，そのためには変換対象の M-file に適切に指示行を挿入する必要がある．このファイル操作や，CMC の起動および MEX-file ソースから MEX-file 生成に使用する mex コンパイラの起動，さらには生成された MEX-file の起動を行うのが，上記 (2) のドライバである．

実装方式としては，(1) と (2) をひとまとまりのプログラムとして構成する方法と，それぞれを独立なモジュールとして構成する方法が考えられる．前者の場合はシステムの構造がシンプルであるが，ユーザインターフェース部が引数情報等を MATLAB システムに (Engine ルーチンを用いて文字列経由で) 問い合わせて把握しておく必要がある．それに対し，後者のように (2) のドライバ自体を MEX-file として MATLAB 環境の中で実行させれば，MATLAB 実行環境の情報を (`mxArray` 操作ルーチン呼び出し経由で) 容易に得ることが出来る．ここでは，後者を採用した．

以下，(1) および (2) について，各項で述べる．

3.2.1　ユーザインターフェース部

MATLAB 対話環境では，M-file 等で提供されているプログラムの単独起動の他に，複数のプログラムを連続的に実行させる命令列の記述，繰り返し構造や条件分岐を用いた制御構造の記述を行うことが出来る．すなわち，一連のユーザ入力によって，複数の M-file の実行が起動され得る．ユーザインターフェース部に求められる作業の内容は，特殊化すべき複数の M-file の名前と，それぞれの M-file で参照される引数の名前の列の組を抽出し，それぞれについて，適切なタイミングでドライバを呼び出して特殊化したコードを実行するよう，一連のユーザ入力文字列を整形し，MATLAB インタプリタに送ることである．

我々のプロトタイプでは，ユーザからの入力として，コマンド (UNIX コマンドも含む) の単独実行，あるいはそれを複数連ねたコマンド列にのみ対応している．各コマンドについて，それが M-file 呼び出しであるかどうかを調べ，もしそうならば CMC により変換するように命令列を変換する (図 **4**)．例えば

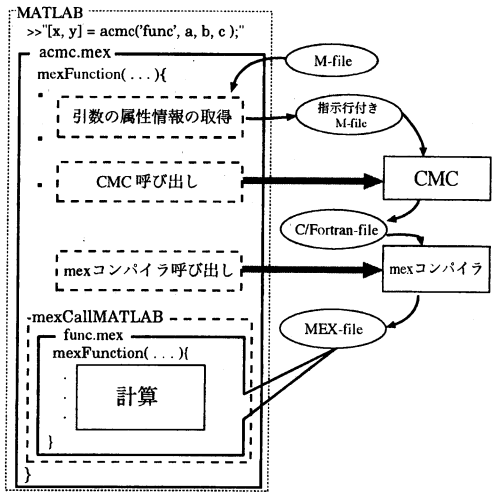


図 5　MEX-file によるドライバの動作の様子

```
>> tic;[x,i]=sor(A,x0,b,tol);toc
```

という入力列に対し，`sor.m` が参照可能であれば

```
>> tic;[x,i]=acmc('sor',A,x0,b,tol);toc
```

に変換して MATLAB に渡す．すなわち，`sor.m` を，引数 `A`, `x0`, `b`, `tol` の情報をもとにして特殊化し，MEX-file 化する．ここで 'acmc' は次項で述べる MEX-file で構成されたドライバである．MATLAB は，この文字列を受け取るとセミコロンで区切られた手続き呼び出しを順に実行する．すなわち，`tic` の実行，`acmc` の実行，`toc` の実行が順に行われる．このうちの `acmc` の実行中に，(特殊化後の) `sor` が実行される．

ユーザインターフェース部には基本的に「状態」を持たせないようにしているが，カレントディレクトリの情報はユーザ入力から検出して更新する．

3.2.2　引数情報取得と CMC の起動

前項で述べたインターフェース部により特殊化を加えることを指示された手続きは，MATLAB 環境から得られる引数の属性情報とともに，子プロセス内で起動された CMC や `mex` によって，MATLAB から呼び出し可能な MEX-file に変換される．我々のプロトタイプでは個々の手続きについてこの一連の処理を行うドライバを，MATLAB コマンドラインから呼び出せるように MEX-file として構成した (図 **5**)．例えば

```
>> [x,i]=sor(A,x0,b,tol);
```

