OpenMP を用いた帯行列に対する直接解法の並列化

長谷川 秀彦†

帯行列に対する直接解法は数値シミュレーションの核となる処理であるが，アルゴリズムの性格上，並列化効果が現れにくい．本報告では帯行列に対する直接解法がいろいろな共有メモリ方式の並列計算環境でどのような振舞いを示すかを明らかにするため，Pentium III からなる SMP，スーパコンピュータの 1 ノード，cc-NUMA による並列コンピュータを用いて測定・評価を行った．その結果，OpenMP を用いた並列化は少ない手間で良好な性能が得られること，問題が細分化されることでメモリへの負荷が軽減されて性能劣化を遅らせられる効果があることが分かった．適切な解法アルゴリズムの選択，問題サイズと計算環境に応じたスレッド数の選択も重要である．また，アンローリングなどの従来からの高速化手法も有効で，OpenMP による並列化と共存できることも明らかになった．最終的に Itanium2（1.3 GHz）8 CPU で，帯半幅 500，次元数 250,000 の方程式の LU 分解が 40 秒，求解が 4 秒でできる．対称正定値問題なら，分解が 10 秒，求解が 1 秒である．

Parallelization for Band Gaussian Elimination with OpenMP

Hidehiko Hasegawa†

We applied some tuning and parallelization methods for some Gaussian Elimination solvers for the band matrices with OpenMP, and measured their performance on some latest parallel computing environments. The storing method and band algorithm let us difficult to tune and parallelize them. From the measurement the parallelization using OpenMP was effective on some shared-memory parallel computing environments. Especially on the Symmetric MultiProcessor, it reduces the load to the memory system, then may show a high parallel performance ratio larger than the number of CPUs. Finally we conclude that the parallelization using OpenMP is very cost-effective for the Gaussian Elimination solvers to the band matrices.

1. はじめに

数値シミュレーションで最も時間がかかるのは，差分法や有限要素法で離散化して得られた連立 1 次方程式を繰り返し解く処理である．このとき，係数行列は場の幾何学的・物理的特徴を反映し，疎な行列になる．たとえば，差分法において，1 次元の場を p 分割した場合には次元数 p の 3 重対角行列が得られる．2 次元の正方形の場の 1 辺を p 分割した場合，帯半幅 p，次元数 p^2 で 1 行あたり 5 つの要素（i 行について $i-p$，$i-1$，i，$i+1$，$i+p$ 列）だけに値を持つ疎行列が得られる[1)]．3 次元の立方体の場の 1 辺を p 分割した場合は，帯半幅 p^2，次元数 p^3 で 1 行あたり 7 つの要素（i 行について $i-p^2$，$i-p$，$i-1$，i，$i+1$，$i+p$，$i+p^2$ 列）だけに値を持つ疎行列となる．3 重対角行列を係数とする場合は，3 重対角行列用の特別な解法を用いることが多いが，その他の疎行列を係数とする連立 1 次方程式の解法には，帯行列と見なして直接解法を適用する，疎行列に対する直接解法であるスパースソルバを適用する，疎行列に対して CG 法などの反復解法を適用するなどのアプローチがある．

疎構造が対角要素を中心とする帯の範囲内におさまる場合は，ゼロ要素に対する演算とメモリは必要になるが，帯行列と見なしてガウスの消去法やコレスキー分解などの直接解法を適用するのが手軽である．反復解法は，収束が係数行列と右辺の数学的特性に強く左右されることと，右辺に対しては毎回はじめから反復させる必要があるという特徴を持つ．一方，直接解法は数学的特性に影響されず一定の演算量で解け，係数行列を分解してしまえば右辺を繰り返し解くのに必要な演算量は少ないという特徴を持つ．

大規模な帯行列をコンピュータ上に格納するには，問題に合わせた行列の表現，計算環境（使用するプログラム，ハードウェアなど）に合わせたデータの格納方式が重要である．密行列に対する LU 分解やガウス

† 筑波大学図書館情報学系
Institute of Library and Information Science, University of Tsukuba

の消去法では，コンピュータの性能の限界まで到達できたが，帯行列の場合には性能が出にくかった．多くの場合，疎行列に対するアルゴリズムでは，メモリアクセスがネックとなっている．

帯行列に対する直接解法を高速化する計算環境としては，ベクトル・コンピュータ，クラスタ，共有メモリ方式の並列コンピュータと，それらの組合せが存在する．直接解法の核となる処理の高速化にはアンローリングなどが有効だが，メモリを節約するための格納方法により，すべてのブロック化とアンローリングを機械的にテストする ATLAS[2]のようなアプローチを帯行列に適用するのは難しい．ベクトル・コンピュータにおけるベクトル化はベクトル長が比較的短い場合でも有効であった．MPI を用いた分散並列化は，アルゴリズムの局所性からロードバランスがきわめて悪く，ロードバランスを改善しようとするとデータ分散による負荷が大きくなる．

本研究では，帯行列に対する直接解法について，OpenMP を用いた並列化がいろいろな並列計算環境でどのような性能と特徴を示すかを明らかにする．

2. 環境と問題

2.1 測定環境

対象とするマシンは DELL Power Edge 6350，NEC SX-6，HITACHI SR8000，SGI Altix である．

DELL Power Edge 6350（筑波大学）は 4 つの Pentium III，Xeon 550 MHz，Cache 512 KB からなる Symmetric MultiProcessor で，2 GB の共有メモリを持ち，1 CPU のピーク性能は 550 MFLOPS である．SSE は使用しない．オペレーティングシステムは RedHat Linux 6.1，Fortran コンパイラは PGI Workstation 4.0 で最適化オプション-fast を用いた．

