帯行列に対する直接解法の並列化

長谷川 秀彦[†], 伊藤 祥司[††]

[†] 筑波大学 図書館情報学系　[††] 筑波大学 電子・情報工学系 & 学術情報処理センター

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。本報告では帯ガウスがいろいろな並列計算環境（共有・分散）でどのようなふるまいを示すかを明らかにするため、Pentium III からなる SMP, スーパーコンピュータの１ノード、 cc-NUMA による並列コンピュータ、ベクトル並列コンピュータを用いて測定・評価を行った。その結果、並列化効果は著しいとはいえないが、 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、特に SMP では問題が細分化されることでメモリへの負荷が軽減され台数以上の加速率が得られることがわかった。また、アンローリングなど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。

Parallelization for Band Gaussian Elimination

Hidehiko Hasegawa[†] and Shoji Itoh[††]

[†] Inst. of Library & Info. Sci., University of Tsukuba

[††] Inst. of Info. Sciences and Electronics, and Sci. Info. Processing Center, University of Tsukuba

We applied some tuning and parallelization methods for the Band Gaussian Elimination solvers, and measured their performance on some latest parallel computing environments. The storing method and algorithm let us difficult to tune and parallelize them. At the measurement they did not show a good parallel performance, however the parallelizaton using OpenMP was very cost-effective, especially on the Symmetric MultiProcessor. It reduces the load to the memory system, then shows a high parallel performance ratio larger than number of CPUs. Finally we conclude that some tuning methods such as unrolling are applicable with the paralleization using OpenMP for the Band Gaussian Elimination solvers.

1　はじめに

数値シミュレーションでもっと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、２次元の正方形の場の１辺を p で等分割した場合、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次元の立方体の場では、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) だけに要素を持つ疎行列となる。疎行列を係数とする方程式の解法には、帯行列とみなして直接法を適用する、疎行列に対する直接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、帯行列とみ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、与えられた右辺に対しては毎回１から反復させる必要があるという特徴を持つ。いっぽう、直接解法は数学的特性に影響されず一定の時間で解けること、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 分解やガウスの消去法では、コンピュータの性能を極限まで達成することが可能だったが、帯行列の場合には性能がでにくかった。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、分散並列コンピュータ、共有並列コンピュータと、そ

れらの組み合わせが存在する。直接解法の核となる演算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、密行列に対して機械的にすべてのブロック化とアンローリングをテストする ATLAS[2] のようなアプローチを帯行列に対して適用するのは難しい。これまでベクトル・コンピュータにおけるベクトル化はベクトル長が短いものの有効であった。共有並列コンピュータにおける OpenMP を用いた並列化の実行はたやすいが、性能は未知数である。分散並列コンピュータにおける 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極めて悪く、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。

1.1 測定環境

本研究では、帯行列に対するガウスの消去法アルゴリズムについて、いろいろな計算環境でどのような並列化性能と特徴が得られるかを明らかにする。対象とするマシンは DELL Power Edge 6350, SGI Altix, HITACHI SR8000, Fujitsu VPP5000 と IBM RS6000 である。DELL Power Edge 6350 は 4 つの Pentium III, xeon 550MHz, Cache 512KB からなる Symmetric MultiProcessor で、2GB の共有メモリを持つ。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 である。Power Edge 6350 のピーク性能は 1 CPU で 550MFLOPS, システムでは 2.2GFLOPS である。SGI Altix は 32 個の Intel Itanium2, 1.3GHz (Madison) からなる cc-NUMA の共有並列コンピュータで 32GB の共有メモリを持つ。オペレーティングシステムは RedHat Linux, コンパイラは Intel Compiler 7.0 で、ピーク性能は 1 CPU あたり 5.2 GFLOPS である。HITACHI SR8000 は 8 つの CPU を持つノードから構成される分散並列コンピュータで、本研究ではノードを SMP とみなして使う。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。Fortran コンパイラは最適化 fortran90 V01-05-/A である。Fujitsu VPP5000 はベクトルプロセッサをノードに持つ分散並列コンピュータで、8.6GFLOPS, 16GB の PE 80 台から構成されている。オペレーティングシステムは UXP/V, Fortran コンパイラは富士通製である。IBM RS6000 は Power3-II, 400 MHz からなるワークステーションで逐次コードの比較に使う。コンパイラは IBM 製で、ピーク性能は 1.6 GFLOPS である。

