OpenMP を用いた並列反復法ライブラリの性能評価

小武守 恒（JST, 東大）　長谷川 秀彦（筑波大学）　西田 晃（東京大学）

現在開発中の複数の反復解法、前処理、格納形式を容易に組み合わせて使用できる反復法ライブラリの概略を示す。反復解法において核となるのは行列ベクトル積であり、その性能はデータの格納形式と計算環境に強く依存する。そこで、3 つの格納形式（CRS, BSR, DIA）に対し共有メモリ並列計算機上で OpenMP を用いて並列化した行列ベクトル積の性能を調べた。その結果、格納形式によって性能が大きく異なること、場合によっては格納形式を変換して使用したほうが良いことが分かった。

Performance Evaluation of a Parallel Iterative Method Library using OpenMP

H. Kotakemori（JST, Univ. of Tokyo）H. Hasegawa（Univ. of Tsukuba）
and A. Nishida（Univ. of Tokyo）

We show the outline of the iterative method library that can be used by combining two or more iterative methods, preconditioners, and the storage formats. A kernel operation the iterative methods is a matrix–vector product. The performance of a matrix–vector product strongly depends on the data storage formats and the computer environments. Then, we examine the performance of the parallel matrix–vector products using OpenMP for three storage formats (CRS, BSR and DIA) on the shared memory parallel computers. As a result, we found that in some cases we should use an optimal storage format with data conversion.

1. はじめに

大規模疎行列に対する線型方程式 $Ax = b$ の高速解法は科学、工学における計算で重要である。解くべき問題に依存して、反復解法と前処理の選択によっては、収束するまでに多くの反復回数が必要となったり、収束しない場合がある。また、これらのアルゴリズムの計算時間は、計算環境とデータの格納形式にも強く依存する。この問題を解決するためには、反復解法、前処理の組み合わせ、データ格納形式を様々な計算環境上でプログラムを変更することなく利用できるような反復解法ライブラリが必要である。われわれは、「大規模シミュレーション向け基盤ソフトウェアの開発」プロジェクト[1] として複数の反復解法、前処理、格納形式を組み合わせて使用することができるライブラリの開発を行っている。本報告では、共有メモリ型並列計算機上におけるわれわれの反復解法ライブラリの性能について議論する。

行列ベクトル積は、反復解法における最も重要なカーネルで、その性能が反復解法の性能に重大な影響を与える。問題と計算環境によってどの格納形式が効率が良いかは大きく異なる。ユーザが異なった格納形式に対して自分のプログラムを変更するのは重荷である。ユーザの使い勝手を向上させるには、格納形式間の変換サブルーチンが必要になる。われわれは、行列ベクトル積がある回数以上行われるならば格納形式の変換を行うことで、より高速に行列ベクトル積が実行できることを示す。

2. ライブラリの概要

2.1 解法・前処理・格納形式

対象とする反復解法は、表 1 に示す通り大規模な実行列用で定常（SOR 等）、非定常（GPBiCG 等）など 10 種類程度を採用している。前処理としては、表 2 に示す通りよく知られているスケーリング、不完全 LU 分解などに加えて、定常反復解法に有効な $I + S$ 型[2]、smoothed aggregation に基づく代数的マルチグリッド SA-AMG[3]、SOR などの反復法を用いる Hybrid 法[4]、A-直交化に基づき逆行列 A^{-1} そのものを近似する近似逆行列 SAINV[5] などを採用している。データの格納形式は表 3 に示す通り CRS など 10 種類程度を採用している。

これらの解法、前処理、格納形式が容易に組み合わせて使用できるようになっている。

2.2 主な関数

表 4 に主な関数を示す。求解までの基本的な流れは、行列の作成、右辺、初期解ベクトルの作成（あるいは、ファイルから行列データの読み込み）、求解となる。求解を呼び出す C 言語の関数は次のとおりである。

```
int ssi_solve(SSI_MATRIX *A, SSI_SCALAR b[],
  SSI_SCALAR x[], SSI_SCALAR params[],
  int options[], SSI_SCALAR status[])
```

