Vol. 25 No. 5 情報処理学会論文誌 Sep. 1984

直接解法による連立1次方程式のコンピュータ解法の特性解析†

小国 力†† 後保 範††
西方政春†† 長堀文子††

コンピュータによる技術計算量が増大するにつれて，技術計算用ソフトウェアがもつ製品としての性能がきわめて重要となり，計算時間や計算費用を通じて利用者の研究活動や設計活動に多大の影響を与えている．とくに，最近普及しはじめたスーパコンピュータを使って計算する場合にこの影響は著しい．性能のよい技術計算用ソフトウェアを作成する上で，CPU 時間，経過時間，記憶容量，およびファイル所要量といった資源の特性をハードウェアの種類や FORTRAN コンパイラに応じて捉え，これら資源をバランスよく最小化したソフトウェアの設計が必要である．逆に，こうした態度がハードウェアやコンパイラの改良にもつながると考える．さらに，連立1次方程式の解法でいえば，取り扱う行列そのものの特性，つまり，係数行列の形，規模，および sparsity が上述の資源とどう関わりをもつかを明らかにし，扱う行列に相応しいアルゴリズムを設計する必要がある．連立1次方程式の直接解法に対しここで用いた特性解析は，これまで数値解析とプログラムの間にあったギャップを埋めるための試みの一つである．

1. はじめに

産業界における大型の技術計算のうち大半が行列計算に関係し，その中心を占めるのが連立1次方程式の解法である．連立1次方程式はガウス消去法やクラウト法に基づく LU 分解による直接解法か，SOR 法や PCG 法などの反復法が用いられているが，標準パッケージとしては圧倒的に直接解法が使われている．スーパコンピュータが普及するにつれて流体力学などの計算が可能になってくるに従い，反復法も盛んになってこよう．

LU 分解に基づくプログラムの性能は，基本となるハードウェアから始まって，中間の特殊なハードウェア，さらに各種のソフトウェアの特性のいかんにより大きく左右される．これらの中間諸元は最終製品である技術計算ソフトウェアに図1に示す順序で影響を与える．

さらに，これら諸元のほかに，扱う行列そのものの性質，つまり，係数行列の形，規模，および sparsity が主記憶装置や補助記憶装置に関係して重要になる．また，LU 分解の途中で現われる fill-in 要素の個数を少なくすることも重要になる．

本論文では LU 分解法を FORTRAN コンパイラ特性およびハードウェア特性の下に考察している．FORTRAN コンパイラの特性としては

（1）列方式によるデータの配置
（2）内積計算の精度および速度

を挙げることができる．データの配置方法としては2次元配列と1次元配列とがある．1次元配列を使うときには，記憶装置容量の節約と，2次記憶装置の非同期入出力が可能になるが，添字処理による負荷がかかる．ガウス法では消去計算に伴う三項演算が，クラウト法では内積計算が必要だが，内積計算は途中の演算結果を主記憶装置に格納することなく，すべての演算がレジスタ上で行われるため，性能はよくなる．さらに，ハードウェア特性としては，

（1）仮想記憶装置
（2）ベクトルプロセッサ

を取り上げる．仮想記憶装置容量を越えたときの領域管理のやり方が，経過時間に致命的な影響を与える．ベクトルプロセッサにはさまざまな種類があるが，ここでは，日立製作所の S-810 を例にとる．他のパイプライン型ベクトルプロセッサとは 1st order iteration などが違うが，直接解法では使用しないので本論文の内容は一般的に当てはまると考えてよい．

LU 分解における演算量はよく知られているように $n^3/3$ のオーダである．しかし，FORTRAN プログラムとしてコンピュータ上で計算する場合には，このほかに DO ループ処理としての初期操作と判定・分岐が必要であり，代入演算に伴う主記憶装置との取出し，

† Characteristic Analysis for Computer Algorithms with Direct Methods of Simultaneous Linear Equations by TSUTOMU OGUNI, YASUNORI USHIRO, MASAHARU NISHIKATA and FUMIKO NAGAHORI (Software Works, Hitachi, Ltd.).
†† (株)日立製作所ソフトウェア工場

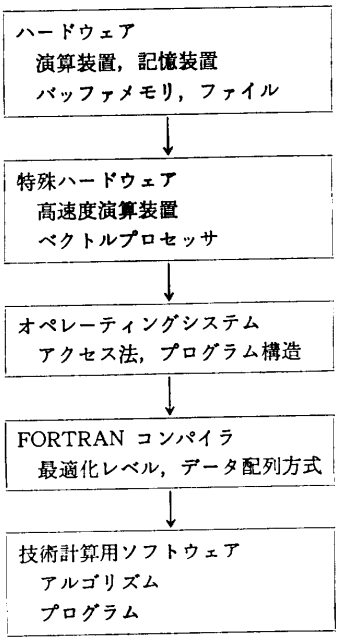


図1 性能に関する中間諸元
Fig. 1 Elements affecting the performance of engineering software.

