Vol. 30 No. 4 情報処理学会論文誌 Apr. 1989

帯行列に対する直接解法の高速化†

長谷川 秀彦††

本論文では帯行列を係数とする連立一次方程式の直接解法を検討した．方程式を少数回解く場合には帯ガウス，同一の係数行列に対して右辺の計算を繰り返す場合には Martin と Wilkinson の帯ガウス，係数行列が対称正定値ならば対称帯ガウスが良く，大規模問題用のプログラムを作成するうえではメモリ参照に注意を払う必要があった．ベクトル計算機で高速化をするためには並列に実行される演算パイプに対して必要なロード・ストアパイプが少ない2段2行同時のアンローリングが良い．ベクトル計算機や IAP では高速化のために制限三項演算の適用が必須だが，コンパイラにはそれが識別できないこともあるので注意が必要である．またメモリとレジスタ間の無駄なデータのやりとりを減らすことも重要な点である．高速化手法の適用をコンパイラだけに期待するのは難しく，アルゴリズムから検討しなおす必要があった．陽アンローリングはプログラムを複雑にするが，結果として得られたプログラムはベクトル計算機と汎用計算機の両方で高速に実行が可能となる．より多重のアンローリングは保守の問題を考えると問題である．本論文に述べられたこれらの高速化手法は精度を悪化させず，しかも汎用的である．

1. はじめに

まず帯行列を係数とする連立一次方程式に対する直接解法について述べる．一般的な帯行列の場合には帯ガウス[1),2)]と Martin と Wilkinson の帯ガウス[3),4)]を使用する．後者は右辺の計算に重点を置いた算法である．対称正定値帯行列の場合には対称帯ガウス[1),5)]を使用する．これらの算法を大規模な問題に適用したときのメモリ不足に対処するため，主記憶とディスク間のページスワップに配慮してアルゴリズムを作成する．

次にベクトル計算機と汎用計算機での高速化手法について述べる．ベクトル計算機の普及につれて大規模な線形計算が日常的に行われるようになったが，ベクトル計算機の性能を生かすためにはプログラムに手を加える必要がある．本論文ではベクトル計算機と汎用計算機の両方に好ましい高速化手法を帯行列に対する直接解法に適用した結果について述べ，その問題点を明らかにする．

プログラムの作成にあたっては以下の点に注意する．

（1） 精度の保証された一般的なアルゴリズムを用いる．

（2） アルゴリズムに忠実にプログラムを作る．

（3） 特定の計算機だけに高速とはしない．

† High Performance Band Equations Solvers by HIDEHIKO HASEGAWA (University of Library and Information Science).

†† 図書館情報大学

2. 帯行列に対するガウスの消去法

帯行列の場合には，係数行列と作業領域を格納するのに必要なメモリ容量が一番の問題である．

部分軸選択を行うクラウト法を帯行列に適用すると方程式の交換によって帯行列が密行列化し，最悪の場合には下三角部分は密で上三角部分は2倍の帯半幅を持った行列となる．これではメモリ容量を節約した意味がなく，部分軸選択が必要な帯行列にクラウト法は使用できない．

ガウスの消去法では部分軸選択のために上三角行列に帯半幅の2倍が必要になるだけである[2)]．したがって帯行列を係数とする連立一次方程式の解法としてはガウスの消去法を基本にするのが良い．

2.1 帯ガウス

帯行列を係数とする連立一次方程式にガウスの消去法を適用したときの第 k 段では，非対角帯半幅を ml として，方程式の交換がなければ i は $k+ml$ 行まで，j は $k+ml$ 列までを消去の対象にすれば良い．

k 段で第 k 番方程式と $ip(k)$ 番方程式との交換が起こったとする．下三角部分の $k-1$ 列まではすべて0なので，k 列から上三角部分の非零要素の端までが交換対象となる．$ip(k)=k+ml$ 行と交換が起こった場合に第 k 番方程式は非零要素が $k+2ml$ 列までとなり，帯半幅が一番広がる．k 番方程式の非零要素の右端が $k+2ml$ 列になったとき，k 番方程式による消去を行うと $k+1$ 番から $k+ml$ 番までの方程式の $k+2ml$ 列が非零になるが，右帯半幅は常に $2ml$ 以内に収まる．

次は必要な領域 $(3ml+1)\times n$ ワードをメモリ上に

どう格納するかが問題である．大規模計算を行う際には，プログラムがページスワップの影響を受けにくいようにすることが重要である．そのためには連続的なメモリ参照でワーキングセットが小さいほうが良い．ワーキングセットとは第 k 段の消去過程で実効的に必要とされるメモリ容量のこととする．

