ハイパフォーマンス
コンピューティング 82－7
（2000．8．3）

異機種並列計算機における連立一次方程式ライブラリの性能評価

黒田久泰† 片桐孝洋†† 金田康正†

我々の開発したGMRES(m)法を利用した連立一次方程式ライブラリでは，実行性能が最大限になるような最適なパラメータを自動的に設定することができるように設計されている．例えば，問題の種別や実行プロセッサ台数などはライブラリ実行時になって初めてわかる性質である．本論文ではこのライブラリを利用する上でのインタフェースの概略を示し，いくつかの並列計算機上でその性能を調べた．その結果，自動選択されるパラメータは並列計算機の種類によって変わり，それぞれにおいて台数効果も得られることがわかった．

Performance Evaluation of Linear Equations Library on Parallel Computers

HISAYASU KURODA,† TAKAHIRO KATAGIRI†† and YASUMASA KANADA†

Our linear equations library which uses GMRES(m) method is designed to adjust some parameters automatically in order to get high performance. This adjustment which is decided at the time of executing library varies with a characteristic of the problems, the number of execution processors and so on. In this paper, we show the outline of our library interface. From the experimental results, we found that auto-tuned parameters depend on a architecture of parallel computers and our library achieves good speed-up on each parallel computer.

1. はじめに

ここ数年，並列計算機が広く普及し利用し易い環境が整備されつつある．しかしながら，並列プログラミングを研究の主体としていない実験系の研究者にとって，新たに並列プログラミングを習熟することは大変な労力を必要とする．実際，著者の所属する並列計算機の共同利用施設においても，ベクトル型計算機から並列計算機に移行した際に，性能が上がらずに困っていると相談に来る利用者が多い．

ベクトル型計算機の利用者に，並列計算機を使うにあたって薦める方法として，「コンパイラの自動並列化」を使う方法と「HPF」を使う方法がある．コンパイラの自動並列化の機能を使った場合には，逐次版と同じプログラムが利用できるという利点があるものの，ループ内に関してのみ並列化されることが多く，ループ長が大きい場合にしかうまく機能しないことが多い．また，ループ以外のところでの大域的な並列化が効かないため，一般的な解法に関してプロセッサ数に比例するような実行性能を得ることは非常に難しい．HPF(High Performance Fortran)を使った方法では，利用者側で指示文を適切に記述することの難しさに加え，コンパイラが最適なオブジェクトコードを出すことも難しいという問題がある．

実行性能を追求した方法として，MPI[1]やPVM[2]を利用した並列プログラムに書き換える方法があるが，これは最初に述べたように利用者に多大な労力を求める．それは多くの場合，ある問題を並列計算機で解かせる場合に，データ構造や解法アルゴリズムそのものを根本的に見直す必要があるからである．さらに，逐次の場合と比較して，アルゴリズムの選択やプログラミング技術の差異でプログラムの実行時間には大きな差が開くことが多く☆，性能が上がる保証はない．

このようなことを考えると，第三者によって構築された実行性能の高い並列ライブラリを使うことが考えられる．例えば，並列ライブラリとしてはScaLAPACK[3]やPLAPACK[4]などが挙げられる．しかし，並列ライブラリの利用がそのまま並列計算機を効率良く利用することに結び付くとは限らない．ライブラリの多くは，問題を解くために多くのパラメータを的確に指定しなくてはならない．コンパイル時に決まる

† 東京大学情報基盤センタースーパーコンピューティング研究部門
Computer Centre Division, Information Technology Center, the University of Tokyo
†† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Graduate School of Science, the University of Tokyo

☆ JSPP 並列ソフトウェアコンテストにおいても上位者と下位者との間ではプログラム実行時間に大きな差が生じやすい．

ループアンローリングの段数をどうするかという基本的なことから，反復法における前処理行列に何を適用するかといった応用的なことまで，ライブラリの実行性能に影響する要因として数多くのことが挙げられる．

実際，問題のサイズや性質が変わったり，同じ問題でも実行するPE台数を変更したりする場合に，パラメータ内容を変更すべきであることはすでに文献5で示されている．本稿では，4種類の並列計算機に本ライブラリを実装し，機種の違いによってどのようにパラメータ内容が異なっていくかについて述べる．

