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

2006－ARC－167 (6)
2006－HPC－105 (6)
2006／2／27

MATLAB記述に基づく PSBLASを用いた並列疎行列計算コードの生成

笹岡泰司† 川端英之† 北村俊明†

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等による記述に変換する処理系を開発している．本稿では，新たにCMCに導入した機能，すなわち分散メモリ型並列計算機向けのコード出力機能について述べる．開発中のシステムは，疎行列計算ライブラリであるPSBLASを用いるSPMDプログラムを出力する機能を備えている．CG法やBi-CGSTAB法といった基本的な疎行列計算プログラムを用いて数台規模の並列計算機環境で実測を行ったところ，ユーザに負担をかけることなく比較的良好な台数効果が得られることが確認できた．

A MATLAB-Based Code Generator for Parallel Sparse Matrix Computation utilizing PSBLAS

Taiji Sasaoka,† Hideyuki Kawabata† and Kitamura Toshiaki†

MATLAB is a language and an execution environment for matrix computations, that has been widely used. We have been developing CMC, a compiler for matrix computations, which translates MATLAB-based scripts into Fortran 90 programs by static analysis. In this article, we present a new feature of the system of generating SPMD-style parallel sparse matrix computation code utilizing PSBLAS. The user can construct parallel computation code without knowing the details of the machine's architecture which is intended to execute the code. Experimental results show that sparse solvers such as CG and Bi-CGSTAB methods implemented using PSBLAS can compete with MPI programs. Using CMC, the user can readily generate PSBLAS code to enjoy the performance of parallel computers quite easily.

1. はじめに

並列計算機の発展に伴い，従来では困難であった大きな規模の数値計算が可能となりつつある．しかしながら，ハードウェアのポテンシャルを十分に引き出すことは必ずしも容易ではなく，応用に従事するものにとっては並列計算機の高速性を生かしたプログラミングは簡単とは言えない．大規模な疎行列を扱う必要のある行列計算については特にコードが複雑なものになり易く，プログラム記述の困難さもさることながら，コードの保守性や性能可搬性を高く保つことは難しい．

行列計算プログラムを記述し易い言語や実行環境は数多く開発されている．Mathworks社のMATLAB[1]はその一つで，行列計算アルゴリズムの字面をほぼそのまま保ったままプログラム記述が行えるように構文が工夫されている．また動的型付けであるので変数宣言も必要としないし，疎行列を用いるプログラムも記述し易い．しかしながら，プログラムの実行速度については，規模の大きい疎行列を用いた計算や複雑なアルゴリズム記述が必要である場合は特に，FortranやCで記述したプログラムとは比較にならない場面もある．また，分散メモリ型並列計算機の効率的利用は現時点ではほとんど期待できない．

高速な数値計算のためのソフトウェア開発のアプローチとしては，BLASをはじめとするライブラリパッケージの整備およびチューニングが広く行われている．現在では多くの高性能計算機に対しハードウェア毎にチューニングされたベンタ提供の数値計算パッケージが利用可能であるし，BLAS等の標準的ライブラリについては自動チューニングに関する研究も広く行われている[2]ので，数値計算ライブラリを使用すれば性能可搬性の高いプログラムを開発し易いといえる．また，ScaLAPACKなどの並列計算機環境向けのライブラリも開発されているし，疎行列計算をサポートするPSBLAS[9]などの環境も利用可能である．しかしながら，数値計算ライブラリルーチンの使用にあたっては，ユーザは必ずしも直感的とは言えない（一

† 広島市立大学
Hiroshima City University

般に多くの）パラメータの指定方法を熟知する必要があり，プログラム開発やデバッグ作業が容易であるとは必ずしも断言できない．

我々は，数値計算プログラムを記述し易い MATLAB をベースとするプログラムから高速実行可能なコードを生成する方式について検討を行っている[3),4)]．開発中の処理系である CMC コンパイラを用いたシングルプロセッサ上での実測では，問題に依存して速度向上率が大きく変化するものの，疎行列を含む計算処理を MATLAB での実行よりも高速化できることが確認されている．