方程式ごとメモリに格納する横方向番地付けの行型ガウスのワーキングセットは $3ml^2 \fallingdotseq (ml+1)\times(3ml+1)$，未知数ごとメモリに格納する縦方向番地付けの列型ガウスのワーキングセットは $6ml^2 \fallingdotseq (3ml+1)\times(2ml+1)$ である．これから帯行列に対して方程式入れ換え方式の部分軸選択を行うときには横方向番地付けをした行型ガウスが良い．

行型ガウスの k 段の消去では

```
amax=|a_kk| ; ip(k)=k
do i=k+1, min(k+ml, n)
    if |a_ik|>amax then
        amax=|a_ik| ; ip(k)=i
do j=k+1, min(k+2ml, n)
    a_k,j ⇄ a_ip(k),j
do i=k+1, min(k+ml, n)
    a_ik=−a_ik/a_kk
    do j=k+1, min(k+2ml, n)
        a_ij=a_ij+a_ik*a_kj
```

となり，このような算法を帯ガウスと呼ぶ[1),2)]．

実際の帯ガウスのプログラムでは係数行列を

$$a'_{j-i,i}=a_{ij}$$

と変換して

$$a'(-ml:2ml,n)$$

という配列に収める．最内側のループを最適化するため，ループ内で不変な a_{ik}（実際は $a(k-i,i)$）にスカラ t を用いて無駄な添字演算を省く．さらに a_{kj}（実際は $a(j-k,k)$）は第 k 段の消去過程では不変なので添字計算の簡単な一次元配列を使用する．新たに $2ml$ 回の代入操作が必要なものの，$2ml^2$ 回の添字計算が簡単となり演算時間は減少する．帯ガウスの主部をプログラムふうに書くと

```
do j=k+1, min(k+2ml, n)
    wk(j)=a(j−k, k)
do i=k+1, min(k+ml, n)
    a(k−i, i)=−a(k−i, i)/a(0, k)
    t=a(k−i, i)
    do j=k+1, min(k+2ml, n)
        a(j−i, i)=a(j−i, i)+t*wk(j)
```

となる．

帯ガウス全体の演算量は，ベクトルのスカラ倍をベクトルに加える形の制限三項演算

$$a_i=a_i+t*b_i$$

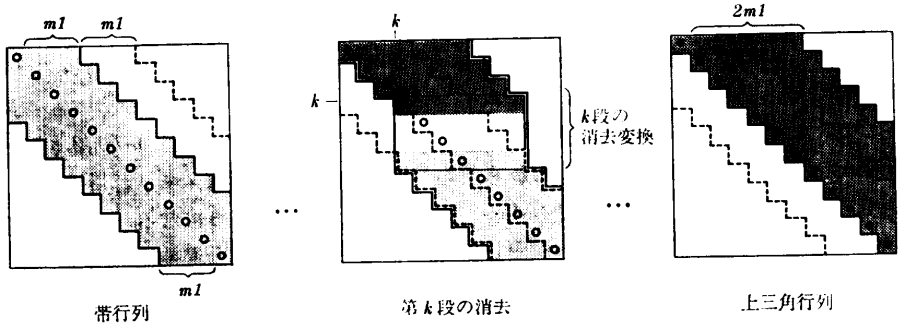
を基準にして約 $2ml^2n$ 回である．

2.2 Martin と Wilkinson の帯ガウス

帯ガウスの右辺の前進消去過程に現れる

```
do i=k+1, min(k+ml, n)
    b(i)=b(i)+a(k−i, i)*b(k)
```

の処理は不連続なメモリ参照となりページスワップの影響を受けやすく，反復解法などには向かない．帯ガウスと同一のメモリ容量で連続なメモリ参照にするのが Martin と Wilkinson の帯ガウスであり，局所的なメモリ参照が改善されるほかに，ページスワップの対象になるメモリ容量が帯ガウスの約半分になる[3),4)]．



帯幅 $2m1+1$ の範囲で消去が可能

図 1 帯ガウスの前進消去過程

Fig. 1 The forward elimination process on band Gaussian elimination.