入力として、行列 A、右辺ベクトル b、初期近似解 x、収束判定基準、反復解法・前処理に必要な実数パ

表 1 反復解法

非定常解法	CG 法
	BiCG 法
	CGS 法
	BiCGSTAB 法
	BiCGSTAB(l) 法
	GPBiCG 法
	QMR 法
	Orthomin(m) 法
	GMRES(m) 法
定常解法	Jacobi 法
	Gauss-Seidel 法
	SOR 法

表 2 前処理

Jacobi
SSOR
ILU(k)
I+S
SA-AMG
Hybrid
SAINV

表 3 格納形式

Dense	(DNS)
Coordinate	(COO)
Compressed Row Storage	(CRS)
Compressed Column Storage	(CCS)
Modified Compressed Sparse Row	(MSR)
Diagonal	(DIA)
Ellpack-Itpack generalized diagonal	(ELL)
Jagged Diagonal	(JDS)
Block Sparse Row	(BSR)
Block Sparse Column	(BSC)
Variable Block Row	(VBR)

表 4 主な関数

関数名	機能
ssi_matrix_create	指定された格納形式の行列を作成
ssi_vector_create	ベクトルの作成
ssi_matrix_input	ファイルから行列データの読み込み
ssi_matrix_convert	指定の格納形式へ行列を変換
ssi_solve	求解

ラメータ params、反復解法・前処理などを選択する options を指定する。params と options は以下のように配列に値を代入する。

```
params[SSI_PARAMS_RESID] = 1.0e-12;
options[SSI_OPTIONS_MAXITER] = 1000;
options[SSI_OPTIONS_SOLVER] = SSI_SOLVER_BICGSTAB;
options[SSI_OPTIONS_PRECON] = SSI_PRECON_TYPE_NONE;
```

反復回数、実行時間などは status に出力される。

3. OpenMP を用いた並列実装

OpenMP による並列化は、並列化すべきところにディレクティブを挿入し、並列化コンパイラがディレクティブの指示に基づいて並列化を行う。

3.1 CRS (Compressed Row Storage)

CRS 形式は 3 つの配列 (ptr,index,value) に格納する。nnz を行列 A の非零要素数とする。長さ nnz の倍精度配列 value は行列 A の非零要素の値を行方向に沿って格納する。長さ nnz の整数配列 index は配列 value に格納された非零要素の列番号を格納する。長さ $n+1$ の整数配列 ptr は配列 value と index の各行の開始位置を格納する。CRS 形式の行列ベクトル積 $y = Ax$ のコードを以下に示す。

```
#pragma omp parallel for private(i,j,t)
  for(i=0; i<n; i++)  {
    t = 0.0;
    for(j=A.ptr[i];j<A.ptr[i+1];j++)
      t += A.value[j] * x[A.index[j]];
    y[i] = t;
  }
```

3.2 BSR (Block Sparse Row)

BSR では行列を $r \times c$ の大きさの部分行列（ブロックと呼ぶ）に分解する。BSR は CRS と同様の手順で非零ブロック（少なくとも 1 つの非零要素が存在する）を格納する。$nr = n/r$、$nnzb$ を A の非零ブロック数とする。BSR は 3 つの配列 (bptr, bindex, value) に格納する。長さ $nnzb \times r \times c$ の倍精度配列 value は非零ブロックの全要素を格納する。長さ $nnzb$ の整数配列 bindex は非零ブロックのブロック列番号を、長さ $nr+1$ の整数配列 bptr は配列 bindex のブロック行の開始位置を格納する。

ブロックサイズが 2×2 (i.e. $r = 2, c = 2$) の BSR 形式の行列ベクトル積のコードを以下に示す。大きな r はベクトル x の要素に対するロード命令数を減少させ、大きな c は内側のループ j のアンローリングとして作用する。

```
#pragma omp parallel for private(i,j,jj,t0,t1)
  for(i=0; i<nr; i++)  {
    t0 = t1 = 0.0;
    for(j=A.bptr[i];j<A.bptr[i+1];j++)  {
      jj  = A.bindex[j];
      t0 += A.value[j*4+0] * x[jj*2+0]
      t1 += A.value[j*4+1] * x[jj*2+0]
      t0 += A.value[j*4+2] * x[jj*2+1];
      t1 += A.value[j*4+3] * x[jj*2+1];
    }
    y[2*i+0] = t0; y[2*i+1] = t1;
  }
```