本稿では，この度新たに CMC コンパイラに導入した，数値計算ライブラリを利用したコードを生成する機能，特に PSBLAS を用いて分散メモリ型並列計算機上で実行可能なコードを生成する機能について述べる．疎行列に対する線形ソルバを題材とした比較的小規模な分散メモリ型計算機上での実測では，現在の CMC コンパイラによって自動的に処理できる範囲で，PSBLAS を用いた並列化が効果的であることが確かめられた．

2. 準備：並列疎行列計算用ライブラリ

2.1 線形計算パッケージの階層的構成

数値計算プログラムで頻繁に用いられる基本的なアルゴリズムの多くは，ライブラリパッケージとしてまとめられている．代表的なものとして，行列とベクトルの間で行われる種々の演算をまとめた BLAS や線形系ソルバなどをまとめた LAPACK[5)] が挙げられる．疎行列計算のためのライブラリインターフェースとして Sparse BLAS も提案されている[6)]．これらは多くの計算機の上にチューニングされたかたちで載せられていることが多い．

LAPACK は分散メモリ型並列計算機上で実行させるための拡張 (ScaLAPACK[7)]) も行われている．そこではまず共通化した通信ライブラリのインターフェース (BLACS[8)] と呼ばれる) を想定し，その上に計算ルーチンを構築するという手法がとられている．BLACS は例えば MPI の上に実装される．

ScaLAPACK は密行列データ構造しかサポートしていない．疎行列計算を並列に実行するためには，それに対応した計算ライブラリが必要である．PSBLAS はそのような用途を想定して開発されたライブラリパッケージの一つである．本稿で述べる疎行列計算の並列処理では以下に述べる PSBLAS を用いる．

2.2 PSBLAS について

PSBLAS は分散メモリ型並列計算機上で疎行列線形システムに関する計算用に設計されたライブラリパッケージである[9)]．PSBLAS を用いた並列プログラムは，SPMD による実行形態で行列計算をデータ並列的に行う方式をとる．

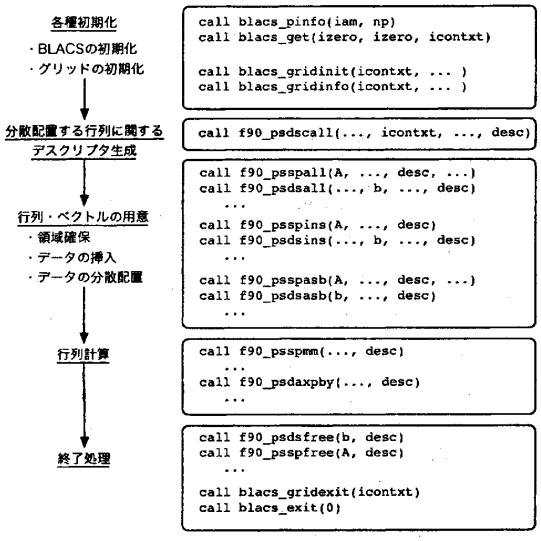


図 1 PSBLAS による並列プログラムの概要

図 1 に，PSBLAS による行列計算プログラムの構造の概略を示す．PSBLAS を用いたプログラムでは，行列計算の並列処理に先んじて，各行列やベクトルは，別に用意しておく「デスクリプタ」で指定される方法で各プロセッサに分散配置させておく．行列計算は，分散配置された行列やベクトル（を指す「ポインタ」）および対応するデスクリプタを与えて個々の計算ルーチンを呼び出すことによって行う．各計算ルーチンは，必要なデータ通信をユーザに見えない形で行いつつ指定された計算を実行する．計算ルーチン内では大域的な通信が行われるので全プロセスが同じ計算ルーチンをコールする．