帯ガウスの前進消去過程は図1のようになるので，消去変換に用いるαを分離すれば必要な領域の帯幅は$2ml+1$である．帯幅$2ml+1$の長方形領域で帯ガウスを実行するのがMartinとWilkinsonの帯ガウスである．データの格納方法は異なるが，演算順序は同じなので計算精度は同等である．

係数行列は横方向番地付けで格納する．帯ガウスでは第k段の消去過程で0にされた部分にαをいれておいたが，Martinと Wilkinsonの帯ガウスでは消去後に結果を1列シフトする．主要部分をプログラムふうに書けば，

```
do i=k+1, min(k+ml, n)
  ac(i-k, k)=-a(k-i, i)/a(0, k)
  do j=k+1, min(k+2ml, n)
    a(j-k-1, i)=a(j-k, i)+
      ac(i-k, k)*a(j-k, k)
```

となり，対応する右辺の前進消去過程は

```
do i=k+1, min(k+ml, n)
  b(i)=b(i)+ac(i-k, k)*b(k)
```

と連続的なメモリ参照になる．

ワーキングセットは$2ml^2 \fallingdotseq (2ml+1)ml+ml$で帯ガウスの約2/3である．前進消去過程のメモリ参照は上三角部分が$(2ml+1)n$，αを格納する部分が$ml \cdot n$，合計は$(3ml+1)n$になる．これは帯ガウスと等しい．右辺の計算のメモリ参照は，前進消去過程が$ml \cdot n$，後退代入過程が$(2ml+1)n$，合計は$(3ml+1)n$である．帯ガウスの右辺の計算のメモリ参照は前進消去過程，後退代入過程とも$(3ml+1)n$である．

右辺の計算を重視した算法について，同一の係数行列に対する右辺の計算をどのくらい繰り返すとCPUタイムが有利になるかに注目する．MartinとWilkinsonの帯ガウスでは演算量がほぼ同じでメモリ参照が改善されているが，前進消去過程はシフト操作のために計算が複雑になる．左辺に必要なCPUタイムは帯ガウスが短いが，右辺に必要なCPUタイムはMartinと Wilkinsonの帯ガウスが短い．これから方程式を繰り返して解いた場合に，帯ガウスとMartinと Wilkinsonの帯ガウスのCPUタイムが等しくなる繰り返し回数を計算できる．この回数を平衡点と呼ぶ[4]．平衡点よりも繰り返しが多い場合はMartinと Wilkinsonの帯ガウスのCPUタイムが少ない．問題が大規模になるにつれてページスワップの影響が大きくなるため，平衡点は下方に移動すると考えられる．

2.3 対称帯ガウス

対称正定値行列が係数の場合には軸選択が不要なのと消去の過程で対称性が保たれることから，一般にはコレスキー分解が用いられるが[1),3),6)]，対称帯ガウスを用いても容易に解くことができる[1),5)]．

帯コレスキー分解は帯を保ったままで，対称正定値行列をU^TDUまたはU^TUに分解する算法である(Uは上三角行列，Dは対角行列)．古典的なコレスキー分解を用いて$A=U^TU$と分解するためには$\sqrt{}$演算が必要なので，方程式を解く目的では改訂コレスキー分解 (Modified Cholesky decomposition) を用いて$A=U^TDU$と分解するのが一般的である．コレスキー分解では内積演算が中心なのでストア命令が少なくて済むが，精度を得るためには演算の順序に注意が必要である．帯コレスキー分解は汎用計算機では対称帯ガウスよりも高速だが，ベクトル計算機では対称帯ガウスが高速である[5)]．また高速化手法の適用も難しいため本論文では扱わない．

対称帯ガウス[1),5)]は軸選択を行わず上(下)三角部分だけでガウスの消去法を行う算法である．対称正定値行列が係数の場合，軸選択が不要なのと対称性が保たれることから消去範囲は帯ガウスの約1/4になる．しかも消去変換に用いるαはa_{ik}と対称な位置にあるa_{ki}を用いて作ることができるので，αを格納する必要はない．対称帯ガウスのアルゴリズムは

```
do i=k+1, min(k+ml, n)
  t=-aki/akk
  do j=i, min(k+ml, n)
    aij=aij+t*akj
```

となる．結果の上三角行列は$A=U^TDU$と分解したときのDUになる．帯ガウスと同様に実際のプログラムでは横方向番地付けで格納し，一次元配列などを用いて無駄を省く．

制限三項演算の回数は$ml^2n/2$回で帯ガウスの約1/4となる．必要なメモリ容量は$(ml+1)\times n$ワードで帯ガウスの約1/3になる．ワーキングセットはml^2で，メモリ参照は前進消去過程，右辺の前進消去過程，後退代入過程とも$ml \cdot n$であり，どの過程においても連続的である．