3.3 DIA (Diagonal Storage)

DIA は 2 つの配列 (index, value) に格納する。長さ $nnd \times n$ の倍精度配列 value は行列 A の非零な対角を格納する。nnd は非零な対角の本数である。長さ nnd の整数配列 index は主対角から各対角へのオフセットを格納する。DIA 形式の行列ベクトル積のコードを以下に示す。

```
#pragma omp parallel for private(i)
  for(i=0; i<n; i++)
    y[i] = 0.0;
#pragma omp parallel for
    private(i,j,k,is,ie,n1,n2,jj,ii)
  for(k=0;k<threads;k++)  {
    n1 = n/threads; n2 = n%threads;
    is = k<n2 ? (n1+1)*k : n1*k+n2;
    ie = k<n2 ? is+n1+1  : is+n1;
    for(j=0;j<nnd;j++)  {
      jj = A.index[j];
      ii = _max(is,-jj)-_min(ie,n-jj);
      for(i=_max(is,-jj);i<_min(ie,n-jj);i++)
        y[i] += A.value[nn*k*nnd + j*nn + ii++]
              * x[jj+i];
    }
  }
```

われわれの実装では、2 スレッドで実行するとき

表 5　実行環境

環境	SGI Altix3700
CPU	Itanium2 1.3GHz × 32
キャッシュ	L1D:16KB L2:256KB L3:3MB
メモリ	32GB
コンパイラ	Intel C/C++ 8.1
最適化オプション	-O3

行列 A が

$$A = \begin{pmatrix} 11 & & & \\ 21 & 22 & & \\ & 32 & 33 & \\ 41 & & 43 & 44 \end{pmatrix}$$

ならば、従来では `A.value` = {0, 0, 0, 41, 0, 21, 32, 43, 11, 22, 33, 44} と格納するところを `A.value` = {0, 0, 0, 21, 11, 22, 0, 41, 32, 43, 33, 44} と格納する。すなわち、行ブロック分割された非零な対角に対してそのブロックごとに DIA 形式で格納するように修正した。

3.4 格納形式変換

SPARSKIT[7] を参考に CRS から他の形式への変換ルーチンを実装した。しかし、OpenMP を用いた並列化にはいくつかの修正が必要であった。

たとえば、CRS から BSR への並列格納形式変換を考える。BSR は 3 つの配列 `bptr`, `bindex`, `value` を持つ。逐次実装の場合、最初に非零ブロック数 ($nnzb$) を決め、次に配列を確保する。そして、各配列は同じループ内でデータが格納される。しかしながら、並列実装では、各スレッドは配列 `bindex` と `value` のデータを格納する開始位置を知っている必要があるため、配列 `bptr` の値が最初に必要となる。

4. 性 能 評 価

数値実験は表 5 に示す環境で行った。16 スレッド以下の場合は `dplace` コマンドを用いて 1 ノードに 1 スレッドを割り当てるようにした。16 スレッドまではメモリとのバスを 1 つの CPU で占有できる。16 スレッドをこえると 2 つの CPU でバスを共有するため転送スピードが低下する。

係数行列 A は、3 次元ポアソン方程式を有限要素法で離散化して得らた。行列の次数は 1,000,000 とし、CRS、BSR_31、DIA で格納した場合に必要となるメモリはそれぞれ約 307、395、206MB である。右辺ベクトルは解ベクトルがすべて 1 となるように設定し、収束判定基準は 10^{-12} とした。

4.1 各反復解法の並列性能

格納形式を CRS、前処理なしの場合の各反復解法の実行時間（秒）と速度向上を表 6 に示す。ite. は反復回数である。16 スレッドまではほぼ理想的な速度向上を示している。32 スレッドでは、24.24〜27.36 の速度向上を示している。反復回数は CG と BiCG が 395 回、Orthomin(6) が 364 回、そのほかの解法は約 250 回程度である。

4.2 行列ベクトル積の並列性能

行列ベクトル積の並列化性能を確認するため 1,000 回繰り返した実行時間 (秒) を表 7 に示す。各格納形式の中で実行時間が最も少ないものを太字で示している。