PSBLAS は，通信ライブラリとして BLACS を用い，ノード内の計算処理には BLAS や Sparse BLAS などを用いている．PSBLAS のルーチンは，疎行列と密行列の積やベクトルの内積を求めるなどの計算処理を並列に行なうルーチンと，各プロセスにデータを分配したりする補助的なルーチンで構成される．プロセスの論理的トポロジの設定は BLACS ルーチン呼び出しにより行う．BLACS 自体は 2 次元グリッドに対応しているが，PSBLAS では 1 次元グリッドのみをサポートしている．例えば疎行列 A と列ベクトル b の積を求める場面がある場合は，A と b の両者を行方向に分割した分散配置とする必要がある．

行列の分散配置は，行方向の分割に限定されるものの，ユーザは各行列やベクトルの第 i 行目が何番目のプロセスに割り付けられるかを返す関数を記述することによって，任意のパターンでの分割を指示することができる．

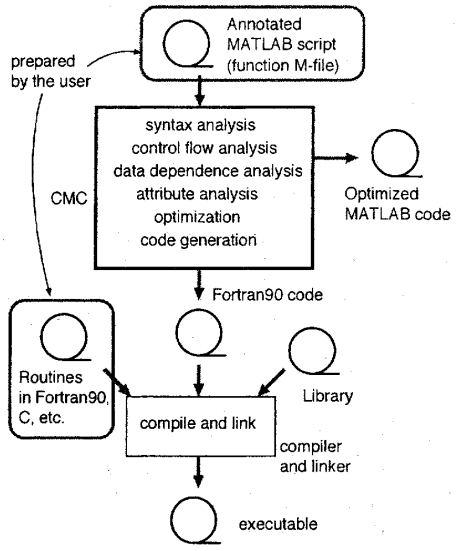


図 2　CMC によるプログラム解析とコード出力の流れ

3. MATLAB ベース記述からの PSBLAS コード生成

3.1 CMC による MATLAB 記述の解析 [3),4)]

CMC コンパイラによるプログラム変換処理の流れを図 2 に示す．CMC は注釈付きの MATLAB プログラムを入力として受け，各種解析と最適化を行った後に，Fortran などによるサブルーチンを出力する．このルーチンは他のライブラリと自由にリンクして実行することができる．

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全て静的に決定される．

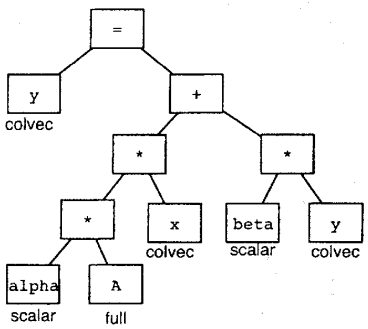
なお，これらの解析の後に，無用命令の除去やループ不変式の移動等の古典的な最適化，および，行列計算に関する強さの軽減処理なども行う [3)]．

3.2 BLAS を用いるコードの生成

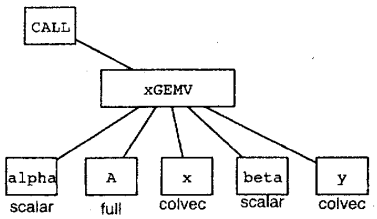
BLAS ライブラリには，様々な行列計算アルゴリズムを実装できるように多数のインターフェースが用意されている．そのため BLAS ルーチンを用いたプログラム記述の際にはどのルーチンをどのように組み合わせて用いるかに選択の余地があり，効率よいコードの生成のためには工夫が必要である．

開発中のシステムでは，構文木の各ノード（変数参照や演算）についてデータの形状などの属性値を検出した後に，BLAS ルーチンが利用できる部分の抽出を行う．例えば，A が実行列，x と y が実ベクトル，alpha と beta が実スカラ変数であったときに

$$y = alpha * A * x + beta * y;$$



(a) Original syntax tree



(b) Modified syntax tree

図 3　式表現とその変形

という文に対する構文木は図 3(a) のようになるが，この文はその全体が BLAS の DGEMV ルーチン呼び出しに置き換えられるので，図 3(b) に示すように構文木を変形する．

