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

2007－HPC－111 (17)
2007／8／1

数値処理ライブラリを使用するプログラムの処理内容の「可視化」

木村充宏† 川端英之† 北村俊明†

大規模数値計算プログラムの開発においては，高い性能可搬性や開発期間の短縮を実現するために，数値処理ライブラリの使用が欠かせない．しかしながら，類似したルーチン名が多く，呼び出しに多くの引数を必要とする数値処理ライブラリのインタフェイスは必ずしも使い易いとは言い難い．また，数値処理ライブラリに依存したプログラムは可読性についても必ずしも優れているとは言い難い．これに対し，我々は，プログラムの処理内容の理解を助けるために，数値処理ライブラリの呼び出しを自然な数式表現に変換するシステムを提案する．変換後のコードは MATLAB インタプリタで実行可能で，コードの動作検証に用いることができる．本稿では我々が提案するシステムの設計と実装について述べる．

Visualisation for Numeric Processing Program with Library Routines

Mitsuhiro Kimura,† Hideyuki Kawabata†
and Toshiaki Kitamura†

Library routines for numerical computation have been needed to develop large-scale numerical computation programs in order to make portability of performance high and development period short. However, those routines are not necessarily very easy to use because their complicated interfaces are not understandable intuitively. Also, programs heavily dependent on such routines tend to be difficult to read. In this paper, we propose a system that translates library routine calls in numerical computation programs into mathematical expressions based on MATLAB. The translator helps users to understand the behavior and meaning of numerical programs. In addition, we designed the translator such that the output of our translator is MATLAB-executable, so that the behavior of the output is easily checked. We show the design and implementations of our system.

1. はじめに

大規模数値計算プログラムの開発を行うにあたって，BLAS[1] や LAPACK[2] 等の数値処理ライブラリが頻繁に利用されている．ライブラリとしてまとめられたルーチンの多くはインタフェイスの共通化がされており，また実行速度の点においても十分にチューニングされているものが多いことから，処理の中核となる部分をライブラリルーチンを用いて構成することで高い性能可搬性を持つプログラムを短期間で実装することが可能である．

一般にライブラリルーチンはそれ自身の汎用性のために目的とするプログラムに対して必要以上に多機能であったり多重定義を用いた実装がなされていたりする．特に，数値処理を行うライブラリはそれぞれのルーチンがとる引数によってその処理内容が微妙に異なる場合が多い．また，処理と名前の一貫性を保つために類似した名前のルーチンが多数存在する．それらを用いて作成されたプログラムは同じような名前のルーチンが多くの引数とともに何行にもわたって並ぶような構造をとる傾向にある．

このようなプログラムは処理内容の直感的な把握が難しく，デバッグ等の妨げになる可能性がある．加えて，一見しただけでプログラム全体の処理内容を理解することは必ずしも容易ではない．

これらの問題に対して，プログラムのコードに含まれるライブラリルーチンの呼び出しを自然な数式表現に変換するとプログラム全体の見通しが良くなり，可読性が向上すると考えられる．そこで，我々は処理内容の直感的な把握が困難な表現からそれを容易にする表現への変換を行うシステムの提案をする．

本稿では提案システムのプロトタイプとして開発したシステム，すなわち，数値処理ライブラリの呼び出しを含む Fortran90 をベースとした計算コードから数値計算コードを簡潔に記述可能な MATLAB[4] を用いた数式表現への変換系を軸にして，我々が提案する「可視化」システム全体についての説明を行う．

以下，2 章で本システムの入出力である数値処理ライブラリと MATLAB について概説し，3 章では我々

† 広島市立大学
Hiroshima City University

```
* y = ALPHA * A * x + BETA * y
  dgemv (TRANS, M, N, ALPHA, A
         LDA, x, INCX, BETA, y, INCY)
```

図 1　BLAS の dgemv ルーチンのインタフェイス

```
call dgemv('N', size(A,1), size(A,2), &
     &    1.0, A, size(A, 1), r, 1, &
     &    0.0, TMP0, 1)
TMP1 = ddot(size(r,1), r, 1, r, 1)
TMP2 = ddot(size(TMP0,1), p, 1, TMP0, 1)
alpha = TMP1 / TMP2
call dcopy(size(r,1), r, 1, rnew, 1)
call daxpy(size(TMP0,1), -alpha, &
     &    TMP0, 1, rnew, 1)
```