という手続き実行の代わりに，

```
>> [x,i]=acmc('sor',A,x0,b,tol);
```

という対話入力を行えば，`sor.m` が CMC によって MEX-file に変換された上で呼び出される．

本ドライバは，手続き名とそれへの引数列，および戻り値を格納する変数列を引数として，MATLAB インタプリタにより起動される．本ドライバの処理内容

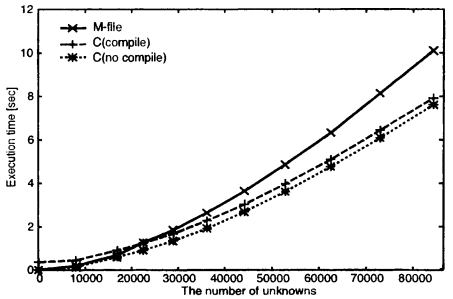


図 6　CG 法プログラムの実行に要した時間.

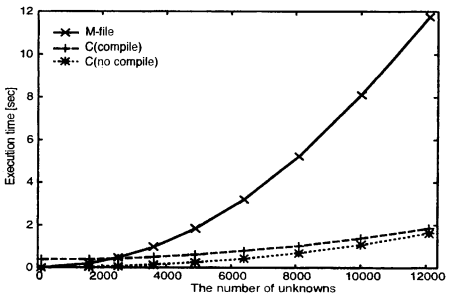


図 7　SOR 法プログラムの実行に要した時間.

は具体的には次の通りである.

(1)　指定された手続き名に対応する M-file をもとに, 引数の属性情報を指示行として挿入した M-file を新たに作成する.

(2)　新たに生成した M-file を CMC で MEX-file ソースに変換し, mex で MEX-file に変換する.

(3)　生成した MEX-file を呼び出し, 実行結果を戻り値として MATLAB インタプリタに返す.

本ドライバには, ある M-file に対する特殊化の結果 (MEX-file) が過去に得られていれば, それを再利用する機能を持たせている. "特殊化の結果" である MEX-file は CMC に与える引数属性情報と M-file との組から決まるので, MEX-file のファイル名を変更したバリアントを生成して対応関係を把握しておけば再利用が可能となる. ただし M-file の内容を処理系が把握しておくのは効率的ではないと考えられるで, 現在の実装では, CMC に与える引数属性情報とそれに対応する MEX-file のバリアント名との対応を把握しておき, 処理対象の M-file よりも最終更新時刻が後である当該 MEX-file バリアントがあるかどうかで再変換の必要性の有無を決定している.

4. 実　　測

開発したシステムの有効性の評価のためいくつかのプログラムを用いて実測を行った. 評価基準は特殊化によるプログラムの実行時間の短縮の度合いである. 実測に用いたアプリケーションは, 疎行列を係数行列に持つ連立一次方程式の求解 (CG 法および SOR 法) と, 密行列の要素アクセスを反復するベンチマークである. 実測に用いた計算機の仕様を以下に示す.

- CPU: Pentium4 3.0GHz (主記憶 512MB).
- OS: Linux 2.6.16
- MATLAB: Version 7.1.0.183 (R14) SP3
- C コンパイラ: gcc 4.1.1 -O3

4.1 CG

2 次元正方格子上で Laplace 方程式を自然順序付け 5 点差分で解いた. 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.

```
function [B] = thr(A, th, alpha)
B = A;
for j = 1 :  size(B, 2)
  for i = 1 :  size(B, 1)
    if B(i, j) > th
      B(i, j) = B(i, j) * alpha;
    end
  end
end
```

図 8　密行列の要素参照を反復するプログラム

表 1　図 8 の実行に要した時間

行列の次元数	MATLAB [sec]	MEX-file [sec]
1000	0.030	0.022
2000	0.12	0.086
4000	0.47	0.32

実測結果を図 6 に示す. 実行開始時の特殊化処理に要する時間はほぼ一定で 0.5 秒程度であった. 規模の大きいプログラムの実行の際には, 特殊化に要する時間が実行時間全体に対して占める割合は僅かである.

4.2 SOR

前項で用いた連立一次方程式を SOR 法により解いた. 諸条件は前項と同様である. SOR 法による加速係数は 1.8 とした. 実測結果は図 7 に示す.