いくつかの変数が欠落している場合（例えば y=A*x;）や，交換則を適用できる演算子がある場合（加算やスカラ乗算など；例えば y=y-A*x;）についてもある程度考慮したマッチングアルゴリズムを用いている．また，変換は構文木に対してトップダウン的に行い，適宜中間変数を用いて部分木を切断して再帰的に置き換えている．

3.3 PSBLAS を用いる並列コードの生成

PSBLAS ライブラリを用いた並列プログラムは，2.2 節で述べたように，図 1 のような構造となる．ここで，MATLAB ベースの入力記述に対して，以下のような制約を仮定して並列化を施すならば，コードの変換処理はほぼ機械的にできる．実際，図 4 に示す CG 法のプログラムは図 5 のように書き換えれば図 1 中の「行列計算」にあたる部分に直接組み込むことができる．

(1) プロセスグリッドを一次元配置に限定する．
(2) プログラム中に現れる全ての行列とベクトルを全プロセスに唯一のデスクリプタを用いて（行方向分割で）分散する．
(3) 行列とベクトルに絡む演算を全てデータ並列的に全プロセスで処理する．

BLACS のみを用いた並列処理記述ではプロセスグリッドは 2 次元とすることも可能であるが，PSBLAS

```
function [x,i] = cg0s(A,x0,b,tol)
r = b-A*x0;
rn = norm(r);
x = x0;
p = r;
i = 0;
rho = r' * r
while 1
  i = i + 1;
  q = A * p
  alpha = rho / (p' * q)
  x = x + alpha * p;
  r = r - alpha * q;
  if norm(r)/rn <= tol, break, end
  rho_old = rho
  rho = r.'  * r
  beta = rho / rho_old
  p = r + beta * p;
end
```

図 4　CG 法のアルゴリズム

を使用する上では (1) が必須となる．また，制約 (3) は (2) を課す以上は必然的に課される条件である．すなわち，複数のプロセスに分散配置する行列やベクトルに絡む演算を行う場合には，各プロセスに閉じた計算をマニュアル的に行うことは実質的にはできず，PSBLAS の計算ルーチン呼び出しを用いる必要がある．結局，PSBLAS を用いた並列コードの自動生成を考えるにあたっては，制約 (2) をどのように扱うかが検討を要する点であるといえる．

我々の現在の実装では，行列やベクトルを全て単純に行方向のブロック分割により分散配置することを仮定している．後述するように，制約 (2) を課したままでこのような単純な方法を適用しても，数台規模の並列計算機で CG 法などを効率的に実行できる場合はある．個々の行列やベクトルに対応させて別々のデスクリプタを与えることができるようにすることは（分散させずに複写して持たせる場合も含めて），指示行などの導入で容易に対処できる．

4. 実測および評価

本章では CMC コンパイラによって生成される PSBLAS 並列プログラムの実測結果をまとめる．CG 法および Bi-CGSTAB 法による連立一次方程式の求解を題材として用い，CMC コンパイラの現バージョンが出力することの出来る PSBLAS による並列コードの実測による評価を行う．

PSBLAS コードの比較対象として，BLAS で記述した CG 法，Bi-CGSTAB 法のコードを MPI で並列化したものを用いた．データ分割は行ブロック分割を行う．MPI での実装に際しては，無駄なデータ転送を出来るだけ発生させないよう考慮している．