本報告におけるライブラリの大きな特徴は，利用者にパラメータを設定させるのではなく，ライブラリ自身が最適なパラメータを自動的に選択するという点である．そして，ライブラリ内にはコンパイル済のオブジェクトが大量に含まれており，プログラム実行時に，各ルーチンにかかる実行時間の計測を行ったり，残差ノルムの減少具合を見たりしながら，最適なオブジェクトコードを選択して実行するという方法を取っている．具体的には，GMRES法 (Generalized Minimal RESidual method)[6] の改良版であるGMRES(m) 法☆を使ったライブラリ ILIB_GMRESについての性能評価を行った．

2. ILIB_GMRES の概要

2.1 インタフェース

ILIBでは，ライブラリの実行性能を重視するとともに，使い易さも考慮して構築されている．

ここでは，ILIB_GMRESで提供されているインタフェース (「基本関数」「ベクトル操作関数」「行列操作関数」の3グループに別れる) について説明する．これらのインタフェース関数を通して，利用者からは見えない部分で，各PEへのデータ配置が行われるため，利用者はMPIライブラリや並列プログラミングの知識を必要としない．実際，これらのインタフェースを利用して記述したコードは，逐次版と並列版で同じソースコードを共有することができる．

基本関数

int IlibInitialize(int *argc, char ***argv, char *filename, char *help) 初期化のための関数．コマンドラインオプションを引き渡すことで，各種のパラメータを設定することができる．
int IlibFinalize() 終了のための関数．
int IlibGetSize(MPI_Comm mpi_comm, int *size) MPI_Comm_Size() に相当する．PE の個数を返す．
int IlibGetRank(MPI_Comm mpi_comm, int *rank) MPI_Comm_Rank() に相当する．ローカル PE のプロセッサ番号を返す．

ベクトル操作関数

int VecCreate(MPI_Comm mpi_comm, int ilib_decide, int size, Vec *v) ベクトルを生成する．
int VecDuplicate(Vec *a, Vec *b) 指定されたベクトルと同じサイズのベクトルを生成する．そのためベクトルのサイズを指定する必要がない．
int VecGetOwnershipRange(Vec *v, int *first_row, int *last_row) ローカル PE が保持しているベクトルのインデックスの範囲を取得する．
int VecSetValues(Vec *v, int m, int *im, double *values, InsertMode insertmode) ベクトルの複数個の要素に値を代入または加える．
int VecSetValue(Vec *v, int im, double value, InsertMode insertmode) ベクトルの 1 つの要素に値を代入または加える．
int VecAssemblyBegin(Vec *v) 全 PE 間で整合性を取る操作を開始する．
int VecAssemblyEnd(Vec *v) 全 PE 間で整合性を取る操作の終了を待つ．この操作の後に，Ilib_GMRES() 関数を呼ぶことができる．
int VecZeroEntrie(Vec *v) ベクトル内の要素をすべて 0 にする．ベクトルを生成した時にベクトルの要素は 0 に初期化されていない．
int VecDestroy(Vec *v) ベクトルの要素の値を保持していたメモリを解放する．
int VecView(Vec *v, Viewer viewer) ベクトルの要素を標準出力に表示する．

行列操作関数

int MatCreate(MPI_Comm mpi_comm, int m, int n, Mat *A) 行列を生成する．非零要素が 1 つもない状態で生成されるので，生成した時点では 0 に等しい．
int MatGetOwnershipRange(Mat *A, int *first_row, int *last_row) ローカル PE が保持している行列の行インデックスの範囲を取得する．
int MatSetValues(Mat *A, int m, int *im, int n, int *in, double *values, InsertMode insertmode) 行列に複数個の要素の値を代入または加える．
int MatSetValue(Mat *A, int im, int in, double value, InsertMode insertmode) 行列に 1 つ要素の値を代入または加える．
int MatAssemblyBegin(Mat *A, MatAssemblyType matassemblytype) 全 PE 間で整合性を取る操作を開始する．
int MatAssemblyEnd(Mat *A, MatAssemblyType matassemblytype) 全 PE 間での整合性を取る操作の終了を待つ．この操作の後に，Ilib_GMRES() 関数を呼ぶことができる．
int MatZeroEntrie(Mat *A) 行列の要素をすべて 0 にする．
int MatDestroy(Mat *A) 行列の要素の値を保持していたメモリを解放する．

☆ GMRES(m) 法は，GMRES 法を m 回の反復を周期としてリスタートを行う．リスタートを行うことで，必要となる計算量と記憶容量を抑えている．

2.2 メイン関数

ILIB_GMRES を呼び出す関数は次のとおりである.

int Ilib_GMRES(Mat *A, Vec *x, Vec *b)
GMRES(m) 法を用いて連立一次方程式 $Ax = b$ を解く.