3. アンローリング

アルゴリズムどおりのプログラムでもベクトル計算機を用いれば高速に実行されるが，ベクトル計算機の性能を出しきるためには高速化手法の適用が必要である．文献 7) にはベクトル計算機向けの高速化手法として

(a) ベクトル化制御行の利用
(b) ループ・アンローリング
(c) ベクトル長の増加
(d) コンパイラへのデータ構造の明示
(e) メモリ競合の防止

があげられている．(c)は消去範囲から決まる．(e)は連続的なメモリ参照なので問題ない．(d)はプログラムを実際に書くときの注意事項で，(a)はコンパイラがうまく動かないときに使われるいわば「奥の手」で一般性がない．ここでは(b)のループ・アンローリングを中心にメモリとベクトル演算器間のデータ転送に注意しながら高速化を図る．

アンローリングはループ内の演算密度を上げて高速化を図る手法である．ベクトル処理の機構を複数組持った並列パイプライン方式のベクトル計算機ではパイプラインの並列実行により，汎用計算機ではループオーバヘッドの減少により高速化が図られる．以下，どのようなアンローリングが好ましいかを検討する．

ガウスの消去法は3重ループ内の演算なので，段 k，行 i，列 j の3通りのループについてアンローリングが可能である．並列パイプライン方式のベクトル計算機では同時実行が可能なベクトル演算器数と演算器にデータを供給するパイプライン数の関係が問題となる．汎用計算機ではメモリとレジスタ間のロード・ストアとループの判定回数などの演算以外の命令の数が主な問題となる．連続的なメモリ参照なのでページスワップの影響は考えなくても良い．消去演算の数は一定である．議論を簡単にするためアンローリングは2重に限定し帯ガウスのみについて述べる．その他のアルゴリズムについても同様に適用可能である．

ここでは，演算およびロード・ストアのパイプラインが十分に用意されていて並列に動作すると仮定する．

3.1 並列パイプライン方式のベクトル計算機の場合

帯ガウスの主部

```
do j=k+1, min(k+2ml, n)
  a(j-i, i)=a(j-i, i)+t*wk(j)          (1)
```

ではロードパイプ2，ストアパイプ1が使われる．

段 k についてのアンローリングを施して2段同時[8]にすると

```
do k=1, n, 2
  do i=k+1, min(k+ml, n)
    do j=k+1, min(k+2ml, n)
      a(j-i, i)=a(j-i, i)+                 (2)
        t*wk(j)+t1*wk1(j)
```

となる．ループ内で(1)の2倍の演算が行われ，実行時間は(1)の半分になる．2段同時はロードパイプ3，ストアパイプ1が同時に働く．2段同時はロードパイプを並列に稼動させるのが特徴である．段に関するアンローリングはプログラム全体に影響が及ぶため，実際は2段が限度であろう．

行 i についてのアンローリングを施して2行同時にすると

```
do k=1, n
  do i=k+1, min(k+ml, n), 2
    do j=k+1, min(k+2ml, n)              (3)
      a(j-i, i)=a(j-i, i)+t*wk(j)
      a(j-i-1, i+1)=a(j-i-1, i+1)+
        u*wk(j)
```

となる．ループ内で(1)の2倍の演算が行われ，実行時間は(1)の半分になる．ロードパイプ3，ストアパイプ2が同時に働く．2行同時はストアパイプとロードパイプが同じ数だけ増えるのが特徴である．ループ内の演算密度に比べて必要なパイプラインが多いため，小さい倍率でパイプラインが不足する．

列 j についてのアンローリングを施して2列同時にすると

```
do k=1, n
  do i=k+1, min(k+ml, n)
    do j=k+1, min(k+2ml, n), 2           (4)
      a(j-i, i)=a(j-i, i)+t*wk(j)
      a(j+1-i, i)=a(j+1-i, i)+
        t*wk(j+1)
```

となる．ループ内で2倍の演算が行われ，ロードパイ

プ3（または4），ストアパイプ2が同時に働く．最内側のループに対するアンローリングはループ長が半減し(c)の条件を満たさないため問題である．

2段同時に行 i についてのアンローリングを施して2段2行同時にすると

```
do k=1, n, 2
  do i=k+1, min(k+m1, n), 2
    do j=k+1, min(k+2m1, n)
      a(j-i, i)=a(j-i, i)+
        t*wk(j)+t1*wk1(j)
      a(j-i-1, i+1)=a(j-i-1, i+1)
        +u*wk(j)+u1*wk1(j)
```
(5)