NEC SX-6/4B（日本原子力研究所）は 4 つの CPU を持つノードから構成される分散メモリ方式のベクトル並列コンピュータで，本研究ではノードを SMP と見なす．1 CPU あたりのピーク性能は 8 GFLOPS，ノードの共有メモリは 16 GB である．コンパイラは Fortran90/SX 265 で，最適化オプションとして-R2 -C hopt -C sopt -Wf ”-pvctl fullmsg vwork=stack” を用いた．

HITACHI SR8000（日本原子力研究所）は 8 つの CPU を持つノードから構成される分散メモリ方式の並列コンピュータで，本研究ではノードを SMP と見なす．1 CPU あたりのピーク性能は 1.5 GFLOPS，ノードの共有メモリは 16 GB で擬似ベクトル機構を有する．コンパイラは最適化 fortran90 V01-05-/A で，最適化オプションとして-W0,’opt(o(ss),loopdiag(1)), langlvl(hf(77),l(90),pause(1)), pvec(diag(1), pvfunc(2))’, -pardiag=3, -procnum=8, -save, -loglist を用いた．

SGI Altix（東京大学）は 2 個の Intel Itanium2, 1.3 GHz（Madison）と 2 GB の共有メモリを持つノードから構成される cc-NUMA の共有メモリ方式並列コンピュータである．CPU 数は 32 個，メモリ容量は 32 GB で，1 CPU あたりのピーク性能は 5.2 GFLOPS である．オペレーティングシステムは RedHat Linux，Fortran コンパイラは Intel Compiler 7.0 で最適化オプション-O3 を用いた．並列実行には dplace コマンドを利用し，管理プロセス以外のプロセスをラウンドロビン方式で割り当てた．ほぼ占有状態で使用したため，スレッドは CPU 0 番から順に連続して割り当てられている[3]．

2.2 テスト問題

テスト問題は 2 次元のほぼ正方形の場を差分法で離散化して得られる対称正定値行列である．各辺の分割を M1 とすると，帯半幅 M1，帯幅 2·M1+1，次元数 N = M1·(M1+1) の帯行列が得られる．どのプログラムも解が正しく求まることは確認してあるので，今回の実験では，解が正しいと見なせる範囲においては並列化による解の変化・誤差について考慮せず，実行に要する時間のみを評価対象にする．

3. 基本アルゴリズムと高速化

帯ガウスでは帯の範囲を A(j-i,i) = $a_{i,j}$ と変換して部分軸選択付きガウスの消去法を適用する．このとき，Fortran 言語でメモリ参照が連続的になるのは行型のアルゴリズムである．行交換によって帯幅は最大で 3·M1+1 まで拡大する可能性があるため，(3·M1+1)·N の配列に収める（M1 は帯半幅，N は次元）．アルゴリズムは

```
do k = 1, N
  do i = k+1, min(k+M1,N)
    do j = k+1, min(k+2*M1,N)
      A(j-i,i) = A(j-i,i)
                -A(k-i,i)*A(j-k,k)/A(0,k)
```

$\{\ a_{ij} = a_{ij} - a_{ik} * a_{kj}/a_{kk}\ \}$

```
    end do
  end do
end do
```

といった 3 重ループになる[4),5)]．これが基本的な帯ガウス BGLU1 である．演算回数は積和演算で数えて，係数行列に対する前進消去が $2 \cdot M1^2 \cdot N$ 回，右辺の前

進消去が M1·N 回，後退代入が 2·M1·N 回である（Flops では 2 倍になる）．

帯ガウスでは消去された部分 A(k-i,i) に消去に用いた-A(k-i,i)/A(0,k) $= -a_{i,k}/a_{k,k}$ を格納するが，係数行列と $-a_{i,k}/a_{k,k}$ の部分を分離してメモリ参照を減少させ，仮想メモリ上での実行効率をあげるのが Martin-Wilkinson 流の特殊ガウス BHLU1 である[6]．アルゴリズムは

```
do k = 1, N
  do i = k+1, min(k+M1,N)
    AL(i-k,k) = -A(0,i)/A(0,k)
   { a'ki = -aik/akk }
    do j = k+1, min(k+2*M1,N)
      A(j-k-1,i) = A(j-k,i)
                   +AL(i-k,k)*A(j-k,k)
                 { aij = aij + a'ki * akj }
    end do
  end do
end do
```

のようになる．特殊ガウスでは結果 a_{ij} を A(j-k-1,i) にシフトして格納するため，添え字式が複雑になり，コンパイラによっては自動ベクトル化できないことがある．演算回数は帯ガウスと同じだが，メモリ参照が改善される．

対角優位行列の場合は，軸選択が不要なので行交換による帯幅の拡大がない．対角優位かつ対称の場合は，対角性を保持するアルゴリズムを使えば帯の上三角（下三角）部分だけを保持して消去をすればよく，このアルゴリズムを対称帯ガウス BSLU1 という[7]．演算回数は積和演算を 1 回と数えて，係数行列に対する前進消去が $M1^2 \cdot N/2$ 回，右辺の前進消去が M1·N 回，後退代入が M1·N 回である（Flops では 2 倍になる）．上三角部分だけで消去を行うアルゴリズムは

```
do k = 1, N
  do i = k+1, min(k+M1,N)
    do j = i, min(k+M1,N)
      A(j-i,i) = A(j-i,i)
                 -A(i-k,k)*A(j-k,k)/A(0,k)
                 { aij = aij - aki * akj/akk }
    end do
  end do
end do
```

のようになる．

これらのアルゴリズムはメモリ容量と演算量を削減するが，アルゴリズムとデータ構造を複雑にするので，高速化・並列化にとっては問題となる．評価には文献 4) に掲載されているコードを用い，逐次コードのチェックでは必要に応じて改変した．

表 1 BGLU1 の書き方と性能．M1=100，N=10100
Table 1 Type of source and its performance by MFLOPS.