図 2　BLAS ルーチンの使用例

の設計した「可視化」システムの枠組みを述べ，4 章で実際のシステムの実装方式について述べる．そして，5 章で現状の問題点やその解決方法を整理し，6 章で全体をまとめる．

2. 準　　備

2.1 数値処理ライブラリ

大規模数値計算プログラムの開発に使用される数値処理ライブラリについて，広く用いられている BLAS を例に挙げて説明する．**図 1** は BLAS に含まれる行列とベクトルの積を計算する dgemv ルーチンのインタフェイスである．ルーチンを呼び出す際には引数として多くのパラメータを渡す必要があることがわかる．数値処理ライブラリには，C／Fortran 等の使用言語の違い，BLAS／LAPACK／SuperLU 等の処理内容の違いによって様々な種類があるが，どれも同様のインタフェイスを持つ傾向にある．

ゆえに，ライブラリルーチンの呼び出しを主として構成されたプログラムはみな類似した構造をとりがちである．ライブラリルーチンの呼び出しを主として構成されたプログラムの例として，BLAS ルーチンの使用例を**図 2** に示す．A は行列，TMP0，r，p，rnew はベクトル，TMP1，TMP2，alpha はスカラである．このコードは A，p，r を用いて rnew の値を求める計算を行うが，ライブラリ呼び出しが連続しているので，一見してそれを理解するのは難しいと考えられる．

数値処理ライブラリの使い難さを解消することをねらった試みとしては，C++等のオブジェクト指向言語を用いて情報隠蔽を行うことで，ライブラリルーチンの呼び出しに必要な行列の行数，列数，ベクトルの長さ等の引数をユーザが与えなくてもよくすることで，ライブラリ呼び出し自体を容易にする試みもある[3]．

```
alpha = (r' * r) / (p' * A * r);
rnew = r - alpha * A * r;
```

図 3　MATLAB コードの例

2.2 MATLAB

MATLAB は行列計算プログラムを簡潔に記述することができる言語である．MATLAB プログラムでは変数宣言が不要で，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，ユーザは記憶領域管理の手間やデータ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．そのため，プログラムの処理内容の直感的な把握が可能である．また，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．

図 2 で示したコードを MATLAB で記述すると**図 3** のようになる．図 2 と比べて，図 3 のコードは非常に見通しがよい．

3. プログラムの処理内容の「可視化」

本章では，我々が提案する“プログラムの処理内容の「可視化」”を実現するためのシステムの構成方式について述べる．

3.1 システムの概要

我々が定義する「可視化」とは，処理内容の直感的な把握が困難な表現からそれが容易になる表現への変換を指している．2.1 節で述べたように，図 2 のようなライブラリルーチンの呼び出しが多用されるコードを一見してその処理内容を理解することは難しいと考えられる．ルーチンの呼び出しに必要な引数の多さが原因であろう．これは，同様の処理内容を MATLAB で記述したコードが図 3 のように簡潔な表現になることからも明らかである．

コード全体での処理内容を理解するという点では，図 3 から得られる情報で十分である．また，数値処理プログラムの内容の把握は，プログラミング言語の記述よりも数式を用いた表現の方が容易である．そこで，我々は計算コードの処理内容の理解を妨げる可能性のある情報を取り除き，見通しをよくする，すなわち計算コードの「可視化」のために，ライブラリルーチンの呼び出しをより数式に近い表現に変換，すなわち MATLAB プログラム化を行う．

数式表現への変換だけであればクラスライブラリを用いた C++コードを出力する方法もあるが，この度は MATLAB コードを用いた出力を行こととする．

数値処理ライブラリの呼び出しを MATLAB を用いた数式表現で表すにあたって，次のようなことを考慮に入れるべきである．

(1) ライブラリルーチンの呼び出し部だけを正規表現等によるパターンマッチで置換することは必ずしも可能でない

(2) プログラム全体の意味を解析してエラーチェックを行えるようにしたい

(3) プログラム全体を変換して MATLAB で実行可能にし，デバッグ可能にしたい

このような要求を実現するためには，変換対象のライブラリルーチン呼び出しから数式表現への変換に必要な情報を得るために，ソースコード全体を文脈自由文法により構文解析する必要がある．