となる．ループ内で(1)の4倍の演算が行われ，実行時間は(1)の1/4になる．2段2行同時はロードパイプ4，ストアパイプ2が同時に働く．4行同時にすれば同じ演算密度にできるが，4行同時ではロードパイプ5，ストアパイプ4が必要である．2段同時と2行同時を組み合わせることでロード・ストアパイプを有効に活用できる．

演算パイプラインが不足しない場合には2段2行同時にアンローリングするのが良い．演算パイプラインが不足気味でもアンローリングの悪影響は出にくいので，パイプラインが少ない場合にも2段2行同時は効果的であろう．必要なパイプライン数を表1に示す．

3.2 汎用計算機の場合

汎用計算機では同時に複数の演算が実行できない．消去演算の回数は一定なので，主としてロード・ストア命令と分岐命令などの演算以外の命令の実行回数が減少することで高速化が図られる．

ロード・ストア命令 (LD, STD) の実行回数が少なければ，メモリとレジスタ間で無駄な移動を行わずに演算が行われる．分岐命令 (BCT) の実行回数が少なければ，ループ内で多くの演算が行われる．汎用計算機でもパイプライン処理が行われているため，分岐命令などのパイプライン処理が中断する命令は少ないに越したことはない．アンローリングを施したときの命令数の合計を表1に示す．

2段同時ではロード・ストア命令の実行回数が減少し，2行同時と2列同時ではロード命令の実行回数だけが減少する．分岐命令の実行回数はどのアンローリングの場合も半減する．2段2行同時にするとロード・ストア命令，分岐命令の実行回数ともさらに減少する．アンローリングによって副次的な処理は必要になるが，2段2行同時にアンローリングしたプログラムは汎用計算機でも高速に実行できる．単一パイプライン方式や，パイプラインが少ないベクトル計算機の場合にも，ループあたりの前処理の負担が減るため汎用計算機と同様な効果が得られる．

より大きな倍率のアンローリングを施しても，ロード・ストア命令や分岐命令の実行回数の減少の割合はわずかである．プログラムの分かりやすさを考えれば2段2行同時が良いだろう．

実際のプログラムではすべての処理が一様に実行さ

表1 命令数の合計とパイプライン数
Table 1 The operation count and number of pipelines.

左辺: left side　　m 帯半幅，n 元数

	汎用計算機			演算密度	ベクトル計算機		
	ロード LD	ストア STD	分岐 BCT	演算パイプ	ロード パイプライン	ストア パイプライン	ループ長
帯ガウス(1)	$4m^2n$	$2m^2n$	$2m^2n$	1	2	1	$2m$
2段同時(2)	$3m^2n$	m^2n	m^2n	2	3	1	$2m$
2行同時(3)	$3m^2n$	$2m^2n$	m^2n	2	3	2	$2m$
2列同時(4)	$3m^2n$	$2m^2n$	m^2n	2	3(4)	2	m
2段2行同時(5)	$2m^2n$	m^2n	$m^2n/2$	4	4	2	$2m$

右辺: right side

	前進消去過程			後退代入過程*		演算密度
	ロード LD	ストア STD	分岐 BCT	ロード LD	分岐 BCT	演算パイプ
帯ガウス	$2mn$	mn	mn	$4mn$	$2mn$	1
2段同時	$\frac{3}{2}mn$	$mn/2$	$mn/2$	$3mn$	mn	2

* 後退代入過程は内積演算となる．

表 2 FORTRAN 77 ソースプログラム行数
Table 2 Number of source statements.

アンローリング	左辺: left side			右辺: right side	
	なし	2段同時	2段2行同時	なし	2段同時
帯ガウス	40	88	91	22	40
Martin らの帯ガウス	46	115	171	23	42
対称帯ガウス	19	35	40	16	29

コメント行は除く

れるように配慮してある．これはプログラムを分かりやすく保ちメンテナンスを容易にするためである．既存のプログラムに改良を加えるときには変数領域の制限などから場合分けの必要があるが，新規に作成するときにはメモリが多少余計に必要でも場合分けは避けたほうが良い．とはいえシフト操作を含んだ演算のアンローリングでは場合分けが避けられなかった．各種アンローリングを施した FORTRAN 77 ソースプログラムの行数を表 2 に示す．

3.3 右辺に対するアンローリング

右辺の前進消去過程は2重ループ内の演算なのでベクトル計算機向きのアンローリングは一種類に限定される．右辺の前進消去過程をアンローリングするためには左辺を2段同時にする必要がある．逆に左辺を2段同時や2段2行同時にすると自動的に右辺のアンローリングが図られる．