	DELL	SR8K	Altix
(A) Original	83	252	957
(B) Scalar	84	254	957
(C) Work Array	82	251	1330
(D) Both	82	256	1330

3.1 コンパイラ向けコード

無駄なロード・ストアが発生して性能劣化が起こったり，ベクトル化されなかったりすることを防止するため，これまでの帯ガウスでは j のループで不変な-A(k-i,i)/A(0,k) にスカラ変数を用いたり，i に関して不変な A(j-k,k) に対して 1 次元配列を用いたりしてきた．以下の 4 通りのプログラムを 1 CPU 上で最適化オプションを指定してコンパイルした実行形式の性能を**表 1** に示す．

(A) A(j-i,i) = A(j-i,i)-A(k-i,i)*A(j-k,k)/A(0,k)
(B) A(j-i,i) = A(j-i,i)+T*A(j-k,k)
(C) A(j-i,i) = A(j-i,i)-A(k-i,i)*WK(j)/A(0,k)
(D) A(j-i,i) = A(j-i,i)+T*WK(j)

DELL Power Edge 6350 と HITACHI SR8000 ではプログラムの書き方によらずほぼ一定の性能を示すが，SGI Altix 上の Intel コンパイラでは 1 次元配列を使用すると性能が 1.3 倍になる．これから，同じ意味であっても，コードの書き方によって性能が異なることが分かる．このときのいちばん高速なコードのピーク性能に対する割合は Power Edge 6350 が 15%，SR8000 が 17%，Altix が 25%である．

3.2 単一 CPU での高速化

単一 CPU における高速化の基本は，データの再利用，すなわち 1 演算あたりのロード・ストアを減少させることである．それには，いったんロードしたデータにできるだけ多くの演算を施してからストアすればよい．実現にはループアンローリング（ループ内に多くの演算を詰め込むこと）が使われる．帯ガウスは 3 重ループからなるアルゴリズムなので，3 つのループの組合せに対するアンローリングが考えられる．最内側ループ j についてのアンローリングはベクトル処理可能なループのループ長が短くなり，ベクトルコンピュータでは大幅な性能劣化となるので，これまでは行われてこなかった．いちばん外側のループ k についてのアンローリングは，ストアの削減につながると同時に右辺の前進消去も高速化されるが，変更の影響は

表 2 命令の実行回数
Table 2 Number of LOAD, STORE, Branch. m = M1.

	LOAD	STORE	Branch	演算密度
原型	$4\,mn^2$	$2\,mn^2$	$2\,mn^2$	1
2 段同時	$3\,mn^2$	mn^2	$2\,mn^2$	2
2 行同時	$3\,mn^2$	$2\,mn^2$	$2\,mn^2$	2
2 列同時	$3\,mn^2$	$2\,mn^2$	$2\,mn^2$	2
2 段 2 行	$2\,mn^2$	mn^2	mn^2	4

表 3 アンローリングと性能. M1=100, N=10100
Table 3 Loop unrolling and its performance by MFLOPS.

	DELL	SX-6	SR8K	Altix
Normal BGLU1	82	1680	256	1650
2 段同時 BGLU2	152	2600	492	1790
2 段 2 行 BGLU4	145	2990	504	3300
Normal BHLU1	87	702	310	1730
2 段同時 BHLU2	151	2690	538	2270
2 段 2 行 BHLU4	147	2440	558	1730
Normal BSLU1	82	1830	216	754
2 段同時 BSLU2	148	3360	447	1230
2 段 2 行 BSLU4	146	1550	533	1390

表 4 BGLU4 の書き方と性能. M1=100, N=10100
Table 4 Type of source and its performance by MFLOPS.

2 段 2 行	DELL	SR8K	Altix
(A) Original	110	484	553
(B) Scalar	124	486	2030
(C) Work Array	129	518	576
(D) Both	145	504	2700

プログラム全体に及ぶ．1 つ内側（中間）のループ i についてのアンローリングは簡単だが，ストアの削減効果はない[5),6)]．以下，k について 2 重のアンローリングを施した 2 段同時を BGLU2，BHLU2，BSLU2，k と i について 2 重のアンローリングを施した 2 段 2 行同時を BGLU4，BHLU4，BSLU4 と呼ぶ．ロード，ストア，分岐がどれだけ削減されるかを**表 2** に，アンローリングしたコードの実測結果を**表 3** に示す．ここではスカラと 1 次元配列を使用した (D) のコードを用いた．**表 4** に BGLU4 の書き方を変えたときの性能変化を示す．