格納が必要となる．これらの操作はハードウェア命令としては乗算や加算に匹敵するサイクル数を要し，たんにプログラムの実行速度を乗算回数だけの関数として捉える従来のやり方は妥当とはいえない．このため，本論文ではサイクル数の最小化という立場からアルゴリズムを評価していく．

2. *LU* 分解法の演算量

LU 分解に基づくアルゴリズムは

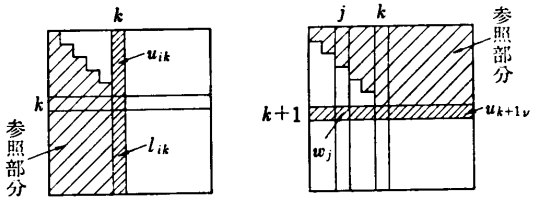
（1）一度分解が済めば，右辺を変更するときは代入計算で新しい解は簡単に求まる．つまり，分解に必要な演算量が $n^3/3$ のオーダに対し，代入計算に必要な演算量は n^2 のオーダにすぎない．

（2）もとの行列と *LU* 分解後の行列の構造が同じである．

という二つの特長により，直接解法の主流になっている．*LU* 分解法には，ガウス法とクラウト法があり，さらにその変種が考えられる．ガウス法では，消去計算（更新計算）である三項演算

$$a_{ij}'=a_{ij}-a_{ik}a_{kj}/a_{kk}=a_{ij}+t_k\times a_{ik}$$
$$t_k=-a_{kj}/a_{kk} \qquad (1)$$
$$(i,j=k+1,\cdots,n)$$

が基本であり，$|t_k|<\varepsilon$ のとき更新計算を行う必要がないので計算速度を向上でき，疎行列に向いている．



（a）積形式ガウス法　　（b）村田式クラウト法[7]

図2 ガウス法とクラウト法
Fig. 2 Gauss elimination method and Crout method.

クラウト法では内積計算

$$l_{kk}u_{kj}=a_{kj}-\sum_{\nu=1}^{k-1}l_{k\nu}u_{\nu j}$$

が基本である．いずれの場合にもFORTRANの列方式を意識したアルゴリズムを採用する必要がある．つまり，ガウス法もクラウト法も通常の方法をとらず，**図2**のようなアルゴリズムがよい．

ガウス法では，k 段階の消去計算である式(1)は

$$\begin{cases}a_{ij}^{(k)}=a_{ij}^{(k-1)}+t_k\times a_{ik}^{(k-1)}, & t_k=-a_{kj}^{(k-1)}/a_{kk}^{(k-1)}\\ a_{kj}^{(k)}=a_{kj}^{(k-1)}/a_{kk}^{(k-1)}\end{cases}$$
$$(i,j=k+1,\cdots,n)$$

と書ける．これは行列表示で[8]，

$$\boldsymbol{a}_j^{(k)}=\boldsymbol{a}_j^{(k-1)}+t_k\boldsymbol{a}_k^{(k-1)}=E_k\boldsymbol{a}_j^{(k-1)}$$
$$E_k=(\boldsymbol{e}_1,\cdots,\boldsymbol{e}_{k-1},\boldsymbol{\eta}_k,\boldsymbol{e}_{k+1},\cdots,\boldsymbol{e}_n)$$
$$\eta_k^T=(0,\cdots,0,1/a_{kk}^{(k-1)},-a_{k+1\,k}^{(k-1)}/a_{kk}^{(k-1)},\cdots,-a_{nk}^{(k-1)}/a_{kk}^{(k-1)})$$