後退代入過程のアンローリングも可能である．

4. 数値実験

実際に計算機を用いて CPU タイムの実測を行い

(1) アンローリングの効果

(2) 平衡点

の2点について考察を行う．

実測には並列パイプライン方式のベクトル計算機として S-820/80，汎用計算機として M-260 H，中間的な計算機として IAP (Integrated Array Processor) 付の M-682 H を用いた．計算は2次元の場における拡散方程式を差分法で近似したときに現れる帯行列を用いて倍精度で行った．次元 n は $m1(m1+1)$ と $2m1(m1+1)$ の2通りである．

残差ベクトルの1ノルムと2ノルムについて精度の比較を行ったが，一般的なプログラムと比較して有意な差は見られなかった．帯ガウスと Martin と Wilkinson の帯ガウスの精度は全く同一であった．

実測結果を表 3～表 5 に示す．

4.1 アンローリングの効果

ベクトル計算機 S-820/80 では帯ガウスの左辺の計算にコンパイラによるアンローリングが施され，それぞれが8行同時，2段4行同時，2段4行同時として測定されている．そのためベクトル計算機では帯ガウスが対称帯ガウスよりも速いことがある．ベクトル計算機では2段同時にアンローリングすると約 60% 程度，2段2行同時にアンローリングすると約 50～60% の CPU タイムで済む．右辺についても2段同時にアンローリングすると約 80% の CPU タイムで済む．右辺は2重ループなのでかなりの好成績といえよう．

汎用計算機上では2段同時，2段2行同時とも約 70% 程度の CPU タイムで済む．右辺の計算を2段同時にすると約 85% の CPU タイムとなる．

表 3 S-820/80 での実測結果 (単位: 秒)
Table 3 Results on a vector computer S-820/80 (s).

		左辺: left side									右辺: right side 100回繰り返し					
		帯ガウス			Martin らの帯ガウス			対称帯ガウス			帯ガウス		Martin らの帯ガウス		対称帯ガウス	
$m1$	n		2段同時	2段2行同時		2段同時	2段2行同時		2段同時	2段2行同時		2段同時		2段同時		2段同時
20	420	0.009	0.006	0.006	0.01	0.007	0.007	0.009	0.005	0.005	0.12	0.09	0.12	0.09	0.12	0.09
	840	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.25	0.19	0.25	0.19	0.24	0.17
50	2550	0.10	0.07	0.07	0.16	0.09	0.09	0.13	0.07	0.07	0.83	0.62	0.82	0.63	0.79	0.58
	5100	0.20	0.15	0.15	0.33	0.19	0.18	0.26	0.15	0.14	1.70	1.29	1.67	1.34	1.65	1.22
100	10100	0.83	0.67	0.69	1.36	0.84	0.76	1.07	0.60	0.55	3.62	2.91	3.53	2.79	3.54	2.63
	20200	1.67	1.28	1.34	2.89	1.81	1.63	2.14	1.22	1.11	7.31	5.98	7.52	6.16	7.17	5.39
150	22650	3.12	2.61	2.73	4.90	3.33	3.03	3.67	2.11	1.92	8.63	7.19	8.38	6.73	8.30	6.38

表 4 M-260 H での実測結果 (単位: 秒)
Table 4 Results on a conventional computer M-260 H (s).

		左辺: left side									右辺: right side 100 回繰り返し					
		帯ガウス			Martin らの帯ガウス			対称帯ガウス			帯ガウス		Martin らの帯ガウス		対称帯ガウス	
$m1$	n		2段同時	2段2行同時		2段同時	2段2行同時		2段同時	2段2行同時		2段同時		2段同時		2段同時
20	420	0.48	0.34	0.35	0.55	0.41	0.39	0.15	0.10	0.10	3.71	3.17	3.57	3.08	2.64	2.40
	840	0.98	0.71	0.70	1.12	0.84	0.80	0.30	0.21	0.20	7.52	6.38	7.24	6.20	5.29	4.54
40	1640	7.30	5.09	5.03	8.34	6.19	5.85	2.03	1.38	1.37	27.43	23.31	26.67	22.89	19.50	16.50
	3280	14.50	10.14	10.18	16.77	12.42	11.81	4.02	2.78	2.74	55.24	46.92	53.78	46.11	38.71	33.10
50	2550	17.47	12.27	12.18	20.21	14.78	14.16	4.70	3.23	3.23	52.96	44.74	51.28	44.05	36.79	31.26

表 5 M-682 H (IAP) での帯ガウスの実測結果 (単位: 秒)
Table 5 Results of band Gaussian elimination on M-682 H (s).