BGLU1 に対する 2 段同時化の効果は，M1 = 100，N = 10100 のとき，Power Edge 6350 で 1.8 倍，NEC SX-6 で 1.5 倍，SR8000 で 1.9 倍，Altix で 1.1 倍である．2 段 2 行同時化の効果は，Power Edge 6350 と SX-6 で 1.7 倍，SR8000 で 1.9 倍，Altix で 2.0 倍である．2 行同時だけの効果は測定していないが，2 段同時化が効果的であること，2 段に加えて 2 行同時化を施すことでより性能が向上することが分かる．表 4 によれば，最高性能を達成するコードはそれぞれのマシンで異なるが，アンローリングによってループ内が複雑になると，スカラと 1 次元配列を用いたコードが全体的に良好な性能を示す．最高性能を達成するコードのピーク性能に対する割合は Power Edge 6350 が 26%，SX-6 が 37%，SR8000 が 34%，Altix が 63%である．このとき，BGLU4 の核となるループは

```
do k = 1, N, 2
 k1 = k+1
 do i = k1+1, min(k1+M1,N), 2
  do j = k1+1, min(k1+2*M1,N)
    A(j-i,i) = A(j-i,i)+T1*WK1(j)+T2*WK2(j)
    A(j-i-1,i+1) = A(j-i-1,i+1)
                  +U1*WK1(j)+U2*WK2(j)
  end do
 end do
end do
```

のようになっている．

表 5 過去のデータ
Table 5 Historical data by MFLOPS. M1=100, N=10100

	BGLU1	BGLU2	BGLU4	Peak
S-810/20	142	234	232	630
S-820/80	640	1060	1070	2000
S-3800/480	839	1320	1470	4000
M682H-IAP	90	110	110	-
M-880/310	54	70	72	-

3.3 過去のデータ

過去のマシンでどのような傾向だったかを**表 5** に示す[5),6),8)]．メモリ系が高性能だったベクトルコンピュータでもピーク性能の 50%程度しか出ないこと，同時期のメインフレームでは大幅に遅かったことが分かる．この場合でも，アンローリングは効果的で，特に 2 段同時化が良かったことを示している．

ここでは S-820/80 と S-3800/480 のピーク性能としてロード・ストアパイプラインの性能を示した．演算のピーク性能は S-820/80 が 3 GFLOPS，S-3800/480 が 8 GFLOPS である．

4. OpenMP による並列化

共有メモリ方式では，すべてのプロセッサが一様なメモリ空間にアクセスするため，これまでのプログラムを変更せず並列実行できる．一方複数のプロセッサに対して一様なメモリ空間を提供するため，システムとしては実装が難しくなり，並列コンピュータを構成するプロセッサ数は比較的少数になる．共有メモリ方式の並列コンピュータ上の並列化手法としては，OpenMP を用いる方法[9)]，スレッドプログラミング，

表 6 2 段 2 行同時 BGLU4 の性能
Table 6 Performance of BGLU4. M1=100. Machine, # of threads, by MFLOPS.

	DELL, 4	SR8K, 8	Altix, 8
(1) most inner	152	410	80
(2) middle (array)	299	1100	1100
(3) middle (private)	331	1210	1200
(4) PPGen	308	1650	659
(5) Auto	-	1860	1160

Recursive Algorithm[10]などがある．ここでは，Fortran または C プログラムに適用が可能な OpenMP を用いたコンパイラによる並列化についてその効果と特徴を検証する．

OpenMP では，並列化すべきところにプログラマがディレクティブ（指示行）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．並列実行可能かどうかはすべてプログラマに任されており，結果の正しさ，性能などもディレクティブの入れ方に依存する．並列度は実行時にスレッド数として環境変数 OMP_NUM_THREADS に与える．本研究では 1 CPU につき 1 スレッドとしている．

4.1 どのループを並列化するか？

帯行列に対する直接解法では，アルゴリズムが 3 重ループ内の演算の繰返しになっていることから，ループの 1 つだけを並列化し，どのループの並列化が効果的かを調べる．具体的には以下の 5 通りを比較した．

(1) 最内側ループを並列化．
(2) 中間のループを並列化，配列記法使用．
(3) 中間のループを並列化，Private 節を使用．
(4) HITACHI Parallel Program Generator 使用．
(5) コンパイラによる自動並列化．

(1)，(2)，(3) では OpenMP ディレクティブを

```
!$OMP PARALLEL DO {PRIVATE(T,U)}
```

のように記述し，目的のループの直前の行に挿入した．(4) の HITACHI Parallel Program Generator[11]は Fortran プログラムを入力とし，ソースコード中に OpenMP ディレクティブを自動挿入するツールである．Parallel Program Generator（PPGen と略記）で変換されたプログラムは，OpenMP ディレクティブ入りの Fortran ソースプログラムとして，任意の Fortran コンパイラで利用できる．(5) は OpenMP ディレクティブの入っていないコードを与え，コンパイラに自動並列化を行わせる（Auto と略記）．プログラムは 2 段 2 行同時 BLU4 を用い，SGI Altix の場合に限り 1 次元配列の使用をやめた．結果を**表 6** に示す．