1.2 実験問題

テストに用いた問題は 2 次元の長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、帯半幅を M1 とすると、帯幅が 2 ・ M1+1, 次元数は N = M1 ・ (M1+1) と N = M1 ・(2・M1+3) (分散並列化のテストで使用) である。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みなせる範囲において、並列化による解の変化・誤差については考慮せず、実行に要する時間のみを評価対象にする。

2 基本アルゴリズムと高速化

帯ガウスでは帯の範囲内を A(j-i,i) = $a_{i,j}$ と変換して (3・M1+1)・N の配列に収め、部分軸選択付きガウスの消去法を適用する (M1 は帯半幅、 N は次元)。部分軸選択における行の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。アルゴリズムを Fortran で書くと

```
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)
    end do
  end do
end do
```

といった 3 重ループになる [3] [4] 。これが基本的な帯ガウス BGLU1 である。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 2・$M1^2$・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。

対角優位の場合は、軸選択が不要なので行の交換による帯幅の拡大はない。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ならば帯の上三角（下三角）を格納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。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 $M1^2$・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。帯ガウスでは消去された部分に、消去に用いた $-a_{i,k}/a_{k,k}$ = -A(k-i,i)/A(0,k) を格納するが、この部分を係数行列と分離してメモリ参照を減少させ、仮想メモリ上での実効効率をあげるのが特殊ガウス BHLU1 である [5] 。特殊ガウスでは結果を A(j-i-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ベクトル化されないことがあった。これら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。

2.1 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、ベクトル化されないといったことを防止するため、j のループで不変な -A(k-i,i)/A(0,k) にスカラー変数を用いたり、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コンパイラの最適化を補助してきた。念のため、各マシンの 1 CPU 上で最適化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。

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。SGI Altix 上の Intel コンパイラでは、スカラーを使用すると約２倍、１次元配列を使用すると３分の１と性能が大きく変化する。IBM RS6000 では、スカラーを使用すると 22 % ほど性能向上がみられる。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わかる。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 34 %, RS6000 が 22 % である。

2.2 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、ループアンローリングには３つのループの組み合わせが考えられる。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、これまでは禁止的とされてきた。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、変更の影響がプログラム全体に及ぶ。一つ内側のループ i についてのアンローリングは比較的簡単だが、ストアの削減効果はない [4) 5)]。以下、k についてアンローリングを施した２段同時を BGLU2, k についてのアンローリングと i についてのアンローリングを施した２段２行同時を BGLU4 と書く。ロード、ストア、分岐がどれだけ削減されるかを表 2、アンローリングしたコードの実測結果を表 3 に示す。ここではスカラーと１次元配列を使用している。表 4 に BGLU4 の書き方を変えたときの性能の変化を示す。

２段同時化の効果は, M1 = 100, N = 10100 のとき、 Power Edge 6350 で 1.8 倍、 SR8000 で 1.9 倍、Altix で 1.1 倍、RS6000 で 1.5 倍である。２段２行同時化の効果は, Power Edge 6350 で 1.7 倍、 SR8000 で 1.9 倍、Altix で 2.0 倍、RS6000 で 1.5 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わかる。表 4 から、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、BGLU4 のようにループ内が複雑になると、スカラーと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ことがわかる。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 %, SR8000 が 34 %, Altix が 52 %, RS6000 が 34 % である。このとき、 Fortran コードは

```
do k = 1, N, 2
  k1 = k+1
  do i = k1+1, min(k1+M1,N), 2
    do j = k+1, min(k1+2*M1,N)
      A(j-i,i) = A(j-i,i)+T1*W1(j)+T2*W2(J)
      A(j-i-1,i+1) = A(j-i-1,i+1)
                  +U1*W1(j)+U2*W2(J)
    end do
  end do
end do
```

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

	DELL	SR8K	Altix	RS6K
Original	83	252	957	284
Scalar	84	254	1810	348
Work Array	82	251	316	280
Both	82	256	1330	354

表 2: 命令の削減量
Number of LOAD, STORE, and Branch, m = M1

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

のようになり、ソースプログラムの行数は約２倍強になる。

2.3 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[4) 5) 6)]。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 % 程度しかでないこと、同時期のメインフレームでは大幅に遅かったことがわかる。この場合でも、ループアンローリングは効果的で、特に２段同時化がよかったことを示している。