4.1 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通りである：

計算機 **P**　分散メモリ型並列計算機．

- 構成: 8 ノードの SMP クラスタ．

```
subroutine cg0s(A, x0, b, tol, x, i, desc)
use typesp
use typedesc
use f90psblas
use f90tools
  ...
type(d_spmat) ::  A
type(decomp_data_type) ::  desc
  ...
call blacs_gridinfo(...)
  ...
call f90_psaxpby(...,b,...,r,desc)
call f90_psspmm(...,A,x0,...,r,desc)
rn = sqrt(f90_psdot(r,r,desc))
call f90_psaxpby(...,x0,...,x,desc)   !  x = x0
call f90_psaxpby(...,r,...,p,desc)   !  p = r
i = 0
rho = f90_psdot(r,r,desc)
dowhile (.true.)
  i = i + 1
  call f90_psspmm(...,A,p,...,q,desc)
  alpha00 = f90_psdot(p,q,desc)
  alpha = rho/alpha00
  call f90_psaxpby(alpha,p,...,x,desc)
  call f90_psaxpby(-alpha,q,...,r,desc)
  rho = f90_psdot(r,r,desc)
  if ((sqrt(rho)/rn).le.tol) then
    exit
  endif
  rho_old = rho
  beta = rho/rho_old
  call f90_psaxpby(...,r,beta,p,desc)
enddo
  ...
end
```

図 5　PSBLAS による CG 法のプログラム

- OS: Linux 2.4.20
- 各ノードについて:
  - CPU: Intel Xeon 2.8GHz × 2
  - 2 次キャッシュ: 512MB（オンチップ）
  - メモリ: Dual Channel DDR 1GB
  - システムバス周波数: 400MHz
- ネットワーク: Myrinet-2000 (双方向 2Gbps)
- 通信ライブラリ: MPICH v1.2.5 (GM 2.0.5)
- C コンパイラ: Intel Compiler icc v7.0 -O3
- Fortran コンパイラ: Intel Compiler ifc v7.0 -O3 -static
- システム付属の BLAS を使用．

計算機 **S**　スカラ計算機．CPU は Pentium 4 2.4GHz，メモリは 1GB．OS は Linux 2.4.31．MATLAB V7 を実行可能．Fortran コンパイラは g95 -O3 を使用．BLAS はソースを g95 でコンパイルして使用．

計算機 P は合計 16CPU の SMP クラスタであるが，最大 8 並列の実測とし，同一ノードで複数のプロセスを実行させないようにした．計算機 S は，MATLAB V7 での実行時間と CMC により Fortran に変換した場合の実行時間とを比較するために用いた．

計時は標準ライブラリ関数 `gettimeofday()` をリンクして用いた．

4.2 CG　法

2 次元正方領域においてラプラス方程式

$$\partial^2 P/\partial x^2 + \partial^2 P/\partial y^2 = 0$$

を正方格子上の自然な順序付けの 5 点差分近似により

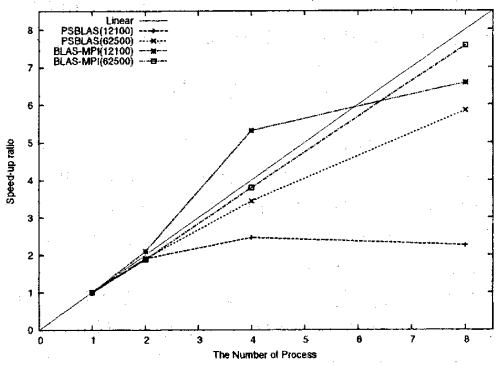


図 6　計算機 P 上での CG 法の実行結果（台数効果）

表 1　計算機 P（8CPU）での CG 法の実行時間．単位は秒．括弧内は収束までの反復回数．

次元数	PSBLAS	MPI
12100 (227)	0.0968	0.0363
22500 (334)	0.150	0.0674
44100 (458)	0.309	0.203
62500 (540)	0.511	0.429

解いた．ディリクレ境界条件として一辺で $P = 1$，他の辺で $P = 0$ とした．得られる連立一次方程式の係数行列は疎で実対称である．これを CG 法で解いた．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．