自動的にパラメータの設定が行われるため, この 3 つの引数を与えるだけで良い. もし, 利用者自身で詳細にパラメータを与えたい場合には, コマンドラインオプションや ILIB 環境設定ファイルで指定することができる.

2.3 サンプルプログラム

ILIB_GMRES を利用する際のプログラム例を図 1 に示す.

```
#include "gmres.h"
void main(int argc, char *argv[])
{
   Mat A;  /* 行列 A を定義 */
   Vec x,b;  /* ベクトル x,b を定義 */

   IlibInitialize(&argc, &argv, NULL, NULL);
   IlibGetRank(NULL, &rank);  /* PE 番号を取得 */

   MatCreate(NULL, n, n, &A);  /* 行列 A を生成 */
   VecCreate(NULL, NULL, n, &x);
   VecDuplicate(&a,&b);

   if( rank==0 ){
      ·····    /* 係数行列 A に値を代入 */
      MatSetValue(&A, i, j, value, INSERT_VALUES);
      ·····    /* 右辺ベクトル b に値を代入 */
      VecSetValue(&b, i, value, INSERT_VALUES);
      ·····
   }
   MatAssemblyBegin(&A, MAT_FINAL_ASSEMBLY);
   VecAssemblyBegin(&b);
   MatAssemblyEnd(&A, MAT_FINAL_ASSEMBLY);
   VecAssemblyEnd(&b);

   Ilib_GMRES(&A, &x, &b);  /* GMRES(m) 法で解く */
   VecView(&x,stdout);  /* ベクトル x の要素を表示 */

   VecDestroy(&b);
   VecDestroy(&x);
   MatDestroy(&A);
   IlibFinalize();
}
```

図 1 ILIB_GMRES を使ったプログラム例

通常, 問題を生成する部分においては, 図 1 のように 1PE のみで生成する場合が多い. MatSetValue() と VecSetValue() の関数は, ローカル PE がその要素を保持するしないに関わらず, どの PE 上でも実行できる関数である. これらの関数でセットされた要素の値は, すぐには本来保存すべき場所に格納されずに, そのインデックスとともにローカル PE のメモリ上に保存されている. 全 PE で, 一斉に MatAssemblyBegin() および VecAssemblyBegin() の関数が呼ばれるとそこで全 PE にまたがってデータの再配置が行われる.

このようにライブラリ自身でデータの分散を行うので, プログラムの作成を容易に行うことができる.

2.4 実行時オプション

ここでは, 実行時オプションについて説明する.

基本設定

-max_time 制限時間 (秒)
指定された時間が経過すると反復を終了する.

-max_iter 最大反復回数
指定された反復回数が経過すると反復を終了する.

-atol 絶対誤差
指定された絶対誤差以内になると反復を終了する.

-rtol 相対誤差
指定された相対誤差以内になると反復を終了する.

-display 表示レベル
表示メッセージの出力量を調整する.

自動チューニング関連

-mat_type {0|1|*fast*|2|*compact*}
行列の格納形式を決定する. 1|*fast* ではループアンローリング向けの各行における非零要素数を固定と見なしたもの (要素のない部分には 0 のデータが書き込まれる). 2|*compact* はメモリの節約を行うための行圧縮格納形式. 行列の各行における非零要素数のばらつきが少ない場合には前者の方は効率が良い.

-unroll アンローリングの種類
行列の各行における非零要素の個数が 9 個以下の場合には, ループを全て展開したコードをそれぞれ用意している. 列方向についても, 上記のそれぞれの場合において 1,2,3,4,8 段展開したものを用意している (合計 45 個のルーチン). また, これと同じように展開したコードで配列参照をプリフェッチにした 36 個のルーチンがある. 1 行の非零要素の個数が 10 個以上の場合には, 可変個数に対応したルーチンが呼ばれるが 8 段まで展開したコードが 8 種類ある. 結局, 合計 89 個のルーチンがあり, 反復が開始された後はその内の 1 つのコードだけを繰り返し実行する.