最内側ループの並列化はいずれの場合においても問題で，性能が大幅に劣る．このため，最内側ループに対して OpenMP による並列化を行うのは現実的でなく，最内側ループに関してはベクトル化の余地が残る．したがって，OpenMP を用いたループの並列化とベクトル化は共存できる．ループ内のスカラを配列記法に戻して（変数を明示して）コンパイラに並列化の解釈を任せる場合 (2) と，記法はそのままで Private 節で宣言する場合 (3) では，Private 節を使用したほうが約 10%の性能向上になる．DELL Power Edge 6350 では，PPGen を用いて OpenMP ディレクティブを挿入したコードは最高性能のコードよりも 6%ほど性能が劣る．Altix では，PPGen を用いたコードの性能は最高性能のコードの約半分しか出ていない．HITACHI SR8000 では，PPGen によって OpenMP ディレクティブを挿入したコード，あるいはコンパイラによる自動並列化を施したコードが高速だった．SR8000 では，単純に OpenMP を挿入したコードに比べて PPGen によるコードの性能が 40%から 50%優れており，ディレクティブの指定方法にさらなる高速化の可能性が残されていることが分かる．ピーク性能（CPU 性能・台数）と比べると，M1 = 200，N = 20200 のとき，Power Edge 6350 で 19%，Altix で 6%，SR8000 で 31%である．

4.2 問題サイズに対する依存性

問題サイズによる性能の変化をみるため，Power Edge 6350 において問題サイズを変えて実行した結果を**表 7** に示す．カッコ内は並列化率（並列版の FLOPS 値/逐次版の FLOPS 値）である．同様に SR8000，Altix において問題サイズを変えて実行した結果を**表 8**，**表 9** に示す．/ で区切られた値は，左から (3) Private 節を用いた並列化，(4) HITACHI Parallel Program Generator による並列化，(5) コンパイラによる自動並列化である．

本来ならば，問題サイズだけでなくスレッド数も変える必要があるが，Power Edge 6350 の場合は CPU が 4 台しかないこと，SR8000 と SX-6 の場合はノード内の一部だけを使うことは想定していないので，スレッド数は固定とした．Altix の場合は，スレッド数を 1，2，4，8，16 と変えて実測したが，ページの制約から問題サイズを変えた結果だけを示す．直接的なデータは示していないが，帯半幅 M1 が p 倍になると元数は p^2，メモリサイズは p^3，演算量は p^4 になるので，スレッド数を固定したデータからでも大まかな傾向は分かる（問題を大きくすると，スレッド数を減らしたことに対応する）．

表 7 によれば Power Edge 6350 での BGLU1 は M1 = 100，N = 10100，4 threads で約 3 倍の性能

表 7 サイズと並列化方法．DELL，by MFLOPS，4 threads
Table 7 (3) Private (ratio)/(4) PPGen.

M1	100	200
BGLU1	249(3.0)/195	297(8.7)/297
BGLU2	345(2.2)/313	449(7.0)/434
BGLU4	333(2.2)/308	427(6.2)/414
BHLU1	244(2.8)/240	286(7.9)/295
BHLU2	315(2.1)/299	419(6.7)/409
BHLU4	301(2.0)/299	387(5.8)/400
BSLU1	187(2.2)/ 70	199(2.3)/ 80
BSLU2	258(1.7)/144	316(2.0)/138
BSLU4	272(1.8)/117	313(2.1)/140

表 8 サイズと並列化方法．SR8000，by GFLOPS，8 threads
Table 8 (3) Private/(4) PPGen/(5) Auto.

M1	100	200	400
BGLU1	1.2/1.4/1.5	1.7/2.1/2.2	1.6/1.8/1.9
BGLU2	1.2/1.6/1.7	2.1/2.8/3.6	2.6/2.9/3.8
BGLU4	1.2/1.6/1.8	1.9/3.5/3.1	2.6/3.5/3.8
BHLU1	1.6/1.4/1.6	2.3/2.4/2.5	2.1/2.1/2.2
BHLU2	1.6/1.3/1.6	2.9/2.7/3.6	3.3/3.1/3.9
BHLU4	1.1/1.1/1.5	1.9/1.9/3.0	2.6/2.7/3.9
BSLU1	0.7/0.4/0.4	1.2/0.8/0.8	1.1/1.2/1.3
BSLU2	1.2/0.7/0.7	2.0/1.4/1.4	1.9/2.1/2.2
BSLU4	1.4/0.9/1.0	2.4/1.7/1.8	2.1/2.7/2.8

表 9 サイズと並列化方法．Altix，by GFLOPS，8 threads
Table 9 (3) Private/(4) PPGen/(5) Auto.

M1	100	200	400	500
BGLU1	0.69/0.67/1.6	1.4/ 1.5/1.1	2.7/ 3.0	3.3
BGLU2	1.1/0.64/1.7	2.3/ 1.5/1.6	4.3/ 3.0	5.1
BGLU4	1.1/0.65/1.1	2.4/ 1.6/2.5	4.8/ 3.4	5.9
BHLU1	0.74/0.75/1.7	1.7/ 1.9/1.6	3.3/ 3.7	4.0
BHLU2	0.90/0.61/2.3	2.1/ 1.4/2.4	4.2/ 3.0	5.0
BHLU4	1.0/0.65/1.0	2.2/ 1.5/2.2	4.6/ 3.1	5.8
BSLU1	0.48/0.14/0.74	1.6/0.19/1.1	3.3/0.22	4.1
BSLU2	0.73/0.23/1.2	2.0/0.32/1.6	4.0/0.40	4.9
BSLU4	0.68/0.24/1.3	1.9/0.36/2.0	4.8/0.42	5.8