また，より見易い出力コードを得るために，構文解析で得た情報をもとに出力コードの整形を行う．このコードの整形には，定数の畳み込みや定数伝播，コピー伝播等のコード最適化技術を用いる．ただし，入力コードと出力コードの構造的な対応をとるために，処理の中核を担う可能性の高いライブラリ呼び出しを含むコードラインの移動はできるだけ行わないようにする．

3.2 システムの入力および出力

3.2.1 入　　力

我々のシステムへの入力は数値処理ライブラリの呼び出しを含む Fortran90 ベースの計算コードである．言語拡張は行わず，指示行等の記述をユーザに強制することはない．

数値処理ライブラリに依存して記述されたプログラムは使用言語，処理内容に関わらず，類似した構造をとるので，ライブラリルーチンの呼び出しに注目するとプログラム全体の処理内容の把握が可能になる．また，入力言語として Fortran90 以外をとる場合に対しても，すぐに対応可能である．

入力コード中に次に示す記述，

```
id ( arg1, arg2, arg3, ...)
```

が現れたら，それを数式表現への変換対象として認識する．ここで，id はルーチン名，arg*は引数を指す．これは Fortran90 ではファンクションとサブルーチンの呼び出しに当たる記述である．

3.2.2 出　　力

我々のシステムの出力は MATLAB コードである．これは入力コード中にあるライブラリ呼び出しをそれに対応する数式表現に変換することによって，コードの処理内容の理解を妨げる可能性のある情報を排除したコードである．

また，このコードは MATLAB インタプリタで実行可能である．これは，プログラムのアルゴリズムレベルでの検証や小規模データでのテストを効率的に行う手助けとなる．

```
(a) call dcopy (size(b, 1), b, 1, r, 1)
      -> r = b
(b) call dgemv ('N', 50, 50, alpha,
                  A, 50, x, 1, beta, r, 1)
      -> r = alpha * A * x + beta * r
```

図 4　簡単な入力に対する出力の例

```
(c) call dgemv ('N', 50, 50, alpha, A, 50,
                  x, 1, (sig*beta), r, 1)
    -> r = alpha * A * x + sig * beta * r
(d) call dgemv (Tr, 50, 50, alpha,
                  A, 50, x, 1, beta, r, 1)
    -> if(Tr == 'T')
          r = alpha * A' * x * beta * r
        else if(Tr == 'N')
          r = alpha * A * x + beta * r
(e) call dgemv ('N', 50, 50, alpha,
                  A, 50, x, 1, 0, r, 1)
    -> r = alpha * A * x + 0 * r
    -> r = alpha * A * x
```

図 5　少し複雑な入力に対する出力の例

ここで行われるライブラリルーチン呼び出しの変換処理では，ルーチン名をキーとしてシステムが持つ変換表を検索し，それが変換可能であるかの判断を行う．対象のライブラリ呼び出しが変換可能であれば，変換表検索の結果得られた出力記述に入力コードから得られた情報を反映させた出力を行う．

3.3 入力コードに対する出力コードの例

入力コードに対する出力コードがどのようなものになるかを説明するためにいくつかの BLAS ルーチンの呼び出しを入力として与えた場合の出力を例として示す．

3.3.1 簡単に変換できるライブラリ呼び出しの例

図 4 は簡単な入力に対する出力の例である．ここで呼び出している dcopy はベクトルのコピーを，dgemv は行列とベクトルの乗算を行う BLAS のルーチンである．ここに挙げた (a)，(b) は共にサブルーチン呼び出しの引数から変数を抜き出してそれぞれのルーチンに対応した MATLAB 表現に置換すればよい．

3.3.2 ライブラリルーチンの処理を決定する引数に変数や式がある場合

図 5 はより複雑な入力に対する出力の例である．(c) のサブルーチン呼び出しの第 9 引数は式表現であるが，そこに含まれる括弧は冗長である．このように，冗長な表現があれば出力の見易さを優先して省くことにする．

