A-003

OpenMP を用いた帯ガウスのＳＭＰ並列化

長 谷 川　　秀 彦†

コンピュータの高性能化・低価格化により、SMP が手軽に使えるようになった。安価な SMP 上で OpenMP を用いて帯ガウスの高速化を行った結果、段に関するアンローリングは依然として有効で、OpenMP の指示行を一つ入れるだけで大幅な高速化が達成された。大規模問題では、並列化によって問題が分割され、メモリ系への負荷が軽減され、台数以上の加速率が得られた。

1. はじめに

帯行列を係数とする連立１次方程式の解法に帯ガウスがある。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)。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[1)]。これまでのコードでは、コンパイラの最適化を補助するため、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次元配列を用いてきた。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、対応する右辺の前進消去が M1・N 回、後退代入が 2・M1・N 回である。

メモリ参照を減らして高速化をはかるため、いちばん外側のループ k についてのアンローリング、もう一つ内側のループ i についてのアンローリングが有効だった。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ため２段化が限度である。いっぽう、 i についてのアンローリングは比較的簡単だが効果もそこそこである。以下、基本となるコード BGLU1, k について２段のアンローリングを施した BGLU2, k について２段と i について２行のアンローリングを施した BGLU4 を対象とする。ベクトル計算機で高速性を得るにはベクトルループ長を長くする必要があったため、再内側ループのアンローリングは行っていない。

† 筑波大学図書館情報学系・図書館情報大学
Inst. of Library & Info. Sci., University of Tsukuba

2. OpenMP による並列化

共有メモリ型の並列計算機上での並列化手法の一つとして OpenMP を用いる方法がある[2)]。OpenMP では !$OMP PARALLEL DO のような指示行をソースコードに挿入し、コンパイラによる並列化を行う。プログラムは指定された部分だけが並列実行され、その他の部分は逐次コードと同じ動作をする。実行時に使用するスレッド数は環境変数で与える。本研究では 4CPU 構成の SMP である DELL PowerEdge 6350 を用いる。CPU は Pentium III Xeon 550MHz, Cache 512KB, 共有メモリ 2GB である。RedHat Linux 6.1 上で PGI Fortran コンパイラ 4.0 を用いた。測定は他にユーザがいないときに行った。

2.1 どのループを並列化すべきか

まずは、どのループの並列化が有効かを確認する必要がある。帯ガウスでは、３重ループの深い部分が何度も繰り返されることに着目して、再内側 j とひとつ外側（中間）のループ i の DO 文の直前に

```
!$OMP PARALLEL DO
```

だけを入れて測定を行った。一番外側のループ k についての並列化はアルゴリズム全体を並列化することになり、非現実的なので対象外とした。測定は帯半幅 M1 = 100, 次元 N = 10100, スレッド数 2 とした。

表１の結果から、特に２段のアンローリングが依然として有効であること、最内側ループ j の並列化は性能が落ちることがわかる。

2.2 問題サイズ

OpenMP による並列化は中間のループ i だけと決め、問題サイズとスレッド数を変化させてその効果を測定した。カッコ内は加速率（並列版の MFLOPS 値 / 逐次版の MFLOPS 値）である。

4 スレッドでの結果（表２）から、スレッド数を増やすことで十分な高速化が達成されることがわかる。しかし、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。ここでの並列化は、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。1 CPU に複数のスレッドを割り当てても速度向上はなかった。

- ２重ループに対する指示行の挿入
- DO NOWAIT の使用
- ループ内のスカラー変数を配列に戻す
- PRIVATE(T) の使用

を変えても性能に大きな変化はなかった。

3. 逐次コードの高速化

LAPACK[3] の DGBTRF に ATLAS[4] 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 SMP でのさらなる高速化を考えると、 BLAS（実際は行列積ルーチン）が共有メモリ向けの並列コードになっている必要があるが、現状の ATLAS ではアンローリングやブロックサイズを変えているだけで、 SMP 向け並列化は行っていない。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[5] ので、BLAS の並列化だけでは最適にならない。

われわれがベクトル計算機向けにチューニングした帯ガウス BGLU1, BGLU2, BGLU4 は、現在の Cache ベースのコンピュータでは ATLAS を活用した LAPACK と大きな性能差ができてしまった。しかも DGBTRF とはデータ格納方法が異なるため、簡単に BLAS 対応のコードに直すことは難しい。ところが、 OpenMP を用いればたった一つの指示行を入れてコンパイルするだけで高性能が得られる。

表 1 並列化ループと性能 MFLOPS, 2 threads
Table 1 Paralleized loop and its performance by MFLOPS, M1=100, N=10100

	BGLU1	BGLU2	BGLU4
逐次	82	152	145
最内側 j	56	101	139
中間 i	150	210	206

表 2 問題サイズと性能 MFLOPS, 4 threads
Table 2 Problem size and its performance 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)

4. ま と め

帯ガウスに対しては、 OpenMP 指示行を一つ追加するだけで並列化が達成できた。並列化はプログラムの局所参照性を高めるためにも有効であり、これまでの高速化手法と矛盾なく導入することができた。本研究で実測したのは１種類のマシン、 4 CPU までなので、結果に一般性をもたせるには別のハードウェア・ソフトウェアでの測定がいる。また、 ATLAS を用いれば単一 CPU 上では性能の上限値が得られるので、それを共有メモリ上の並列ソフトウェアに活かすための方法を考えなければいけない。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、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。

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。

参 考 文 献

1) 小国力編: 行列計算ソフトウェア, 丸善 (1991).
2) Chandra, R. et al.: *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, San Francisco, California(2001).
3) Anderson, E. et al., *LAPACK Users' Guide (Third Ed.)*, SIAM, Philadelphia, Pennsylvania(1999).
4) http://www.netlib.org/atlas/
5) 舘野諭司他: 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. 3 掲載予定 (2003).