CG 法の実測結果と異なり, SOR 法では, 特殊化による実行時間短縮の効果が大きい. SOR 法のプログラムでは下三角行列 M に対して M\N という式が計算されるが, CMC が静的に前進代入コードに変換するのに対し, MATLAB は動的にオペランドの属性 (対角行列か, 下三角か, など) を調べてから計算手段を決定する. この $O(n^2)$ の操作の有無が実行時間差の主要な成分であると考えられる.

4.3 密行列の要素参照

図 8 に示すプログラムを実行した. 実測結果は表 1 に示すとおりである. 表 1 より, MEX-file による実行が MATLAB よりも 3 割程高速化されている. 行列の要素アクセスが頻繁に行われる計算では, 記憶領域管理やプログラムの実行制御における動的処理を排除することによる実行時間短縮効果が現れている.

```
function [i] = fib(n)
%cmc integer,scalar ::  t
%cmc [t]->[t] ::  fib
%cmc t ::  n
if n == 1 || n == 2
  i = 1;
else
  i = fib(n-2) + fib(n-1);
end
```

図 9　フィボナッチ数列の要素を計算するプログラム

表 2　フィボナッチ数列の要素の計算に要した時間

項番号	MATLAB [sec]	MEX-file [sec]
20	0.33	0.00017
25	3.6	0.0016
30	40	0.018

5. 議　　論

開発中のプロトタイプは，script M-file の起動や，for 文，if 文などの制御構文を含むユーザ入力にはまだ対応できていない．これに対しては，ユーザからの入力を一旦 script M-file に書き出す機能，および，script M-file を function M-file に変換する機能を実現することで対処可能である．

属性値に基づく特殊化のために収集する情報がどのようなものであるべきかについても議論の余地がある．例えば，現状では変数（行列）のサイズ情報は最適化に利用していない．しかしこれを考慮に入れれば，より高速実行可能なコードを生成できる可能性もある．一方，MEX-file 生成に要する時間が増大してしまうとすれば，小さいプログラムに対しては無視できないオーバーヘッドとなり得る．扱う属性値，採用する最適化手法を含め，プログラム変換を高速に行うための工夫について，検討の余地は大きい．

現在の MATLAB インタプリタが高速に処理できないプログラムの例の一つは，手続き呼び出しが頻発するコードである．例えば，図 9 に示す再帰呼び出しによりフィボナッチ数列の要素を計算するプログラムの実行に要する時間は図 2 に示すとおりである．図に示すようにユーザ指示行を挿入して CMC で C に変換した MEX-file は，MATLAB での実行と比較して 3 桁高速であった．これをユーザ指示行無しで（実行時引数情報のみで）実現できるように，再帰手続きの型推定機能を実装することも今後の課題の一つである．

6. 関 連 研 究

MaJIC (Matlab Just-In-Time Compiler)[3] は，M-file を Fortran 記述に静的に変換するコンパイラ FALCON[2] をベースに，MATLAB V6 と同様の対話型環境において，JIT コンパイルを提供する．変数の属性推定機能や複数の特殊化バリアントを投機的に生成する機能も持っている．しかし MATLAB ランタイムライブラリへの依存度が大きいと考えられ，MATLAB V7 以降で同等の機能を維持できるかどうかは明らかでない．また，再帰手続きの型を推論できるだけの機能はなく，疎行列の扱いも考慮されていない．

7. ま と め

本論文では，MATLAB で記述された計算コードをユーザに負担をかけず実行時に特殊化し，高速化する処理系の実現方式に付いて述べた．この処理系の実現により，MATLAB 対話環境との連携を保ちつつ，ユーザに負担をかけずに特殊化して MATLAB プログラムを高速処理することが可能になる．いくつかのプログラムを用いた実測の結果では，単純に CMC を用いて実行時特殊化を行っているにもかかわらず，特に規模の大きい計算について，効果的であった．

今後の課題としては，手続き呼び出しの処理への対応などによる MATLAB との互換性の向上，制御構文の入力に対しての処理などがあげられる．

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) http://www.mathworks.com/
2) 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).
3) George, A. and David, P.: MaJIC: Compiling MATLAB for Speed and Responsiveness, ACM SIGPLAN 2002, 2002.
4) 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).
5) 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).
6) 川端英之，北村俊明: 高速な MEX-file を生成できる MATLAB コンパイラ，情報処理学会研究報告，HPC-100, pp.43–48, Dec. 2004.
7) 川端英之，北村俊明: 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，情報処理学会論文誌：プログラミング，Vol.47, No.SIG6(PRO29), pp.21–36 (2006).
8) 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).