と表せる．したがって，第 k 列による更新は第 k 列から作った変換行列 E_k を左から乗じることに等しい．一方，第 k 段階のはじめにおける第 k 列 $a_k^{(k)}$ は

$$\boldsymbol{a}_k^{(k)}=E_{k-1}\cdots E_1\boldsymbol{a}_k$$

に等しい．つまり，$\boldsymbol{a}_k^{(k)}$ は，第1～第$(k-1)$列から作った変換行列 E_j を左から順に $\boldsymbol{a}_k$ に乗じてやれば求まる．これは線形計画法では積形式とよぶ古くからの技法である．

クラウト法では行方向の内積計算を防ぐよう工夫する．つまり，第 k 段階では l_{kk} を求めたのち第 $k+1$ 行の要素 $l_{k+1,1},\cdots,l_{k+1,k}$ を順次求めて，作業領域 $w_1,\cdots,w_k$ に格納する．次に，$u_{k+1,k+2},\cdots,u_{k+1,n}$ を

$$l_{k+1,k+1}u_{ij}=a_{k+1,j}-\sum_{\nu=1}^{k}l_{k+1,\nu}u_{\nu j}$$
$$=a_{k+1,j}-\sum_{\nu=1}^{k}w_\nu u_{\nu j}$$

により求める．この場合，w_ν も $u_{\nu j}$ も列方向に連続的に並んでいるので，内積計算をそのまま使うことができる．このアルゴリズムを村田式クラウト法とよ

表 1 アルゴリズム別演算量（枢軸選択なし）
Table 1 Computational amounts in each algorithm.

アルゴリズム	除算	乗算	加算	取出し	格納	判定・分岐	DO ループ
ガウスジョルダン	n	$n(n-1)(n+1)/2$	$n(n-1)^2/2$	乗算回数$+n$	乗算回数$+n$	乗算回数$+n-1$	$(n-1)(n+5)/2$
外積型ガウス	n	$n(n-1)(n+1)/3$	$n(n-1)(2n-1)/6$	上に同じ	上に同じ	上に同じ	$n(n-1)/2$
村田式クラウト	n	上に同じ	上に同じ	$n(n+1)(n+3)/3$	$(n-1)(5n-2)/2$	$n(n-1)(2n-1)/6+(n-1)^2$	$(n-1)^2$
2列同時消去	n	上に同じ	上に同じ	乗算回数$+n$	乗算回数$+n$	乗算回数$+n-1$	$n(n-1)/4$
改訂コレスキー	n	$n(n-1)(n+4)/6$	$n(n-1)(n+1)/6$	上に同じ	$n(n-1)/2$	上に同じ	$n(n-1)/2$
帯行列用ガウス	n	*1	*3	上に同じ	乗算回数$+n$	上に同じ	$\beta(n-\beta)+\beta(\beta-1)/2$
帯行列用改訂コレスキー	n	*2	*4	上に同じ	$(n-m)m+m(m-1)/2$	上に同じ	$(n-m)m+m(m-1)/2$

*1 $(\beta+1)\alpha(2n-\alpha-1)/2-\beta(\beta-1)(\beta+1)/6$
*2 $m(m+3)(n-m)/2+m(m-1)(m+4)/6$
*3 $\alpha\beta(2n-\alpha-1)/2-\beta(\beta-1)(\beta+1)/6$
*4 $m(m+1)(n-m)/2+m(m-1)(m+1)/6$
α は下帯幅，β は上帯幅，m は帯幅

び，ガウス法よりサイクル数が少ないので，実記憶装置上にすべての密行列データが収まる場合に最も性能がよい．

LU 分解の演算量を標準的な行列について求めた表が**表 1** である．

表 1 には枢軸選択を行わない場合の演算量をあげている．また，この表には FORTRAN プログラムで LU 分解したときに要する主記憶装置とのデータのやり取り（つまり取り出しと格納）の回数，DO ループにおける判定・分岐回数，および DO ループそのものの処理回数を示してある．コンピュータ上で FORTRAN プログラムを実行するには，たんに四則演算だけでは不十分で，データの取出しと格納，および DO ループのための判定・分岐などの回数を求め，それぞれのハードウェア命令のサイクル数を乗じて総サイクル数を求める必要がある．例として，三項演算と内積計算をあるハードウェアで行うときの状態図を**図 3** に示す．

三項演算 $a_i=b_i\times c_i+d_i$ のときは 13 サイクル，内積演算 $s=s+b_i\times c_i$ のときは 12 サイクルを要し，そのうちの 6 サイクルが四則演算であるにすぎない．ここで，密行列に対する各手法の総サイクル数を**表 2** に示す．