になる．問題サイズを大きくすると見かけの並列化率は CPU 台数の 4 を上回って 6 から 8 になるが，並列実行時の MFLOPS 値が変化していないことから，1 CPU での性能が大幅に悪化していることが分かる．これから，問題を大きくして並列化の効果をあげようとすると，SMP の場合には分母となる 1 CPU での性能が大幅に低下し，並列化率が意味をなさなくなることが分かる．このように SMP の場合は，並列化効果よりも，問題を細分化し 1 CPU での性能を劣化させない効果のほうが大きい．全体的にアンローリングは有効で 1 CPU における性能劣化を遅らせる効果がある．

Martin-Wilkinson 流の特殊ガウスは，帯ガウスよりも性能劣化がいくぶん遅いだけで，仮想記憶の場合ほどの効果はない[6]．対称ガウスは，必要なメモリ量が 1/3 であることとから 1 CPU における性能劣化の開始が遅いが，演算量が 1/4 であることから性能値は低くなっている．

SR8000 において，中間のループだけを Private 節を用いた Parallel DO で並列化したのに比べ，Parallel Program Generator によって OpenMP 指示行を挿入したコードやコンパイラによって自動並列化されたコードは，BGLU1 で 1.2 倍弱，BGLU2 と BGLU4 で 1.5 倍程度高速になった．BHLU1，BHLU2，BHLU4 では，どのコードも大体同じような性能を示しているが，問題が大きくなるにつれてコンパイラによって自動並列化されたコードがより高性能になる．BSLU1，BSLU2，BSLU4 では，Parallel Program Generator によって OpenMP 指示行を挿入したコードやコンパイラによって自動並列化されたコードは，M1 = 200 のとき 7 割程度，M1 = 400 のとき最大で 1.3 倍程度になる．3 重ループの中間のループだけでなく 2 重ループに対しても同様な並列化を施すと，BGLU1 ではほとんど効果がないが，BGLU2，BGLU4 では最大で 1.2 倍程度に高速化できる．2 重ループに対する OpenMP 指示行の挿入は，帯ガウスと問題サイズが大きいときの対称ガウスでは効果があったが，特殊ガウスと M1 = 100 のときの対称ガウスでは逆効果だった．

Altix においては，CPU が高速で単体性能が高いため，問題サイズを大きくしないと並列化の効果が出ない．M1 = 100, 200 では，スレッド数（CPU 台数）を減らしたほうが性能は良くなり，M1 = 100, 200, 400 あたりでスレッド数を 16 にすると 0.6 倍程度の性能になる．したがって，スレッド数 8 の場合に並列化性能としてもっともな値を得るには，M1 = 500 以上を用いる必要がある．M1 = 500 の場合，BGLU1 で 14.1 倍，BGLU2 で 7.0 倍，BGLU4 で 8.1 倍，BHLU1 で 7.6 倍，BHLU2 で 5.4 倍，BHLU4 で 6.6 倍，BSLU1 で 3.8 倍，BSLU2 で 3.3 倍，BSLU4 で 2.6 倍になる．このとき帯ガウスでは 1 CPU で性能劣化が生じている．特殊ガウスは，帯ガウスと同じような性能値で並列化率が小さいことから，1 CPU での性能劣化を遅らせていることが分かる．対称ガウスはメモリ負荷が少ないことより（M1 が同一の問題で必要なメモリ量は 1/4），スレッド数 8，問題サイズ M1 = 500 で，3 から 4 倍という適切と思われる値になっている．

Parallel Program Generator が生成した OpenMP コードについては，Altix で使用する限りは芳しくなく，核となる部分 1 カ所に Private 節をともなう Parallel DO を入れたコードのほうが高性能である．帯ガウス BGLU1 の性能差は 10%程度だが，BGLU2,

表 10 2段2行同時 BGLU4 の並列化性能
Table 10 Parallel performance of BGLU4. by FLOPS, ∗: PPGen, +: Auto.

Threads	1	4	8	16
DELL	145 M	333 M	-	-
M1=200		(2.2)		
SX-6	2.4 G	-	171 M	-
M1=200			(0.06)	
SR8K	607 M	-	1.9 G:3.5 G*	-
M1=200			(3.1:5.8*)	
SR8 K		-	2.6 G:3.8 G+	-
M1=400				
Altix	2.9 G	4.1 G	2.4 G:2.5 G+	1.3 G
M1=200		(1.3)	(0.8)	
Altix	3.0 G	6.0 G	4.8 G:4.9 G+	2.7 G
M1=400		(1.9)	(1.5)	

BGLU4 については M1 = 400 で 1.6 倍の開きがある．対称ガウスでは場合によって約 10 倍の性能差があり，この結果は SR8000 上の振舞いとは大きく異なり，OpenMP による並列化は計算環境によって適切な方法が異なること意味する．Altix 上の Intel コンパイラによって自動並列化したコードは全体的に良好な性能を示すが，メモリ対する負荷によって性能が劣化しているところでは，他の並列化コードに比べると大幅な性能劣化になる（M1 = 400 の BGLU1 が 0.48 GFLOPS）．

表 10 に帯ガウス BGLU4 の性能まとめを示す．

NEC SX-6 では，ベクトル性能が良いことと，許されたメモリサイズで扱える問題が小さいため，ここでの問題ではどの解法を用いても並列化の効果はなかった．SR8000 で M1 = 200 のとき，8 スレッドでの並列化性能は 3 倍から 6 倍と妥当な値で，特に Parallel Program Generator を使用した場合は 5.8 倍とノード内並列化としては良い性能を示している．なお，M1 = 400 のときの性能値は妥当な値を示しているが，利用環境の制約により 1 CPU での値が採取できなかった．Altix の場合，性能値 6 GFLOPS 程度は達成できるが，多くの場合，そのときの 1 CPU での性能が極端に低下しており，性能値と並列化率の両方で良い値を達成するのは難しい．