-comm {0|1|*all*|2|*bcast*|3|*isend*|4|*irecv*|5|*send*}
疎行列ベクトル積における通信方式を決める.

-restart {0|1 以上の任意の整数 }
リスタート周期の値を決める.

-restartflag {0|*variable*|1|*fix*}
リスタート周期間隔は可変かそれとも固定か.

-orth {0|1|*cgs*|2|*mgs*|3|*ircgs*}
直交化の際のアルゴリズムを決める.
1|*cgs*：古典的グラムシュミット
2|*mgs*：修正グラムシュミット
3|*ircgs*：反復改良グラムシュミット

-precond {0|1|*none*|2|*poly*|3|*ilu*|4|*modilu*}
前処理行列を決める.
1|*none*：スケーリング前処理のみ
2|*poly*：多項式前処理行列 [7]
3|*ilu*：ブロック不完全 LU 分解前処理
4|*modilu*：上記の対角要素を修正したもの

「-max_time」オプションで指定された時間が経過すると反復を終了して途中までの値を返すようにしている．多くの反復ライブラリでは，収束しなかった場合の回避策として最大反復回数を指定できる場合が多い．しかし，問題のサイズによって1反復にかかる時間は大きく変わるため，小さい反復回数でも長時間かかる場合もある．また，並列計算機のバッチジョブでは，制限時間が設けられていることが多く，制限された時間内に，何度もルーチンを呼び出す場合には，時間を制限することで，できるだけ無駄なく精度の良い結果を得ることが期待できる．

自動チューニング関連項目では，オプションを設定しなかった場合には，0が設定されたと見なしその項目は自動チューニングの対象となる．

しかし，すべてのパラメータが自動チューニングの対象になるのではない．例えば，「-precond modilu」で指定されるオプションは不完全LU分解前処理の対角要素の値を補正して使うためのものであるが，自動チューニングの選択項目としては入っていない．これは，前処理行列の多くはその準備のためにかなりの時間を必要とするため，試験する前処理の種類を絞っているためである．本ライブラリで利用する多項式行列前処理はその準備にほとんど時間を必要としない上に，選択される場合が多いため対象項目として入れてある．結局，自動チューニングにかかる時間そのものも実行時間に含まれるため，短時間でその効果の善し悪しがわかり，かつ，判断にかかった時間以上の効果が得られるものしか対象にはできない．

「-comm」オプションの自動チューニングでは，古典的グラムシュミットと修正グラムシュミットの実行時間の短い方を選択する．この選択は，利用する並列計算機と実行プロセッサ台数によって変化することが多い．問題によっては，古典的グラムシュミットで収束しない場合があるため，収束が悪くなった場合には反復改良グラムシュミットを選択するようになる．このように，一度選択したアルゴリズムを実行状況の変化によって選択し直すことがあるのも本ライブラリの特徴の一つである．

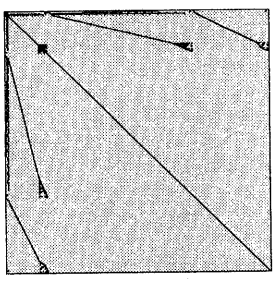

図2 問題2の非零要素の分布

3. 性能評価

ここでHITACHI SR2201☆1, HITACHI SR8000☆2, NEC SX-4/64M2☆3, FUJITSU VPP800/63☆4の各並列計算機上で本ライブラリを実装し，その性能について評価した結果を示す．本ライブラリは，ベクトル演算，行列演算の部分は実行性能を考慮してFortran言語で記述されており，それ以外はC言語で記述されている．疎行列ベクトル積におけるループアンローリングルーチンは，コンパイラでループアンローリングを行わないようなオプションを付けてコンパイルしている．なお，数値実験は以下の条件で行った．

- 収束条件：相対残差 $||Ax-b||/||b|| < 1.0\times10^{-12}$
- 初期近似解：$x_0 = (0,0,\cdots,0)^T$
- 計算精度：倍精度