帯行列の場合には帯部分を圧縮して処理するので添字処理が追加演算として生じる．この追加演算を**表 3** に示す．

追加演算のサイクル数が占める割合は，上帯幅 β，下帯幅 α の帯行列に対し表 3 より，$\beta=2\alpha=2m$ のとき，

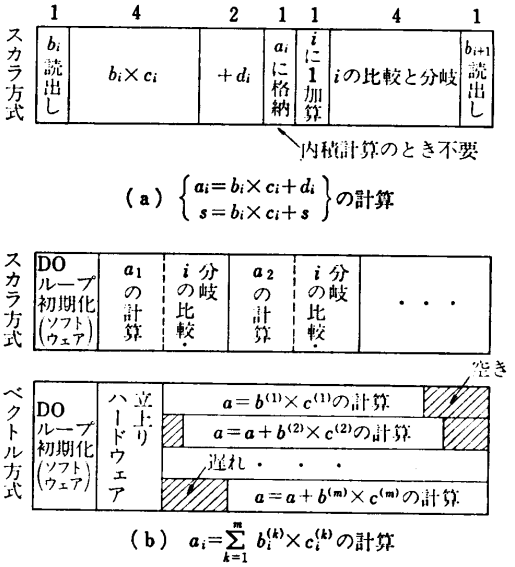


図 3 DO ループの処理
Fig. 3 Processing of DO loops.

帯ガウス法：$\dfrac{4(\beta+1)\alpha}{13\alpha\beta+11\alpha+20\beta}\fallingdotseq 0.3$

帯改訂コレスキー法：$\dfrac{3m+9}{12m+74}\fallingdotseq 0.25$

したがって，圧縮型の帯行列用ルーチンではサイクル数はそれぞれ 30% と 25% の増加になるが，記憶容量ははるかに少なくて済むようになる．記憶容量の必要量は

帯ガウス法：$(\alpha+\beta)/n$

帯改訂コレスキー法：$(m+1)/n$

であり，帯幅が n に比べて小さいときには実記憶装置

表 2 各演算サイクル数のオーダ（分解時のみ）
Table 2 Amount of hardware cycles in each algorithm.

アルゴリズム	乗算（4サイクル）	加算（2サイクル）	取出し（1サイクル）	格納（1サイクル）	DO ループ（20サイクル）	判定・分岐（5サイクル）	合計（サイクル数）	推定値	実測値
積形式ガウス	$n^3/3$	$n^3/3-n^2/2$	$n^3/3$	$n^3/3$	$n^2/2$	$n^3/2$	$13n^3/3+9n^2$	1.00	1.00
村田式クラウト	$n^3/3$	$n^3/3-n^2/2$	$n^3/3+4n^2/3$	$5n^2/2$	n^2	$n^3/3+n^2/2$	$4n^3+23n^2$	0.93	0.92
2列同時消去	$n^3/3$	$n^3/3-n^2/2$	$n^3/3$	$n^3/6$	$n^2/4$	$n^3/6$	$10n^3/3+4n^2$	0.77	0.74
2列同時2列消去	$n^3/3$	$n^3/3-n^2/2$	$n^3/3$	$n^3/6$	$n^2/8$	$n^3/12$	$35n^3/12+2n^2$	0.67	0.68
改訂コレスキー	$n^3/6+n^2/2$	$n^3/6$	$n^3/6+n^2/2$	$n^2/2$	$n^2/2$	$n^3/6$	$2n^3+13n^2$	0.47	0.49

注）実測値および推定値は $n=312$ の場合

表 3 1次元配列と帯行列用の追加演算（分解時のみ）
Table 3 Amount of additional computation necessary for one dimensional array and band matrix.