図 **6** は，未知数（すなわち行列の次元数）が 12100 および 62500 の場合にプロセス数を 1，2，4，8 と変化させたときの並列化による台数効果を示すグラフである．プロセス数を変更して実行するに当たっては，PSBLAS による並列プログラムおよび MPI によるものそれぞれについて，1CPU での実行も含め，全く同一のコードを用いた．図は各コードについて 1CPU での実行に対する速度比率を示している．PSBLAS，MPI のいずれについても，未知数が多い（問題サイズが大きい）場合には良好な台数効果が得られていることが分かる．

プロセス数が 8 の場合の実行時間の比較を表 **1** に示す．いずれの問題サイズにおいても MPI での記述の方が高速であるが，サイズが大きくなるにしたがって実行速度差が少なくなっていくことが分かる．

表 **2** は MATLAB インタプリタでの実行と CMC による変換後の実行の速度比較である．逐次計算機である計算機 S において CMC での実行は MATLAB よりも 2 倍以上高速である．これらを計算機 P での 1CPU による実行結果と比較すると，並列計算機において PSBLAS で得られる台数効果によって MATLAB の逐次実行を大幅にしのぐ実行速度が得られるということが確認できる．

4.3 Bi-CGSTAB 法

2 次元正方領域において楕円型偏微分方程式

表 **2**　計算機 S での MATLAB と CMC 出力との実行時間比較（CG 法）．計算機 P の 1CPU での PSBLAS と CMC の実行結果もあわせて示す．数値の単位は秒．括弧内は収束までの反復回数．

次元数	計算機 S（スカラマシン）		計算機 P（1CPU）	
	MATLAB	CMC	PSBLAS	CMC
12100 (227)	0.591	0.370	0.219	0.213
22500 (334)	1.87	1.04	0.607	0.595
44100 (458)	6.77	3.15	1.78	1.81
62500 (540)	11.9	5.71	2.99	3.09

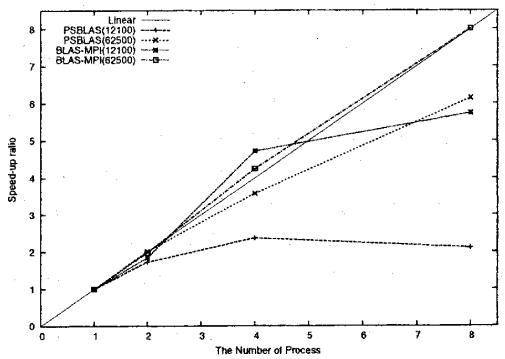


図 7　計算機 P 上での Bi-CGSTAB 法の実行結果（台数効果）

表 **3**　計算機 P（8CPU）での Bi-CGSTAB 法の実行時間．単位は秒．括弧内は収束までの反復回数．

次元数	PSBLAS	MPI
12100	0.150 (170)	0.0590 (166)
22500	0.261 (247)	0.107 (224)
44100	0.554 (340)	0.365 (330)
62500	0.762 (361)	0.557 (336)

$$-\partial^2 Q/\partial x^2 - \partial^2 Q/\partial y^2 + \alpha \partial Q/\partial x = 0$$

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いた．正方領域の 3 辺をディリクレ境界条件とし，1 辺は自然境界条件とした．格子間隔 h に対し，$\alpha h = 0.1$ とした．係数行列は実非対称行列である．

Bi-CGSTAB 法の実測結果を図 **7** および表 **3** に示す．初期解と反復終了の条件は CG 法の場合と同じにした．

PSBLAS による並列実行の効果が CG 法の場合とほぼ同様の傾向であることが確認できる．

5. 関連研究

MATLAB ベースのプログラムに対して ScaLAPACK を利用するコード記述を出力する処理系に関する研究は既に行われている [10),11)]．しかしながら疎行列を扱う機能についての言及はない．

PETSc は数値計算プログラム記述の補助のためのライブラリパッケージである [12)]．PETSc では，疎行

表 4 計算機 S での MATLAB と CMC 出力の実行時間比較 (Bi-CGSTAB 法). 計算機 P の 1CPU での PSBLAS と CMC の実行結果もあわせて示す. 数値の単位は秒. 括弧内は収束までの反復回数.