3.1 問題設定

ここでは次の2つの問題で性能を測定した．

(1) 問題1 (サイズ：大，非零要素：PE間で均等)
領域 $\Omega=[0,1]\times[0,1]\times[0,1]$ における楕円型偏微分方程式の境界値問題

$$\begin{aligned} -u_{xx}-u_{yy}-u_{zz}+Ru_x &= g(x,y,z) \\ u(x,y)|\partial\Omega &= 0.0 \end{aligned}$$

右辺は厳密解が $u=e^{xyz}\sin(\pi x)\sin(\pi y)\sin(\pi z)$ となるように定める．領域を $80\times80\times80$ のメッシュで7点中心差分によって離散化した．得られた連立1次方程式の次元は512,000である．ここではR=1.0とした．

(2) 問題2 (サイズ：小，非零要素：PE間で不均等)
Electronic circuit designの問題．これはMATRIX MARKET(http://math.nist.gov/MatrixMarket/)で提供されている問題で，セット名はHAMM，行列名はmemplusである．連立1次方程式の次元は17,758で，非零要素の個数は99,147である．1行当たりの平均非零要素数は5.6で，最大353，最小1となっている．非零要素の分布を**図2**に示す．なお，右辺ベクトルの要素の値はすべて1とした．

☆1 東京大学情報基盤センターが所有している1024PEのSR2201のうち128PEを使用した．各PEの理論ピーク性能は300MFlops，PE間は三次元クロスバ網で結合されており，その最大転送性能は300Mbyte/秒である．

☆2 東京大学情報基盤センターが所有している128ノードのSR8000のうち16ノードを使用した．各ノードは8つのIPから構成され，各IPの理論ピーク性能は1GFlops(1ノードでは8GFlops)，ノード間は三次元クロスバ網で結合されており，その最大転送性能は片方向1Gbyte/秒，双方向2Gbyte/秒である．

☆3 大阪大学サイバーメディアセンターが所有している64PEのSX-4/64M2のうち16PEを使用した．各PEの理論ピーク性能は2GFlopsである．

☆4 京都大学大型計算機センターが所有している63PEのVPP800/63のうち32PEを使用した．各PEの理論ピーク性能は8GFlops，PE間はクロスバ網で結合されており，その最大転送性能は3.2Gbyte/秒である．

3.2 結　　果

ここでは比較の対象として,「自動チューニングなし」の場合のパラメータとしては下記のものを選んだ.

- 行列格納形式：1 行の要素数を固定した形式
- アンローリング：なし
- 通信方式：MPI_Allgather を使う方式
- 最大リスタート周期：30
- リスタート周期の値：固定
- 直交化：反復改良グラムシュミット
- 前処理：なし

各問題における実行時間 (単位:秒) と自動選択された方式を**表 1**～**表 8** に示す. 表の中の語句の説明は脚注に示す☆.

表 1　問題 1 (HITACHI SR2201)

PE 台数	8	16	32	64	128
自動チューニングなし					
反復回数	1265	1265	1265	1265	1265
実行時間	187.6	112.7	87.6	73.3	67.3
自動チューニングあり					
反復回数	288	300	417	417	417
合計時間	72.1	37.3	12.6	7.4	5.1
Unrolling	N(1,7)	P(1,7)	P(1,7)	P(1,7)	P(1,7)
通信方式	Isend	Isend	Isend	Isend	Isend
直交化方式	MGS	CGS	CGS	CGS	CGS
前処理方式	BILU	BILU	POLY	POLY	POLY
速度向上比	2.6	3.0	7.0	9.9	13.2

4. 考　　察

問題 1 については, まず機種が異なると選択されるアンローリング方式にも違いが出ている. SR2201 はプリフェッチコードが選択されており, SX-4 では列方向に 3 段アンローリングしたコードが選択されている. 通信方式では非同期方式を使うものと同期方式を使うものに二分されたことは興味深い. 直交化方式は SR2201 の PE=8 以外では, すべて CGS が選択された. 前処理行列は PE 台数が増えるに従って, 多項式前処理行列 (POLY) が適用される傾向にあるが, 機種によってその振舞が多少異なっている. 全ての機種において, 前処理方式としてブロック不完全 LU 分解前処理 (BILU) が選択された場合に, 速度向上比があまりよくない結果が出ている. ライブラリ側では, BILU は収束までの反復回数を大きく減らすことが見込めると判断して選択しているものの, 実際には前進代入・後退代入の計算が今回のベクトル向き並列計算機には実行時間から見て不向きであるため速度向上には継っていない. 前処理方式を選ぶ基準についてはまだまだ議論の余地が残っている.