(a) $a(j-i,i)=a(j-i,i)+t*wk(j)+t1*wk1(j)$

		左返: left side			右辺: right side 100 回繰り返し	
$m1$	n		2段同時	2段2行同時		2段同時
20	420	0.02	0.03	0.03	0.22	0.32
50	2550	0.46	0.81	0.81	3.58	3.95
100	10100	9.14	14.81	14.71	25.10	27.30

(b) $\begin{cases} a(j-i,i)=a(j-i,i)+t*wk(j) \\ a(j-i,i)=a(j-i,i)+t1*wk1(j) \end{cases}$

		左辺: left side			右辺: right side 100 回繰り返し	
$m1$	n		2段同時	2段2行同時		2段同時
20	420	0.02	0.02	0.02	0.22	0.26
50	2550	0.46	0.46	0.46	3.58	3.56
100	10100	9.39	7.82	7.81	25.21	24.44

IAP 付の M-682 H では2段同時にすると2倍の CPU タイムが必要となる (表5(a)参照). 2段同時のオブジェクトでは最内側ループが図2(a)のような VEMD (Vector Elementwise Multiply) と VEAD (Vector Elementwise Add) になっている. これはコンパイラが制限三項演算ループを認識していないことを示す. 制限三項演算ループを明示するため, 意図的に

$$a(j-i,i)=a(j-i,i)+t*wk(j)$$
$$a(j-i,i)=a(j-i,i)+t1*wk1(j) \qquad (6)$$

とすると図2(b)のように VSMAD (Scalar Multiply and Add) のループになる. ベクトル命令の実行時間が同じだとすると表5(a)の約半分の実行時間になるはずである. 実際, 表5(b)ではそうなっている. (6)を汎用計算機でコンパイルするとループ内に余計なストア命令が1つ入るため表4の結果よりもわずかに遅くなる.

すべての機種のことを考えると(6)のようにする必要があるが, ベクトル計算機では2段同時(2)を2つの VMAD (Scalar Multiply and Add) に展開できてコンパイラによるアンローリングまでできるのだから, この問題はコンパイラへの注文事項とするのが妥当であろう. 逆に線形数値計算の分野でベクトル計算機とか IAP の性能をひきだすためには, 柔軟な制限三項演算の適用は必須である. この例に限らず, 現時点のコンパイラには問題がある.

帯行列の場合にはループ長が比較的短いことを考えれば, 2段同時, 2段2行同時などのアンローリングから予想どおりの効果が得られたといえよう. ベクトル計算機で高速に実行したい場合には, 人手による陽アンローリングが必要である[9]. IAP のように演算結果を常にメモリへ戻す場合にも2段同時などのアンローリングは有効である.

4.2 平衡点

平衡点が存在するためには Martin と Wilkinson の帯ガウスの右辺が帯ガウスの右辺よりも高速なことが必要である. 汎用計算機では非対角帯半幅 $m1$ が 50 のとき, 平衡点は次元数の約 10% 程度である. リアルメモリ方式のベクトル計算機の場合には平衡点が存在しにくいが, 行列の次元が大きくなると平衡点が現れ, 次元数の 10% 程度になる.

メモリ制約がきつくなるとベクトル計算機でも Martin と Wilkinson のアルゴリズムの効果が現れるので, 反復解法などで同一の係数行列に対して右辺の計算を繰り返す場合は Martin と Wilkinson の帯ガウスが良い.

```
      LAB#0152 EQU *
1830          L     15,192(0,13)
              LA    14,WK
              ST    14,LTM#0270
              LA    14,ITM#0036
              VEMD  2,14,2258(0)
1830          LA    14,ITM#0039
              LA    9,A
              ST    9,LTM#0271
              VEAD  2,14,208(0)
1830          L     8,196(0,13)
              LA    6,A
              LA    14,ITM#0042
              LA    7,WK1
              ST    7,LTM#0272
              ST    6,LTM#0273
              VEMD  2,14,2258(0)
1830          LA    14,ITM#0043
              VEAD  2,14,208(0)
       * INLINE
              L     4,LTM#0225
              A     5,=F'2048'
              A     4,=F'2048'
              ST    4,LTM#0225
      LAB#0142 EQU *
              L     0,GTM#0222
              S     0,=F'256'
              ST    0,GTM#0222
              B     15,LAB#0141
      LAB#0143 EQU *
      LAB#0029 EQU *
1850  300     EQU *
```