5. ま と め

各種の共有メモリ方式の並列計算環境で OpenMP を利用して帯行列に対する直接解法である帯ガウス，Martin-Wilkinson 流の特殊ガウス，対称帯ガウスの高速化・並列化を行った．共有メモリ方式では，メモリ共有によってプログラムの作りやすさを提供しているが，性能はスケーラブルになりにくく，どこまで性能が得られるかが見極めにくい．

良い性能を得るには，多くの場合，スレッド数（CPU 台数）を少なめにするか，問題サイズを大きめにして，1 つ 1 つの並列タスクに十分な仕事を与える必要がある．ところが，メモリに対する負荷がある量を超えると，極端な性能低下が起こる．メモリに過度の負荷をかけうるような大規模な問題に対して，並列化は無条件で効果がある．しかし，その際の性能値，並列化率は好ましい値にはならず，性能は低め，並列化率は CPU 台数を超えた値になる．このとき，並列化率の意味をなしていないが，問題サイズとスレッド数の組合せによって，このような状況は容易に起こりうる．メモリ負荷を軽減する手法として，ループアンローリングは依然として有効であり，帯行列に対するアルゴリズムではアルゴリズムとデータ構造の複雑さからプログラマによるアンローリングが必須だった．アンローリングを施すコンパイラオプションは指定してあったが，BGLU1，BHLU1，BSLU1 をみれば分かるように，人手によるアンローリングを施していないプログラムは性能が出ない．

帯行列に対する直接解法のような 3 重ループの内部が繰り返し実行される数値計算プログラムでは，ネストの深い部分に注目して，たった 1 つの OpenMP ディレクティブを入れるだけで高性能な並列プログラムが得られる．並列化コンパイラによる自動並列化や，Parallel Program Generator を用いた OpenMP ディレクティブの挿入と比較すると，まだ高速化の余地はある．しかし，特殊ガウスや対称ガウスのような複雑なプログラムになると，Parallel Program Generator で生成したコードでは，性能が大幅に劣るケース，別の環境では性能の振舞いが大きく異なるケースが発生した．問題がディレクティブの入れ方にあるのか，コンパイラの解釈能力にあるかについて詳細な解析がいる．

3 重ループだけでなく，2 重ループに対して OpenMP ディレクティブを入れても，大幅な性能向上は得られなかった．Altix のオペレーティングシステムではデータ配置に「ファーストタッチ」が使われており[12]，データは最初にタッチ（アクセス）されたプロセスのローカルメモリに配置される．そのため，配列の初期化のループが並列化されているかどうかで性能が異なる可能性がある．配列の初期化を行うループを並列化した場合と並列化しない場合を比較してみたが，連立 1 次方程式の分解と求解部分に性能差は現れなかった．OpenMP のディレクティブの挿入方法についても

```
!$OMP PARALLEL PRIVATE(T)
do k = 1, N
!$OMP DO
  do i = k+1, min(k+M1,N)
    do j = k+1, min(k+2*M1,N)
      A(j-i,i) = A(j-i,i)-T*W(J)
    end do
  end do
end do
!$OMP END PARALLEL
```

のように，PARALLEL によるスレッドの生成を事前に行う方法についても測定したが，ここでのプログラムの場合は性能が向上せず，単純な方法が最も高性能だった．さらなる性能向上のためには，プログラム全体の見直しが必要だろう．その場合も，性能向上が得られる可能性もあるが，問題サイズとスレッド数の関係によっては性能劣化にもなりうるので注意が必要である．

Power Edge 6350 のような少数の CPU からなる SMP，SR8000 のようなスーパコンピュータの 1 ノード，多数の CPU から構成される共有メモリ方式の並列コンピュータである Altix で，たった 1 行の OpenMP ディレクティブを入れるだけで非常に良い性能が得られた．安価なマシンでは，問題が大きくなると逐次版が極端に遅くなるため，見かけの加速率は台数を超える．このような場合，並列化は問題を小さく分割することで，Cache とメモリの影響で性能が劣化するのを遅らせる効果が大きい．高価なマシンでの性能は問題サイズとスレッド数に強く依存する．ベクトルコンピュータ SX-6 では，ベクトル性能が良いこと，小さい問題しかテストできなかったことから，テストした範囲では並列化が有効という結果は得られなかった．

手間を考えると OpenMP による並列化は非常に効率的である．しかし，アルゴリズム，問題サイズ，計算環境，スレッド数などによって最適な方法が異なり，現状では汎用的なコードとして 1 つの方法を定めることはできなかったが，ベクトル化やアンローリングなどのこれまでの高速化手法と矛盾なく導入できる．些細な並列化であっても，プログラムの局所参照性を高めるために有効で，性能劣化の抑止効果が大きい．プログラムの構造を大きくを変えずに並列化ができるという点で，帯行列に対する直接解法の OpenMP による並列化は無条件で有効である．これは，MPI などを用いた分散並列環境での並列化と大きく異なる点で，少しずつ並列化ができる．