次に (d) の場合はもう少し複雑な変換処理が必要である．dgemv の第 1 引数は第 4 引数に与えられる

行列を転置して使用するか否かを決定する引数であるが，ここでは第1引数が変数であるために，変換後のMATLAB表現を一意に決めることができない．この問題を解決するには第1引数の値を条件とした場合分けを行う必要がある．このように出力後の処理を決定する引数が変数であった場合には，その引数が取りうる値を考慮した場合分けをして出力する．ただし定数伝播などで変数を定数で置き換え可能な場合は，図4(b)と同様の出力を得ることができる．

最後に(e)の記述が入力として与えられた場合について述べる．これは図4(b)の場合と同様にして出力を得ることができる．しかし，出力にはプログラムの実行に影響しない式表現“0 * r”が含まれている．出力後のコードの見やすさを考慮すると，この“0 * r”は出力から除去すべきである．

3.4 拡張性

本システムは入力コード中にルーチンの呼び出しを見つけると，それが変換可能であるかを変換表の検索によって判断し，出力を行う．このため，ユーザー定義のものも含めた新たなライブラリルーチンを導入した場合でも，その入力と出力の組をユーザが変換表に追加するだけで対応が可能である．これについては後で詳しく述べる．

4. 「可視化」システムの実装

本章では我々が提案するシステムの実装について述べる．

4.1 処理の流れ

前節で述べた点を考慮に入れて，我々のシステムがライブラリ呼び出しを数式表現に変換する手順を以下に示す．

ステップ1： 入力コードの字句・構文解析
ステップ2： コードの情報を解析
ステップ3： 変換処理に必要な情報の整理
ステップ4： 出力コード形式に変換
ステップ5： 出力コードの整形
ステップ6： 変換結果の出力

この処理の流れに従ったコード変換を，図2から図3へのコード変換手順を例にして説明する．

まず，ステップ1，2ではコード変換の準備として，図2のコードから抽象構文木の作成と変数情報の取得を行う．次にステップ3で，依存情報の解析及び変数情報の整理を行う．具体的には，直接出力コードの構造に影響する変数に対する定数の畳み込み，定数伝播，コピー伝播等を行う．ただし，図2のコードでは，出力に影響を及ぼすdgemvルーチンの第1引数が定数で与えられているので，このステップでの処理は行われない．

このステップ4では，ステップ1からステップ3で得られた情報とシステムが持つ変換表をもとにして，

```
TMP0 = A * r;
TMP1 = r' * r;
TMP2 = p' * TMP0;
alpha = TMP1 / TMP2;
rnew = r;
rnew = -alpha * TMP0 + rnew;
```

図6 整形前の変換コード

```
rnew = r - (r' * r) / (p' * A * r) * A * r
```

図7 可能な限りの最適化処理を施した場合の出力コード

出力コード形式の抽象構文木を作成する．ここで，出力コードの概形が決定する．この段階で**図6**に示す数式表現を得ることができる．

そしてステップ5では，ステップ4で得た新たな抽象構文木に対して，出力コードの整形を試みる．このコード整形には，定数伝播，コピー伝播等のいわゆるコード最適化技術を利用する．

最後に，ステップ6でコードの変換結果の出力を行う．この結果はMATLABインタプリタで実行可能な形式で出力する．このようにして図3のコードが得られる．ただし，ステップ5でコードの最適化処理を適用する度合いによっては，出力コードの可読性が失われる可能性があるので注意が必要である．図3のコードに更なる最適化処理を加えると，**図7**のコードになり，1つの数式で表現できる一方で式が長くなる．図3，図6，図7のうち，どれを出力コードとして得るかは，その判断をユーザに委ねるのが妥当であろう．

4.2 コードの変換

本システムがコードの変換処理のために用いる変換表の構造について述べる．この変換表はルーチン名と引数情報（名前，型）の組を主キー，出力に影響を与える引数とその値の組を副キーとするテーブルである．