	計算機 S (スカラマシン)		計算機 P (1CPU)	
次元数	MATLAB	CMC	PSBLAS	CMC
12100	0.923(171)	0.519(170)	0.326(172)	0.305(169)
22500	3.38(242)	1.60(244)	0.854(226)	0.932(246)
44100	9.94(336)	5.06(331)	3.08(343)	2.72(318)
62500	13.6(372)	9.27(409)	5.10(395)	5.16(396)

列計算や並列処理をプログラミング言語ではなくライブラリレベルでサポートしており, プログラム自体は C や Fortran などで記述できる. PSBLAS よりもユーザインターフェースが簡素化されているが, 並列処理自体をユーザから隠蔽するものではなく, データ構造に依存する通信処理の煩雑さをユーザから隠そうとしている. ユーザは, データ分割の詳細を意識せずに高水準な表現によりデータ通信を記述できる. PETSc の性能については調査中であるが, 場合によっては CMC に PETSc を利用するコードを出力する機能を付加することも検討したい.

MATLAB を提供する Mathworks は, 複数の計算機による強調を可能にするツールとして Distributed Computing Toolbox (DCT) を提供している [13]. DCT を用いれば完全に独立なジョブを複数の計算機で実行するプログラム記述が容易に行える. しかしながら, 並列に処理するジョブ同士のデータ通信を記述することは困難で, 適用可能なアプリケーションの種類が非常に限られている.

6. まとめ

本稿では, MATLAB に基づく行列言語処理系である CMC の拡張, すなわち PSBLAS ルーチンを用いる並列プログラム生成への対応について述べた. 現時点での実装で並列化可能な行列計算アルゴリズムである (前処理無しの) CG 法や Bi-CGSTAB 法を用いて分散メモリ型計算機上で実測した結果, 出力される PSBLAS プログラムは MPI によるプログラムに匹敵するスケーラビリティを得ることが可能であることが確認された.

この度の実測で使用したプログラムは, 行列ベクトル積を基本とする比較的単純なものであった. PSBLAS ルーチンには係数行列が三角行列である系のソルバなども用意されており, これらを用いると反復解法に前処理を導入することもできる. 今後はより複雑なプログラムに対する適用可能性の調査や性能評価を行いたい. また, PETSc[12] や SuperLU[14] などのライブラリの導入の検討と実測評価も課題として挙げられる.

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) Whaley, R.C. and Dongarra, J.J.: Automatically Tuned Linear Algebra Software, *Proc. SC: High Performance Networking and Computing Conference* (1998).
3) 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).
4) 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).
5) Anderson., E., et al.: *LAPACK Users' Guide*, 3rd Edition, SIAM (1999).
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).
7) Choi., J., et al.: ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers, *4th Symp. on the Frontiers of Massively Parallel Computation*, pp.120–127 (1992).
8) Dongarra, J.J. and Whaley, R.C.: *A User's Guide to the BLACS v1.1*, LAPACK Working Note #94 (1997).
9) Filippone, S. and Colajanni, M.: PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices, *ACM Trans. Math. Softw.*, Vol.26, No.4, pp.527–550 (2000).
10) Quinn, M.J., et al.: Otter: Bridging the Gap between MATLAB and ScaLAPACK, *Proc. HPDC'98* (1998).
11) Ramaswamy, S., et al.: Compiling MATLAB Programs to ScaLAPACK: Exploiting Task and Data Parallelism, *Proc. IPPS'96*, pp.613–619 (1996).
12) Balay, S., et al.: Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries, *Modern Software Tools in Scientific Computing*, Brikhauser Press, pp.163–202 (1997).
13) Mathworks Inc.: *Distributed Computing Toolbox User's Guide*, Version 2(2005).
14) Demmel, J.W., et al.: *SuperLU Users' Guide*, http://crd.lbl.gov/~xiaoye/SuperLU/ (2003).