3 OpenMP による並列化

共有メモリモデルでは、すべてのプロセッサが一様なメモリ空間をアクセスするため、これまでのプログラムモデルを変更せずに並列実行が可能になる。いっぽう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必要があるため、システムとしては実装が難しくなり、並列計算機を構成するプロセッサ数は比較的少数になる。共有メモリ型の並列計算機上の並列化手法としては、 OpenMP を用いる方法[7)]、スレッドプログラミング、 Recursive Algorithm[8)] 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を用いる。

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。並列度は実行時に環境変数 OMP_NUM_THREADS で使用するスレッド数を与える。ここでは 1 CPU につき 1 スレッドとした。一般のコンピュータ上では、ディレクティブはコメントとして扱わるため、様子をみながら徐々に並列化することが可能になる。

帯ガウスにおいては、アルゴリズムが３重ループ内の演算の繰り返しになっていることから、これらのループの１つだけを並列化し、どのような並列化が効果的を調べる。具体的には以下の４通りを比較した。

(1) 最内側ループを並列化、
(2) ひとつ内側のループを並列化、配列記法使用
(3) ひとつ内側のループを並列化、 Private 節を使用
(4) HITACHI Parallel Program Generator 使用

(1), (2), (3) では OpenMP ディレクティブを

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

のように記述した。(4) の HITACHI Parallel Program Generator[9)] は Fortran プログラムを入力とし、並列化コンパイラで使用されている並列化手法を用いてソースコード中に OpenMP ディレクティブを挿入するツールである。Parallel Program Generator (表中では PPGen) 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。プログラムは２段２行同時 BLU4 を用い、 Altix の場合に限り１次元配列の使用をやめた。結果を表 6,7,8 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があり、OpenMP を用いたループの並列化とベクトル化は共存できることがわかる。ループ内のスカラーを配列記法に戻した場合より、記法はそのままで Private 節で宣言するほうが、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。SGI Altix では、 PPGen を用いたコードは最高性能のコードの約半分の性能しかでていない。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最も高速だった。問題サイズが大きくなるにつれて、PPGen によるコードが優位になる傾向がある。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

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

	DELL	SR8K	Altix	RS6K
Original	82	256	1330	354
２段同時	152	492	1570	531
２段２行	145	504	2700	553

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

２段２行	DELL	SR8K	Altix	RS6K
Original	110	484	553	388
Scalar	124	486	2030	486
Work Array	129	518	576	420
Both	145	504	2730	553

の指定方法でさらなる高速化の可能性が残されていることがわかる。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。

表 9,10 のカッコ内に加速率（並列版の FLOPS 値 / 逐次版の FLOPS 値）を示す。表 9 から Power Edge 6350 では M1 = 100, N = 10100, 4 threads で約 2 倍の性能になる。問題サイズを大きくすると加速率は台数を上回り、並列実行時の MFLOPS 値は大きく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わかる。このような SMP の場合は、並列化効果よりも問題を細分化することによる効果のほうが大きいことがわかる。

表 10 から、 Altix では 4 台で約 1.7 倍、 8 台で 1.2 倍、16 台で 0.6 倍となり、並列化の効果があるとは言い難い。他の機種に比べると単一 CPU の性能がよすぎるため、結果的に同一サイズの問題で測定したことが不適切だったかもしれない。SR8000 では PPGen を用いた場合と、一つだけ OpenMP ディレクティブを入れた場合で大幅な性能差が生じた。PPGen を使用した場合は 8 台で 7.1 倍、素朴な並列化で 4.7 倍と単一ノード内での並列化では極めてよい性能を示している。

3.1 OpenMP による並列化のまとめ

Power Edge 6350 のような少数の CPU からなる SMP, SR8000 のようなスーパーコンピュータの 1 ノードでは、たった 1 行の OpenMP ディレクティブを入れるだけで非常によい性能が得られた。特に安価なマシンでは、問題が大きくなると逐次版が極端に遅くなるため、加速率の値は台数を越える。このような場合の並列化は小問題に分割することで、 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。一方、多数の CPU から構成される共有メモリコンピュータである Altix では、単体の CPU 性能がいいこともあり、 OpenMP による並列化ではよい性能が得られていない。

帯ガウスのような 3 重ループで構成される数値計算プログラムでは、ループの深い部分に注目して OpenMP ディレクティブを一つ入れれば簡単に高性能が得られる。しかし、専用の並列化コンパイラによる並列化や、 Parallel Program Generator を用いた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。特にメモリ配置、ループ終了時の処理の指定を詳細に検討する必要があるだろう。しかし、 Parallel Program Generator を使って挿入した OpenMP ディレクティブがすべてのマシンでよい結果だったわけではない。計算環境に依存した問題なのか、コンパイラの側に問題があるのかについて細かな解析が必要である。