アルゴリズム	乗算（4サイクル）	加算（2サイクル）	取出し（1サイクル）	格納（1サイクル）	DO ループ（20サイクル）	判定・分岐（5サイクル）	追加演算（1サイクル）	合計（サイクル数）
ガウス法（1次元）	$n^3/3$	$n^3/3-n^2/2$	$n^3/3$	$n^3/3$	$n^2/2$	$n^3/3$	$8n^3/3$	$7n^3+9n^2$
改訂コレスキー（1次元）	$n^3/6+n^2/2$	$n^3/6$	$n^3/6+n^2/2$	$n^2/2$	$n^2/2$	$n^3/6$	$n^3/3$	$7n^3/3+25n^2/2$
帯ガウス（2次元）	$(\beta+1)\alpha n$	$\alpha\beta n$	$(\beta+1)\alpha n$	$(\beta+1)\alpha n$	βn	$(\beta+1)\alpha n$	$4(\beta+1)\alpha n$	*1
帯改訂コレスキー（2次元）	$m(m+3)n/2$	$m(m+1)n/2$	$m(m+3)n/2$	mn	mn	$m(m+3)n/2$	$3m(m+3)n/2$	$(15m^2+87m)n/2$

*1 $(17\alpha\beta+15\alpha+20\beta)n$

内で演算が可能になるという利点がある．

3. 枢軸選択法

連立1次方程式を精度よく解くには，消去計算ないし LU 分解の際に枢軸要素として列内または行内の絶対値最大の要素を選ぶ必要があり，このことを枢軸選択といっている．FORTRAN 言語のプログラムでは枢軸列の要素のうちで絶対値最大な要素をもつ行（枢軸行）を選び，この行と対角行とを交換する．積形式のガウス法ではこの行交換のやり方に3アルゴリズムがある．

方式1：行の交換は列方向の演算を妨げるので，枢軸列と交換すべき要素行の位置（つまり，枢軸行の行番号 $\nu_1, \nu_2, \cdots, \nu_k$）を記憶しておき，見かけ上は行の交換をやらない．このやり方は線形計画システムで用いられている．

方式2：対角行 k と枢軸行 ν_k を一度に交換する．

方式3：積形式のガウス法特有のもので，第 j 列（$j=1, \cdots, k-1$）による更新の際に，(k, j) 要素と (ν_k, j) 要素を入れかえるとともに，第 k 列については (i, k) 要素と (ν_i, k) 要素を $i=1, \cdots, k-1$ について入れかえる．

方式2の枢軸選択により追加される LU 分解時の演算量は表4に示すとおりである．

表 4 枢軸選択での追加演算量（行交換の場合）
Table 4 Amount of additional computation in pivotal selections.

種類	演算回数	サイクル	サイクル数
絶対値	$(n-1)(n+2)/2$	2	n^2
判定・分岐1	$(n-1)(n+2)$	7	$7n^2$
判定・分岐2	$n(n-1)$	6	$6n^2$
取出し	$2n(n-1)$	1	$2n^2$
格納	$2n(n-1)$	1	$2n^2$
合計			$18n^2$

表4のなかで，判定・分岐1とあるのは絶対値最大な要素を判定するための演算量であり，判定・分岐2とあるのは枢軸選択用の DO ループ処理のための演算量である．サイクル数はオブジェクト・コードを参照して得た．したがって，表4より，LU 分解の合計サイクルと枢軸選択のサイクル数の比は

$$\frac{18n^2}{13n^3/3+9n^2}=\frac{54}{13n+27}\fallingdotseq\frac{4}{n}$$

となり，n が大きいときはその影響は無視できる．方式1による枢軸選択の場合には，間接アドレッシングとして最内部 DO ループで5サイクル要するので，

$$\frac{18n^3/3}{13n^3/3}\fallingdotseq\frac{18}{13}\fallingdotseq 1.38$$

から，40% 程度遅くなる．

表 5 枢軸選択の実験結果 (単位：秒)
Table 5 Results of each pivotal selection method.

	ベクトルプロセッサ	次数 n			
		312	400	500	600
枢軸選択なし	あり	0.361	0.670	1.17	1.88
	なし	4.11	8.74	17.0	29.5
方式 1	あり	0.701	1.36	2.54	4.27
	なし	6.71	14.1	27.6	47.8
方式 2	あり	0.391	0.723	1.23	1.98
	なし	4.17	8.84	17.2	29.8
方式 3	あり	0.431	0.785	1.34	2.14
	なし	4.16	8.82	17.2	29.7

表 6 枢軸選択と作業領域
Table 6 Relation with pivotal selections and working area.