問題 2 については, アンローリング方式については問題 1 と同様に機種によって違いが出ている. 一般にアンローリング段数は大きいほど効率が良いと考えら

表 2　問題 1 (HITACHI SR8000)

IP 台数	8	16	32	64	128
自動チューニングなし					
反復回数	1265	1265	1265	1265	1265
実行時間	78.0	48.4	31.6	27.3	26.2
自動チューニングあり					
反復回数	288	300	417	417	417
合計時間	37.9	20.0	6.3	4.1	3.9
Unrolling	N(1,7)	N(1,7)	N(1,7)	N(1,7)	N(1,7)
通信方式	Irecv	Irecv	Isend	Isend	Irecv
直交化方式	CGS	CGS	CGS	CGS	CGS
前処理方式	BILU	BILU	POLY	POLY	POLY
速度向上比	2.1	2.4	5.0	6.7	6.7

表 3　問題 1 (NEC SX-4/64M2)

PE 台数	2	4	8	16
自動チューニングなし				
反復回数	1265	1265	1265	1265
実行時間	729.1	553.1	287.7	147.7
自動チューニングあり				
反復回数	279	417	288	300
合計時間	457.6	28.7	119.3	61.0
Unrolling	N(3,7)	N(3,7)	N(3,7)	N(3,7)
通信方式	Irecv	Send	Send	Send
直交化方式	CGS	CGS	CGS	CGS
前処理方式	BILU	POLY	BILU	BILU
速度向上比	1.6	19.3	2.4	2.4

表 4　問題 1 (FUJITSU VPP800/63)

PE 台数	2	4	8	16	32
自動チューニングなし					
反復回数	1265	1265	1265	1265	1265
実行時間	261.1	132.7	68.6	36.4	20.6
自動チューニングあり					
反復回数	279	417	288	300	417
合計時間	107.9	7.6	27.5	14.1	1.1
Unrolling	N(1,7)	N(1,7)	N(1,7)	N(1,7)	N(1,7)
通信方式	Isend	Send	Send	Send	Send
直交化方式	CGS	CGS	CGS	CGS	CGS
前処理方式	BILU	POLY	BILU	BILU	POLY
速度向上比	2.4	17.5	2.5	2.6	18.7

☆ 合計時間：自動チューニングにかかる時間も含めた全体の時間
Unrolling：アンローリング方式. N はプリフェッチあり P はプリフェッチなし. () 内の数字は列方向, 行方向の展開数
通信方式：Send は Send と Recv のペアを使う. Isend は Isend-)Irecv の順. Irecv は Irecv-)Isend の順. M_All は MPI_Allgather. One_A は 1PE に集約後 MPI_Bcast を使う.
直交化方式：CGS は古典的グラムシュミット. MGS は修正グラムシュミット.
前処理方式：BILU はブロック不完全前処理行列. POLY は行列多項式前処理.
速度向上比：自動チューニングなしと比べた場合の速度向上比

表 5 問題 2 (HITACHI SR2201)

PE 台数	8	16	32	64	128
自動チューニングなし					
反復回数	775	754	803	715	819
実行時間	101.0	49.9	27.5	13.6	9.4
自動チューニングあり					
反復回数	286	286	286	286	286
合計時間	55.1	27.9	14.8	8.2	5.2
Unrolling	N(1,3)	N(1,7)	N(1,3)	N(1,6)	N(1,7)
通信方式	Irecv	Send	Send	M_All	M_All
直交化方式	CGS	CSG	CGS	CGS	CGS
前処理方式	POLY	POLY	POLY	POLY	POLY
速度向上比	1.8	1.8	1.9	1.7	1.8

表 6 問題 2 (HITACHI SR8000)