例えばいくつかのBLASルーチンを変換表に登録すると**図8**に示したようにid_tagにルーチン名と引数情報の組，arg_tagに出力に影響を与える引数とその値の組，outputに出力コード形式の構文木が登録される．dgemvの呼び出しが図中の入力として与えられた場合は，その入力を解析して得られた情報をもとに，id_tagとarg_tagを検索し，outputを探す．outputが見つかったら，そこにある変数を入力コードから得た値で置き換えて出力する．ここで参照しているoutputの構文木は**図9**中の引数情報が“TRANS == 'T'”の場合の出力テンプレートから得られたものである．

id_tagに検索対象のルーチン名が存在しなければ，変換を行わずに入力記述をそのまま出力する．ルーチン名は存在するが，引数の数が一致しなかった場合は入力コードの誤りとしてエラーを出力する．また，arg_tagの検索で出力結果を一意に決定できなかった

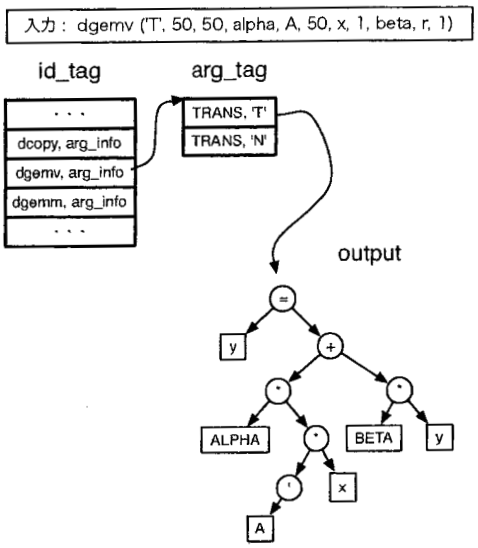

図 8 変換表

```
input :
 dgemv ( CHARACTER*1 TRANS,
         INTEGER M,
         INTEGER N,
         DOUBLE PRECISION ALPHA,
         DOUBLE PRECISION (:,:) A,
         INTEGER LDA,
         DOUBLE PRECISION (:) x,
         INTEGER INCX,
         DOUBLE PRECISION BETA,
         DOUBLE PRECISION (:) y,
         INTEGER INCY )

output :
 if (TRANS == 'T')
   y = ALPHA * A' * x + BETA * y
 else if (TRANS == 'N')
   y = ALPHA * A * x + BETA * y
 end
```

図 9 dgemv ルーチンの入出力定義

場合は図 5(d) のように，タグごとに if-else を用いた場合分けを行って出力する．もし引数の型が変換表に与えられたものと一致しない場合はエラーを出力する．

4.3 変換規則からの変換表の生成

本システムに変換規則を与えるためには図 9 に示すような定義ファイルを用いる．input 部の項目は変換対象とするルーチンのプロトタイプ宣言である．ここにはルーチン名と引数の型および名前を記述する．output 部の項目は変換後の MATLAB 記述のテンプレートである．この部分は，input 部で指定した引数名を用いて記述する．引数の値によって処理内容が異なる場合は if-else 文を用いて記述する．

定義ファイルパーザはこの記述をもとに入出力部の関係を解析して変換表を構築するためのルーチンを生成する．まず input 部から，直後の output 部で用いる各引数の型とそれに割り当てられる局所名を取得する．次に，output 部の構文解析により出力用の構文木を得る．

そして，この入出力の対応情報が変換表の 1 エントリを構成する．

現時点では，静的な定義，すなわち定義ファイルパーザが生成した変換表初期化ルーチンをシステム構築時にリンクする方式をとっているが，実行時の定義ファイル読み込みを可能にすることは容易である．

5. 考　　察

本章では，我々が提案したシステムのプロトタイプについての考察を行う．

5.1 マクロプロセッサとの違い

我々の提案するシステムは数値処理ライブラリの呼び出しを MATLAB 記述に変換する．図 4 のような簡単な例は，正規表現による文字列のマッチングと置換のみで処理することができるため，単なるマクロプロセッサ[5] を用いて定義文を置換することとほぼ同様の処理で実現できる．しかし，図 5 の例を処理するためにはそれだけでは不十分である．そこで，我々のシステムは入力コードを文脈自由文法により構文解析して抽象構文木を作成した後，それに基づいてコードの変換を行う．また 4.2 節で述べたような，変換対象の入力に対しての簡単な型チェックも行う．

5.2 入出力について

現在開発中のプロトタイプは，3 章でも述べたように，Fortran90 ベースの計算コードを入力として受け取り，MATLAB コードを出力する．だが同章 1.1 節でもふれた通り，入力言語のベースを Fortran90 に限定する必要はない．本システムのコード変換は手続き呼び出しごとに行うので，多言語対応は比較的容易であろう．