枢軸選択	行幅 α	列幅 β	増加量	作業領域	データ配置
列交換	$p+q$	q	qn	$(p+2q)q$	たて
行交換	p	$p+q$	pn	$(2p+q)p$	よこ

三つの方式による実験結果を表 5 に示す．行列の次数が大きくなって，仮想記憶装置を使うようになると，方式2はスワッピングを多用することになり，方式3が優位となる．したがって，線形計画法のように行列の要素をひんぱんに変えて LU 分解する場合を除くと，方式3を採用すべきである．

帯行列ガウス法の場合には，枢軸選択をすると行ないし列の幅が増える．もとの帯行列の行幅と列幅をそれぞれ p および q とすると，枢軸選択は表 6 のようになる．

ここで，列交換方式の場合のデータ配置は列方向(縦)に連続にとり，行交換方式の場合のデータ配置は行方向(横)に連続にとる．このほうが，実際に使う作業時の実記憶容量（作業領域：Working Set）が少なくて済む．したがって，$p>q$ のときは列交換方式の縦配置がよく，$p<q$ のときは行交換方式の横配置がよい．

4. 代入計算

LU 分解時に解を求めるための代入計算

$$\begin{cases} Ly=b \\ Ux=y \end{cases}$$

の場合にも，内積計算を行うかどうかによって内積型か三項型か分かれる．内積型の場合は計算が行方向にまたがるので望ましくない．したがって，参照領域を連続的に参照して内積型の速度を上げるため，代入計算に先立って LU 分解で得られた L と U を転置しておくことも考えられる．行列が変わらずに右辺だけを変更して幾通りも解きたい場合に転置行列による内積型は有利になる．

LU 分解時と代入計算時の乗算回数比率をおもな場合について求めると，表 7 が得られる．

表7より，帯幅が小さな帯行列の場合を除けば代入計算の比率は無視してよい．

5. DO ループのステップ幅と配列の次元数の関係

2章の表2に示したように三項演算や内積計算に占める判定・分岐のための負荷の割合は 5/13 または 5/12 であり，きわめて大きい．この負荷を減らすには数個(p 個)の三項演算や内積計算を最内部ループで行うことにより判定・分岐の回数を削減すればよい．ガウス法を例にとると，2次元配列方式のサイクル数のオーダは $13n^3/3$ であり，そのうち判定・分岐が $5n^3/3$，それ以外が $8n^3/3$ である．したがって，DO ループのステップ幅を p としたときには，最内部ループの判定・分岐，および添字計算はほぼ $1/p$ に減るから，このときのサイクル数は

$$8n^3/3+5n^3/3p$$

となる．ステップ幅が1のときの比 q は

$$q=\frac{8n^3/3+5n^3/3p}{13n^3/3}=\frac{8p+13}{13p}$$

となる．p を1から動かしたときの q の推定式と実測値を表 8 に示す．

2次元配列を複数個使うときは，領域を目一杯使い切ることはまれなので，主記憶装置の使用効率は悪く

表 7 LU 分解時と代入時の乗算回数比率
Table 7 Ratio of numbers of multiplications between LU decomposition and substitution.

アルゴリズム	分解時	代入時	比率		
			推定式	$m=15$	$m=25$
ガウス法	$n(n-1)(n+1)/3$	n^2	$3/n$	—	—
改訂コレスキー	$n(n-1)(n+4)/6$	n^2	$6/n$	—	—
帯ガウス	$(\beta+1)\alpha n$	$(\alpha+\beta+1)n$	$(\alpha+\beta+1)/(\beta+1)\alpha$*	0.10	0.06
帯改訂コレスキー	$m(m+3)n/2$	$2mn$	$4/(m+3)$	0.22	0.14

* $\beta=2\alpha=2m$ とした

表 8 DO ループのステップ幅 (n=312)
Table 8 Stride in DO loops.

ステップ幅	1	2	3	4	5	6	7	∞
推定値	1.00	0.80	0.74	0.71	0.69	0.68	0.67	0.62
実測値	1.00	0.77	0.70	0.68	0.65	0.64	0.64	—

表 9 1次元方式による DO ループのステップ幅 (n=312)
Table 9 Stride in DO loops in one-dimensional arrays.