IP 台数	8	16	32	64	128
自動チューニングなし					
反復回数	821	799	769	808	769
実行時間	191.0	57.5	76.4	47.5	23.4
自動チューニングあり					
反復回数	286	286	286	286	286
合計時間	89.4	42.4	62.4	36.6	19.1
Unrolling	N(1,2)	N(1,1)	N(1,1)	N(1,6)	N(1,6)
通信方式	Irecv	Irecv	Irecv	Irecv	M_All
直交化方式	CGS	CGS	CGS	CGS	CGS
前処理方式	POLY	POLY	POLY	POLY	POLY
速度向上比	2.1	1.4	1.2	1.3	1.2

表 7 問題 2 (NEC SX-4/64M2)

PE 台数	2	4	8	16
自動チューニングなし				
反復回数	859	715	821	835
実行時間	44.8	17.8	12.1	7.7
自動チューニングあり				
反復回数	286	286	286	286
合計時間	29.6	16.7	9.1	4.9
Unrolling	N(1,8)	N(1,8)	N(1,8)	N(1,8)
通信方式	Irecv	Irecv	M_All	One_A
直交化方式	CGS	CGS	CGS	CGS
前処理方式	POLY	POLY	POLY	POLY
速度向上比	1.5	1.1	1.3	1.6

れるが，疎行列ベクトル積においては，ループ長はそれほど長くはならないため (1 行あたりの要素数は数十以下である)，係数行列の非零要素の分布によって最適なオブジェクトコードを選択した方が良い．通信方式では MPI_Allgather を使った方式が 4 箇所で表れている．本ライブラリでは，非零要素の分布を調べてできるだけ通信量が少なくなるような通信方法を利用しているが，それでも，メーカー提供の全対全通信である MPI_Allgather の方が選択されることがある．これは，本ライブラリではネットワークの結合網の形状や通信性能まで考慮していないことによる．

どの機種においても，問題 2 のような比較的小さいサイズの問題では，自動チューニングの効果はあまり出てこない．しかし，問題 1 のような大きな問題では，自動チューニングの効果が大きくなることがわかる．

表 8 問題 2 (FUJITSU VPP800/63)

PE 台数	2	4	8	16	32
自動チューニングなし					
反復回数	803	819	807	824	758
実行時間	7.5	4.0	2.2	1.3	0.94
自動チューニングあり					
反復回数	286	286	286	286	286
合計時間	7.0	3.8	2.1	1.2	0.79
Unrolling	N(1,1)	N(1,1)	N(1,1)	N(1,1)	N(1,1)
通信方式	Isend	Send	Send	Send	Send
直交化方式	CGS	CGS	CGS	CGS	CGS
前処理方式	POLY	POLY	POLY	POLY	POLY
速度向上比	1.1	1.1	1.0	1.1	1.2

5. ま と め

本ライブラリは，http://www.hints.org/で公開している．専門知識のない利用者にとっても，多くのパラメータを自動設定してくれるために，容易にこれらのライブラリが利用できる．また，問題の性質を理解している利用者にはついては，パラメータを個別に設定することで性能を向上させることが可能である．

参 考 文 献

1) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. Int J. Supercomputer Applications. 8(3/4). (1994)
2) A.Geist, A.Beguelin, J.Dongarra, W.Jiang, R.Manchek, V.Sundera: PVM: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press. (1994)
3) L.Blackford, J.Choi, A.Cleary, E.D'Azevedo, J.Demmel, I.Dhillon, J.Dongarra, S.Hammarling, G.Henry, A.Petitet, K.Stanley, D.Walker, R.Whaley: ScaLAPACK Users' Guide. SIAM, Philadelphia, PA. (1997)
4) Robert A. van de Geijn: Using PLAPACK. The MIT Press. (1997)
5) 黒田久泰，金田康正：自動チューニング機能付き並列疎行列連立一次方程式ソルバの性能．IPSJ SIG Notes, 99-HPC-76, pp.13 – 18 (1998)
6) Y.Saad and M.H.Schultz: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Scientific and Statistical Computing, No.7, pp. 856–869 (1986)
7) O.G.Johnson, C.A.Micchelli, G.Paul: Polynomial Preconditioners for Conjugate Gradient Calculations. SIAM J. Numer. Anal., Vol.20,No.2 (1983).