また，記述の変換対象とするルーチンの追加に関しても，システムが保持する変換表に新たに導入したルーチンを追加するだけで対応が可能である．この変換表の編集のために，ユーザが与えた入力と出力の組を追加する，不要になった変換対象をマッチングにより削除するなどの操作を対話的に行う環境を提供する予定である．

5.3 変換に必要な情報が不足した場合の処置

次に，図 5(d) の入力に対する出力について，現在は条件分岐を用いて結果として取りうる全ての場合を

出力している．しかし，出力コードに if-else 文が現れることは，コード全体の処理内容を直感的に把握することの妨げになる恐れがある．そこで，ライブラリルーチンの引数が変数だった場合に，それを定数伝播等で解決することができず，更にその引数の値が出力コードに影響を与えるものであれば，部分評価的にできるだけ値を決定して変換を行う等の対処が必要になると考えられる．

5.4 ライブラリ呼び出し以外の記述への対処

また，本システムが現在記述の変換対象としているのはライブラリルーチンの呼び出しだけである．しかし，入力コードを構成する要素はそれだけではない．ライブラリ呼び出し以外の記述も，可能であれば直感的に処理内容を把握し易い表現への変換が望まれる．例えば，ループ内での配列要素アクセスを解析してその意味をくみ取り，ベクトルや行列を用いた表現に変換することができれば，更に見通しの良いコードの出力が期待できる．このようなプログラムのイディオム認識によるコード変換がどの程度可能であるかは，今後検討したい．

5.5 ユーザ補助環境について

どのような出力コードが得られればプログラム全体の処理内容の直感的な把握に役立つのかは本システムを使用するユーザの主観によって決定される部分が多い．その決定は，我々が強制するのではなくできる限りユーザの指示によって決定するのが望ましい．また，コード変換のために必要な情報が入力コードから得られる情報だけでは足りなくても，ユーザから簡単な指示があれば問題を解決することができる場合も多いだろう．これらのことから，我々のシステムを使用するためのインタフェイスとして，ユーザとの対話環境の必要性が挙げられる．

5.6 プログラム開発環境として

3 章でも述べたが，本システムの出力が実行可能な MATLAB プログラムであることが，プログラムのアルゴリズムレベルでの検証や小規模データを用いたテストの手助けとなる．もしユーザが数式表現に変換されたコードを見ることでアルゴリズムの間違いを発見したとき，またはコードのアルゴリズムレベルでの最適化が施せることに気づいたとき，それを直接編集することで問題を解決することができれば，プログラムの開発効率の向上が期待できる．そのような開発環境の提供を視野に入れたシステムの開発も今後の課題の 1 つである．その一例として，我々の研究室で開発を行っている行列言語コンパイラ CMC[6] と本システムの連携を考えている．

6. ま と め

本論文では，数値処理ライブラリを使用するプログラムの「可視化」を行うシステムについて述べた．このシステムの実現により，コード中に存在するライブラリ呼び出しを MATLAB による自然な数式表現に変換することで，処理内容の直感的な把握が困難な記述表現からそれを容易にする記述表現への変換が可能になる．また，本システムの入力および出力は容易に拡張可能であるため，新たな入力言語およびライブラリが導入された場合もすぐに対応が可能である．

今後の課題としては，出力のための変換表を簡単に編集する機能の提供，コード変換に必要な情報が不十分な場合への対応，コードのイディオム認識を用いた記述変換の検討等が挙げられる．

参 考 文 献

1) http://www.netlib.org/blas/
2) http://www.netlib.org/lapack/
3) 木村充宏, 川端英之, 北村俊明: C++言語による疎行列計算クラスライブラリの設計と実装, 情報処理学会ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS2006) 論文集, p.32, Jan. 2006.
4) http://www.mathworks.com/
5) http://www.gnu.org/software/m4/
6) Kawabata, H., Suzuki, M., and Kitamura, T.: A MATLAB-based Code Generator for Sparse Matrix Computations, *Proc. (APLAS2004)* , *LNCS*, Vol.3302, pp.280–295, Nov. 2004.