ステップ幅	1	2	3	4	5	6	7	∞
推定値	1.00	0.68	0.57	0.52	0.49	0.47	0.45	0.36
実測値	1.00	0.69	0.60	0.50	0.50	0.47	0.46	—

なる．また，複数個の1次元配列をバラバラに確保しても使用効率は悪くなる．このため，限られた主記憶装置を有効に使うには，配列をすべて統合して一つの大きな1次元配列に組み入れてしまうとよい．たとえば，構造解析における大型の行列の LU 分解では要素数の大きな二つのベクトルの内積を計算するが，この二つのベクトルを単一の大きな1次元配列に連続して配置させる．このような配列の仕方を1次元配列方式とよぶ．1次元配列方式では，行列の要素を参照するときに要素位置（添字）の計算が9サイクル余分に要り，この添字計算に必要な余分なサイクル数は $3n^3$ となる．この値は FORTRAN コンパイラによって多少は異なるが，かなりな負荷となる．したがって，1次元配列の場合も，DO ループのステップ幅を p にするようなコーディングを採用すべきである．このときのサイクル数は

$$\frac{8}{3}n^3+\frac{1}{p}\left(\frac{5}{3}n^3+3n^3\right)=\frac{8}{3}n^3+\frac{14}{3p}n^3$$

となる．ステップ幅が1のときのサイクル数 $22n^3/3$ との比 r は

$$r=\frac{8n^3/3+14n^3/3p}{22n^3/3}=\frac{4p+7}{11p}$$

となる．p を1から動かしたときの r の推定値と実測値を表 9 に示す．

密行列用のガウス法と改訂コレスキー法を1次元配列方式で考えたときのサイクル数は，2章で示した表3に示してある．改訂コレスキー法の場合，サイクル数は16%増加するが，記憶容量はほぼ半分で済むという利点がある．

ステップ幅が1のとき1次元ガウス法と2次元ガウス法の速度比はサイクル比 22/13=1.69 に近いが，実測値は表 10 に示すとおりであった．

表 10 ガウス法の1次元方式と2次元方式の速度比
Table 10 Ratio of speed between one-dimensional array and two-dimensional arrays.

行列規模 n	8	16	32	64	128	192	256	312
実測値	1.39	1.61	1.77	1.88	1.85	1.85	1.84	1.86

クラウト法の場合は，1次元配列方式による負荷は4サイクルと少なくて済み，2次元配列方式との速度比は 1.33 程度になる．

6. 仮想記憶

大規模な連立1次方程式で実記憶装置にはるかに入りきれない場合を考える．村田式クラウト法では第 k 段階で求める値は1行分であり，仮想記憶装置との間にスワッピングを頻発させ，経過時間の増大を招き，実用に耐えない．これに反し，積形式のガウス法では第 k 段階で求める値は列であり，参照すべき領域も第1列から第 $(k-1)$ 列までの値である．このため，図 4 に示すように，もとの行列をいくつかの列ブロックに分け，最低限二つのブロックが実記憶装置に入るようにブロックの大きさを決める．

このように行列をブロックに分けると，スワッピングはブロック単位で行えるので，スワッピング回数が数十分の一に減り，経過時間を短縮することができる．このようなガウス法をたてブロックガウス法とよぶ[7]．

たてブロックガウス法では，第 k ブロックの処理を2段階に分けて行う．つまり，第 k ブロックを第1〜第 $(k-1)$ ブロックで更新することと，第 k ブロック内での更新および消去とである．一般のガウス法とたてブロックガウス法を同じ連立1次方程式で実験した結果を表 11 に示す．

行列の規模が大きくなるにつれて，たてブロックガ

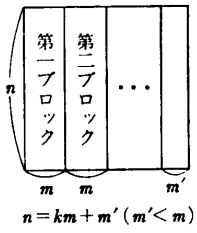


図 4 たてブロックガウス法
Fig. 4 Column-wise block Gauss elimination.

表 11 一般のガウス法とたてブロックガウス法
Table 11 Ordinary Gauss elimination and column-wise block Gauss elimination.

(秒)

次数 n	たてブロックガウス法			一般のガウス法		
	CPU時間	経過時間	比	CPU時間	経過時間	比
400	12.0	12.0	1.0	12.0	12.2	1.0
450	17.0	31.7	1.9	17.1	34.9	2.0
500	23.3	50.0	2.1	24.2	1,352.	56.
550	31.0	105.	3.4	—	—	—
600	40.3	140.	3.5	—	—	—
650	51.2	179.	3.5	—	—	—
700	63.9	220.	3.4	—	—	—