4 MPI による並列化

分散メモリ型並列コンピュータ向けの帯ガウスと対称帯ガウスが日本原子力研究所の PARCEL[10] にある。PARCEL には Fortran と C, MPI と pvm の組み合わせで 4 通りがあるが、ここでは Fortran で MPI 版の対称帯ガウスを使い、ベクトル並列コンピュータ Fujitsu VPP5000 で測定を行った。対称帯ガウスは演算量が帯ガウスの 1/4 で、最内側ループのループ長も短く、しかも可変であるため、ベクトル化・並列化にとって性能がでにくく、より難しい問題となっている。なお、一般の帯ガウスについては現在評価中である。

分散メモリ版の対称帯ガウスのアルゴリズムでは、係数行列の 2 行からなるブロックをサイクリックに分散する。対称帯ガウス 2 段同時 BSLU2 の並列版プログラムの概略を以下に示す。ここで MU は帯半幅、MY はプロセッサのランク、NPROCS は使用するプロセッサ数を表し、LLK, LLK1 はグローバル・インデクスから計算されたローカル・インデクスである。最内側の J ループが各プロセッサで並列に実行される。

```
do K = 1, N, 2
  K1=K+1
```

表 5: 過去のデータ
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
M682H-IAP	90	110	110	-
M-880/310	54	70	72	-

表 6: Performance on Power Edge 6350
BGLU4, 4 threads, by MFLOPS

M1	100	150	200
most inner	152	209	244
middle(array)	299	381	384
middle(private)	331	434	423
PPGen	310	417	407

```
( 奇数段 )
  参照データをブロードキャスト
  T    = -A(1,LLK)/A(0,LLK)
  do J = K+1, min(K1+MU,N)
    A(J-K1,LLK1) = A(J-K1,LLK1)+T*A(J-K,LLK)
  enddo
( 偶数段 )
  参照データをブロードキャスト
  do I = K1+1, min(K1+MU,N)
    if( MY.eq.IPROC(I) ) then
       T  = A(I-K ,LLK)*DIVA
      T1 = A(I-K1,LLK1)*DIVA1
       do J = I, min(K1+MU,N)
         A(J-I,LLIOE) = A(J-I,LLIOE)
                       +T*A(J-K,LLK)
                        +T1*A(J-K1,LLK1)
       enddo
     endif
   enddo
enddo
```

1998 年に IBM SP2 上で測定した２段２行同時 BSLU4 の結果を表 11 に、VPP5000 で測定した２段同時 BSLU2 の結果を表 12 に示す。

SP2 では並列化効果がほとんどなかったが、VPP5000 上の M1=150 の問題では 4 台で 1.6 倍になっている。VPP5000 はベクトル並列コンピュータであり、最内側ループはベクトル実行される。演算量が少なく複雑なアルゴリズムに対する並列化としてはよい値だが、現実に使うには問題があるだろう。また問題が大規模化すれば並列化の有効範囲が広がる可能性もあるが、このままでは大幅な性能向上は期待できない。

4.1 MPI による並列化のまとめ

テストした問題サイズの範囲では並列化が有効といえる結果が得られなかったが、 5 年前の計算環境に比べれば並列化性能は向上している。しかし、この程度の加速率は共有メモリ方式の並列コンピュータや SMP で OpenMP のディレクティブを一つ入れるだけで達成できるため、分散メモリ用にプログラムを書き換え、分割されたデータをファイル経由で受け渡しする手間をかけるほどのメリットはない。高速化のためにデータ分散を変えても、アルゴリズムは細長い配列のごく１部分を更新するだけなので１部の PE しか動作せず、アルゴリズムの制約から高速化は困難だろう。

より大規模な問題と、 PC クラスタのような今回のテスト環境と大きく異なる環境でのデータは必要だが、現状から予想する範囲では顕著な効果は期待薄である。しかし、共有メモリに入りきらない大規模問題が CPU 台数さえ増やせば解けるというメリットがある。