BSR に対して、最適なブロックサイズは 3×1 であった。8 スレッド以下では BSR_31 が、16 と 32 スレッドでは DIA が一番性能がよかった。

多くの場合、ほぼ理想的な速度向上をしている。BSR の 32 スレッドに対する速度向上は 16 スレッドに対する値の 2 倍よりもはるかに小さい。これは、各ノードが 2 つのプロセッサでバスが共有されていることから生じているためである。

DIA は 4 スレッド以上でスーパーリニアな速度向上が見受けられる。特に、16 スレッド以上では各スレッド数 16、32 を大幅に上回る 37.84、63.51 となっている。これは、キャッシュヒット率が改善されたためであると推測される。

これらの結果から、(1) 行列格納形式によって性能が大きく異なる (2) 最適な形式はメモリアーキテクチャ(キャッシュと共有バス) が影響していることが分かる。

4.3 格納形式変換のコスト

CRS 形式で行列 A が与えられていると仮定する。もし、A に対する行列ベクトル積において CRS よりも性能が良い格納形式が分かるならば、行列ベクトル積を高速化するために行列 A をその形式に変換するという手がある。

T_{crs} と T_{tgt} をそれぞれ CRS 形式と目的の形式の行列ベクトル積の実行時間とする。T_{conv} を CRS から目的の形式への変換時間とする。反復回数の閾値 N_{th} を $N_{th} = \lceil T_{conv}/(T_{crs} - T_{tgt}) \rceil$ と定義する。行列ベクトル積を N_{th} 回以上行うならば、目的の形式を、そうでなければ CRS のままの方が良い。

表 8 に CRS から目的の形式への変換時間 (ミリ秒) と反復回数の閾値 N_{th} を示す。"–"は変換すると必ず性能が低下することを意味する。

BSR に対して、1 から 16 スレッドでは T_{conv}、T_{crs}、T_{bsr} が同程度の比率で減少しているため N_{th} の値にあまり大きな変化はなく約 60 回ぐらいであるが、32 スレッドでは N_{th} は 4306 回になる。これは、16 スレッドと比べてメモリバスの競合で T_{conv} の値が大きく、CRS と BSR の性能にあまり差がないため $T_{crs} - T_{bsr}$ の値が小さくなっているためである。

DIA に対して、8 スレッド以上では nnd の値を求める部分でメモリアクセスが集中するため変換時間にあまり変化がなく、N_{th} の値は 16 スレッドを除いて約 80 回ぐらいである。

表 6 格納形式 CRS、前処理なしの場合の各反復解法の実行時間（秒）(カッコ内は Speed-up).

# of threads		1		2		4		8		16		32	
反復解法	ite.												
CG	395	61.46	(1.00)	30.96	(1.99)	15.60	(3.94)	7.47	(8.22)	3.83	(16.05)	2.38	(25.78)
BiCG	395	119.18	(1.00)	60.12	(1.98)	30.37	(3.92)	15.12	(7.88)	7.77	(15.33)	4.92	(24.24)
CGS	287	90.77	(1.00)	45.82	(1.98)	22.76	(3.99)	11.33	(8.01)	5.43	(16.73)	3.39	(26.79)
BiCGSTAB	254	75.16	(1.00)	38.07	(1.97)	19.74	(3.81)	8.99	(8.36)	4.83	(15.57)	2.93	(25.61)
BiCGSTAB(2)	256	79.40	(1.00)	42.70	(1.86)	21.38	(3.71)	10.00	(7.94)	5.21	(15.24)	3.22	(24.68)
GPBiCG	254	95.33	(1.00)	47.07	(2.03)	24.10	(3.95)	11.37	(8.39)	5.68	(16.79)	3.85	(24.75)
QMR	287	95.39	(1.00)	47.52	(2.01)	23.85	(4.00)	11.51	(8.29)	5.60	(17.03)	3.53	(27.02)
Orthomin(6)	364	101.79	(1.00)	51.65	(1.97)	26.45	(3.85)	13.03	(7.81)	6.30	(16.15)	3.72	(27.36)

表 7 行列ベクトル積 1,000 回の実行時間 (秒)(カッコ内は Speed-up).