実は，Power Edge 6350 の 1 CPU 上では，LAPACK [13] の DGBTRF に ATLAS [2] でチューニングされた BLAS を用いると M1=100，N=10100 の問題が 354 MFLOPS で解ける．これは，帯幅が広がったことで帯の内部を密行列と見なして計算することが現実的になりつつあることを示すと同時に，逐次版のカーネルの性能が非常に大事であることを意味する．しかし，現状の ATLAS では BLAS（実際は行列積ルーチン）の共有メモリ向け並列化は行われていないため，OpenMP を用いた並列化に組み込むことはやさしくない．しかも共有メモリでの並列化は OpenMP だけが選択肢ではない．仮に SMP 版の BLAS が得られたとしても，アルゴリズムによってどのレベルで並列化すべきかは異なってくるので [14]，BLAS の並列化だけで最適な並列プログラムができるわけではない．一方，ソフトウェア・ハードウェアの高性能化によって，ほとんど何もせず高速化がはかられるため，採用した高速化手法が将来の可能性を狭めないよう注意しなければならない．

結論をいうのは難しいが，現実的に「共有メモリ環境なら OpenMP による並列化が有効」は事実であり，それに必要な労力・コストはわずかである．しかし，OpenMP だけで最高性能を達成するのは難しい．利用者の立場で，共有メモリ方式の並列計算環境で「どのアルゴリズム・並列化手法が有効かを明らかにする」には，他の解法，たとえば反復解法についても並列化したうえでの比較・検討がいるし，対象とする物理問題によっても問題（帯半幅と次元数の関係，サイズ）は変わってくるので，それらについても考える必要がある．

最後に，Itanium2（1.3 GHz）8 CPU で 3 GB のメモリが利用可能なら，OpenMP を使った単純な並列化によって，正統的な部分軸選択付きの帯ガウス BGLU4 を用いて帯半幅 M1 = 500，次元数 250,000 の問題が 41 sec（5.9 GFLOPS）で解ける．右辺の計算 BGSLV4 を 10 回繰り返しても 40 sec（190 MFLOPS）である．右辺の計算を高速化したいなら，Martin-Wilkinson 流の特殊ガウス BHLU4 を用いれば分解が 43 sec（5.8 GFLOPS），右辺の計算 10 回が BHSLV4 で 16 sec（461 MFLOPS）である．問題が対称正定値なら，対称帯ガウス BSLU4 を用いて分解が 10 sec（5.8 GFLOPS），右辺の計算 10 回が BSSLV4 で 10 sec（500 MFLOPS）である．変更点は BGLU4，BHLU4，BSLU4 に

```
!$OMP PARALELL DO PRIVATE(T,T1,U,U1)
```

を加えるだけである．

謝辞 本研究は，2002 年度図書館情報大学特別研究として研究費の支援を受けた．また Altix の使用にあたり，科学技術振興事業団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」プロジェクトの支援を得た．SR8000，SX-6 は日本原子力研究所に使わせていただきました．HPCS2004/ACS 6 の査読者と筑波大学電子・情報工学系高橋大介氏からは有益なコメントをいただきました．あわせて感謝いたします．

参 考 文 献

1) 村田健郎，小国　力，三好俊郎，小柳義夫（編著）：工学における数値シミュレーション，丸善 (1988).
2) http://www.netlib.org/atlas/
3) Linux アプリケーションチューニングガイド. (007-4639-001) http://www.sgi.co.jp/servers/altix/ADP/programming/overview/pdfs/Linux_Application_Tuning_Guide_ADP_v1.pdf
4) 小国　力（編著）：行列計算ソフトウェア，丸善 (1991).
5) 長谷川秀彦：帯行列に対する直接解法の高速化，情報処理学会論文誌，Vol.30, No.4, pp.402–409 (1989).
6) 長谷川秀彦：帯行列を係数とする連立一次方程式の解法（II），図書館情報大学研究報告，Vol.6, No.2, pp.89–112 (1987).
7) 長谷川秀彦：帯行列を係数とする連立一次方程式の解法（III），図書館情報大学研究報告，Vol.7, No.1, pp.61–74 (1988).
8) 長谷川秀彦：大規模行列計算ソフトウェアについて，東京大学大型計算機センターニュース，Vol.24, No.6, pp.121–148 (1992).
9) Chandra, R., et al.: *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, San Francisco, California (2001).
10) Gustavson, F.G.: Recursive Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms, *IBM Journal of Research and Development*, Vol.41, No.6 (Nov. 1997).
11) http://www.hitachi.co.jp/Prod/comp/soft1/HPC/ppgen/ppgen.html
12) Altix3000 プログラミングガイド．http://www.sgi.co.jp/servers/altix/ADP/programming/overview/pdfs/altix3k_programming_guide.pdf
13) Anderson, E., et al.: *LAPACK Users' Guide, 3rd Ed.*, SIAM, Philadelphia, Pennsylvania (1999).
14) 舘野諭司ほか：共有メモリ型並列計算機上の行列計算に対する並列化手法の性能評価，情報処理学会論文誌：コンピューティングシステム，Vol.44, No.SIG 11(ACS 3), pp.286–295 (2003).

(平成 15 年 10 月 10 日受付)
(平成 16 年 1 月 18 日採録)

長谷川秀彦（正会員）

1958 年生．1983 年筑波大学大学院博士課程社会工学研究科中退．同年図書館情報大学助手．1996 年より図書館情報大学助教授．2002 年 10 月大学統合により筑波大学助教授，図書館情報学系．博士（工学）．数値線形代数全般に興味を持つ．日本応用数理学会，SIAM，ACM 各会員．