5 まとめ

各種の並列環境で帯行列に対する直接解法である帯ガウスと対称帯ガウスの高速化・並列化を行った。手間を考えると OpenMP による並列化は非常に効率的で、特に安価な SMP やスーパーコンピュータの１ノードではよい性能が得られた。しかし、問題の形状（帯幅と次元数）や大きさ、利用する環境によってよい方法が異なっており、現状では汎用的なコードとして一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ようなこれまでの高速化手法と矛盾なく導入できるので、徐々に並列化ができる。些細な並列化であっても、並列化はプログラムの局所参照性を高めるために有効である。プログラム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。いっぽう MPI による並列化はプログラムの変更が必要なうえ、まだ有効といえるような性能を得ていない。

実は、 Power Edge 6350 の単一 CPU 上では、LAPACK[11] 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 で解けてしまう。これは、帯幅が広がることによって、帯の内

表 7: Performance on Altix
BGLU4, 8 threads, by GFLOPS

M1	100	150	200
most inner	0.08	0.12	0.14
middle(array)	1.1	1.6	2.0
middle(private)	1.2	2.1	2.9
PPGen	0.6	1.0	1.5

表 8: Performance on SR8000
BGLU4, 8 threads, by GFLOPS

M1	100	150	200
most inner	0.41	0.47	0.71
middle(array)	1.10	1.72	2.33
middle(private)	1.21	1.85	2.41
PPGen	1.67	2.61	3.58
auto parallel	1.93	2.55	3.56

側を密行列とみなすことが現実的になりつつあることを示している。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）のアンローリングやブロックサイズを変えているだけで、共有メモリ向けの並列化は行っていないため、OpenMP を用いた並列化に組み込むことは易しくない。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をすべきかは異なってくるので [12] ので、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。いっぽう、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、現在の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。

結果に一般性をもたせて「現状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、別のハードウェア・ソフトウェアでの測定がいるし、問題の型とサイズについても吟味がいる。同時に、どのような場合に優位か（ vs 反復法）を言うには他の解法・実現方法との比較も必要である。

結論を言うのは難しいが、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。OpenMP ディレクティブを自動的に入れてくる HITACHI Parallel Program Generator のようなツールも有効である。しかし、最高性能を追求するのであれば道は遠い。

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。PARCEL の利用にあたっては、日本原子力研究所 計算科学技術推進センター山田進氏に協力して頂きました。あわせて感謝致します。

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) 小国力編著: 行列計算ソフトウェア, 丸善 (1991).

4) 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).

5) 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).

6) 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).

7) Chandra, R. et al.: *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, San Francisco, California(2001).

8) F. G. Gusatavson.: *Recursive Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms*, IBM Journal of Research and Development, Vol. 41, Number 6, November 1997.

9) http://www.hitachi.co.jp/Prod/comp/soft1/HPC/ppgen/ppgen.html

10) http://www.jaeri.go.jp/jpn/open/press/000228/sanko02.html

表 9: Parallel Performance on Power Edge 6350
BGLU4, 4 threads, by MFLOPS

	BGLU1	BGLU2	BGLU4
M1=100,N=10100	223	315	301
	(2.6)	(2.0)	(2.0)
M1=150,N=22650	256	357	357
	(4.5)	(3.5)	(2.9)
M1=200,N=40200	261	370	362
	(7.4)	(5.6)	(5.2)

表 10: Parallel Performance of BGLU4
M1=200, N=40200, by FLOPS,∗: PPGen

Threads	1	4	8	16
DELL	145M	423M	-	-
		(2.9)		
Altix	2.4G	4.2G	2.9G	1.5G
		(1.7)	(1.2)	(0.6)
SR8K	504M	-	2.4G:3.5G*	-
			(4.7:7.1*)	

11) Anderson, E. et al., *LAPACK Users' Guide (Third Ed.)*, SIAM, Philadelphia, Pennsylvania(1999).

12) 舘野論司他: 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. 286-296(2003).

表 11: Parallel Performance on SP2
BSLU4, by MFLOPS

PEs	1	2	4
M1=100	91	70	60
N=20300	(1.00)	(0.76)	(0.65)
M1=150	93	100	96
N=45450	(1.00)	(1.08)	(1.03)

表 12: Parallel Performance on VPP5000
BSLU2, by MFLOPS

PEs	1	2	4	8	16
M1=100	455	525	640	539	436
N=20300	(1.00)	(1.15)	(1.40)	(1.18)	(0.95)
M1=150	735	935	1210	1210	810
N=45450	(1.00)	(1.27)	(1.63)	(1.63)	(1.10)