# of threads	1		2		4		8		16		32	
CRS	149.50	(1.00)	74.96	(1.99)	37.43	(3.99)	18.76	(7.97)	9.51	(15.72)	4.97	(30.07)
BSR_31	**85.60**	(**1.00**)	**43.25**	(**1.98**)	**21.53**	(**3.97**)	**10.92**	(**7.84**)	5.63	(15.20)	4.87	(17.58)
DIA	178.50	(1.00)	89.19	(2.00)	44.34	(4.03)	16.40	(10.88)	**4.72**	(**37.84**)	**2.81**	(**63.51**)

表 8 CRS から目的の形式への変換時間 (ミリ秒)(カッコ内は反復回数の閾値 N_{th}).

# of threads	1		2		4		8		16		32	
BSR_31	3370.8	(53)	1720.3	(55)	1073.5	(68)	478.6	(61)	303.8	(79)	439.2	(4306)
DIA	907.4	(–)	485.6	(–)	270.3	(–)	178.0	(76)	165.7	(35)	178.8	(83)

どの反復解法も反復回数は 200 回以上なので CRS から BSR、DIA に変換すればより高速に解を求めることができる。

5. ま と め

本報告では、われわれのライブラリについて反復解法の並列化性能と 3 つの格納形式に対する行列ベクトル積の並列性能について述べた。格納形式によって性能が大きく異なることが分かった。メモリバスが 1CPU で占有される場合は、どの格納形式も速度向上は好ましい値となった。しかし、バスが 2CPU で共有される場合は、BSR 形式は大幅な性能低下を起こすことが分かった。また、今回の行列では行列ベクトル積が 90 回以上行われるならば格納形式の変換を行うことで、より高速に行列ベクトル積が実行できることが分かった。これらのことより、メモリ (具体的にはキャッシュとメモリバス) アーキテクチャが行列ベクトル積の性能に多くに影響を及ぼすことが確認された。したがって、様々な格納形式が利用可能であることは重要である。

謝辞 本報告は、科学技術振興機構（JST）戦略的創造研究推進事業（CREST）「大規模シミュレーション向け基盤ソフトウェアの開発」プロジェクトの一部として実施した。

参 考 文 献

1) 西田晃. SSI: 大規模シミュレーション向け基盤ソフトウェアの概要. 情報処理学会研究報告, 2004- HPC-098, pp. 25-30, April 2004.
2) Toshiyuki Kohno, Hisashi Kotakemori and Hiroshi Niki. Improving the Modified Gauss-Seidel Method for Z-matrices. Linear Algebra and its Applications, Vol. 267, pp. 113–123, 1997.
3) 藤井昭宏, 西田晃, 小柳義夫. 領域分割による並列 AMG アルゴリズム. 情報処理学会論文誌, Vol. 44, No. SIG6(ACS1), pp. 1–8, 2003.
4) 阿部邦美, 張紹良, 長谷川秀彦, 姫野龍太郎. SOR 法を用いた可変的前処理付き一般化共役残差法. 日本応用数理学会論文誌, Vol. 11, No. 4, pp. 157–170, 2001.
5) R. Bridson and W.-P. Tang. Refining an approximate inverse. J. Comput. Appl. Math., Vol. 123, pp. 293–306, 2000.
6) R. Barrett, et al.. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, 1994.
7) Y. Saad. SPARSKIT: a basic tool kit for sparse matrix computations, version 2, June 1994.
8) S. Balay, et al.. PETSc users manual. Technical Report ANL-95/11, Argonne National Laboratory, August 2004.
9) R. S. Tuminaro, et al.. Official Aztec user's guide, version 2.1. Technical Report SAND99-8801J, Sandia National Laboratories, November 1999.
10) 長谷川秀彦, 須田礼仁, 額田彰, 梶山民人, 中島研吾, 高橋大介, 小武守恒, 藤井昭宏, 西田晃. 計算環境に依存しない行列計算ライブラリインタフェース SILC. 情報処理学会研究報告, 2004-HPC-100, pp. 37-42, December 2004.