(a)

$$a(j-i,i)=a(j-i,i)+t*wk(j)+t1*wk1(j)$$

```
      LAB#0143 EQU *
2850          L     15,LTM#0267
              LA    9,A
              LA    14,WK
              ST    14,LTM#0265
              LA    14,ITM#0036
              ST    9,LTM#0266
              VSMAD 2,14,2266(0)
2860          L     8,LTM#0270
              LA    15,A
              LA    14,ITM#0039
              LA    1,WK1
              ST    1,LTM#0268
              ST    15,LTM#0269
              VSMAD 2,14,2266(0)
       * INLINE
2840  LAB#0029 EQU *
2880  300     EQU *
```

(b)

$$\begin{cases} a(j-i,i)=a(j-i,i)+t*wk(j) \\ a(j-i,i)=a(j-i,i)+t1*wk1(j) \end{cases}$$

図 2 M-682 H (IAP) のオブジェクトプログラム
Fig. 2 Object programs on M-682 H (IAP).

5. ま と め

帯行列を係数とする連立一次方程式の直接解法を検討した．方程式を少数回解く場合には帯ガウス，同一の係数行列に対して右辺の計算を繰り返す場合には Martin と Wilkinson の帯ガウス，係数行列が対称正定値ならば対称帯ガウスが良い．プログラムを作成するうえではメモリ参照に注意を払う必要があった．

ベクトル計算機と汎用計算機の両方で高速化をするためには2段2行同時のアンローリングが良い．IAP やベクトル計算機で高速化を図るためには，制限三項演算の適用が必須である．アンローリングで重要な点はメモリとレジスタ間の無駄なデータのやりとりをなくすことである．この点では Dongarra らの方針と同じである[10]．彼らは線形数値計算に現れる基本的な演算を高速化して全体を組織的に高速化しようとしている．ここでは帯行列を係数とする連立一次方程式をガウスの消去法系統の算法で解くためのアルゴリズムについて高速化を行った．影響が LU 分解全体に及ぶため作業は大変なものの高速化の割合は大きい．より多重のアンローリングは高速化のメリットが少ない割にプログラムが複雑になる．高速化と保守の問題を併せて考えれば2段2行同時程度のアンローリングが最適であろう．

高速化手法の適用をコンパイラだけに期待するのは難しい．すべてのコンパイラでベクトル計算機と汎用計算機の両方に効果的なアンローリングができれば問題の幾分かは解決するが，現時点では人手による陽アンローリングにも十分価値が認められるだろう．

本論文では右帯半幅と左帯半幅が同じものと仮定して議論を進めたが，右帯半幅と左帯半幅は異なっていても良い．プログラムは容易に拡張できる．

謝辞 本論文のテーマ全体にわたって指導と討論をしてくださった村田健郎先生に深く感謝します．

参 考 文 献

1) 村田健郎：線形代数と線形計算法序説，p. 225，サイエンス社，東京 (1986).
2) 長谷川秀彦：帯行列を係数とする連立一次方程式の解法(I)，図書館情報大学研究報告，Vol. 6, No. 1, pp. 45-59 (1987).
3) Wilkinson, J. H. and Reinsch, C.: *Handbook for Automatic Computation*, Vol II—*Linear Algebra*, p. 439, Springer-Verlag, Berlin (1971).
4) 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. 89-111 (1987).
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(III)，図書館情報大学研究報告，Vol. 7, No. 1, pp. 61-73 (1988).
6) 森 正武：数値計算プログラミング，p. 342，岩波書店，東京 (1986).
7) 二宮市三：スーパコンピュータと数学ライブラリ，情報処理，Vol. 27, No. 11, pp. 1235-1241 (1986).
8) 村田健郎：スーパーコンピュータと線形計算，コンピュータから生まれた新しい数学（別冊数学セミナー），pp. 193-208 (1986).
9) 村田健郎，小国 力，三好俊郎，小柳義夫：工学における数値シミュレーション，p. 322，丸善，東京 (1988).
10) Dongarra, J. J. and Sorensen, D. C.: *High Performance Computers and Algorithms from Linear Algebra, Large Scale Eigenvalue Problems*, pp. 15-36, North-Holland, Amsterdam·New York·Oxford·Tokyo (1986).

(昭和63年7月 8 日受付)
(平成元年1月17日採録)

長谷川秀彦（正会員）

1958 年横浜生．1980 年筑波大学第 1 学群自然学類（数学専攻）卒業．1983 年同大学院社会工学研究科（経営工学専攻）中退．同年図書館情報大学助手．学術修士．現在，連立一次方程式の解法，固有値解析などの線形数値計算法に興味を持つ．ACM 会員．