ウス法が経過時間で優位になり，行列が実記憶装置に入るときは両者は等しい．さらに，たてブロックガウス法において，仮想記憶装置を使ってそのやり取りをオペレーティング・システムに任せるのではなくて，2次記憶装置を使ってFORTRANの非同期入出力機能を用いればスワッピングに要する時間を少なくできる．ただし，この場合，配列は1次元方式でなければならない．拡張記憶装置付きのスーパコンピュータの場合は，データ参照・転送時間が速いので非同期入出力は不要となる．

7. ベクトルプロセッサにおける特性解析

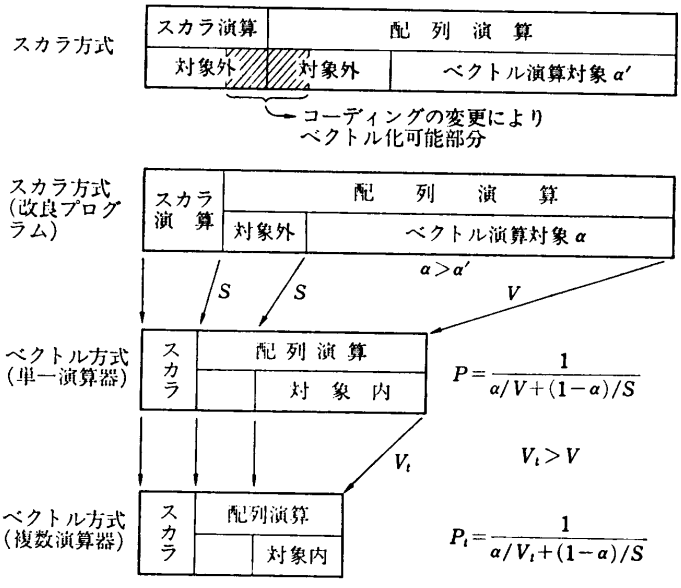
ベクトル演算器を複数個もつスーパコンピュータによる性能向上は二つの特性によって決まる．一つは従来のベクトルプロセッサの場合に考えられてきた特性で，FORTRANプログラムのベクトル比率である．ベクトル比率を上げるには，ベクトルの長さを増やすか，スカラをベクトル化してやる必要が生じる．大形の構造解析プログラムでは，従来帯外の非ゼロ要素を active column 方式によって例外処理したり，1列中の非ゼロ要素をまとめていくつかのブロックに分けて処理するなど，スカラマシン向きのアルゴリズムを採用していたが，スーパコンピュータではベクトル長を増すためにスカイライン法 (envelope 法ともいう) を採用することが必要となる．このとき2列間の内積計算量は増加するが，この計算がベクトルプロセッサの上で行われるので，処理ははるかに速くなる．しかし，一般的には主記憶装置は増える．最近のように百万バイト単位の主記憶容量が実装される場合には，このことは大した欠点にはならない．FORTRAN プログラムのベクトル化率と，ベクトルプロセッサでの効果を図5に示す．

従来のスカラマシンに対し，いま扱っているスーパコンピュータのスカラ演算の性能向上比を S，ベクトル演算の性能向上比を V とする．ベクトル演算を行うには，DO ループの前処理と同じく負荷が必要となり，V はベクトル長に無関係な定数とはならないが，ここでは一定とみなして話を進める．FORTRAN プログラムのベクトル演算対象比率（つまりベクトル化率）を α とすれば，スカラマシンにくらべたスーパコンピュータの性能向上比は

$$P=\frac{1}{\alpha/V+(1-\alpha)/S}$$

となる．既存の FORTRAN プログラムでは一般の α は大きくないので，すでに述べたように，アルゴリズムを変えたり，コーディングを修正して α を大きくする必要がある．

性能向上のための第2の特性は並列処理可能な演算器の個数である．演算器の個数 t を増やせば，ベクト



(改良プログラムの例) 枢軸選択によるガウス法の行交換

```
DO 200 J=1,N                    DO 210 J=1,N
AX=A(K,J)                   210 T(J)=A(K,J)
A(K,J)=A(IROW,J)      ⇨         DO 200 J=1,N
200 A(IROW,J)=AX                A(K,J)=A(IROW,J)
                            200 A(IROW,J)=T(J)
```

図 5 ベクトルプロセッサの性能向上
Fig. 5 Improved performance with a vector processor.

表 12　ベクトル化率と性能向上比 P_t
Table 12 Ratio of vectorization and ratio of improved performance P_t.

ベクトル化率 α	P_4			P_2	P_1
	S=1.0	S=1.5	比率	S=1.0	S=1.0
0.3	1.42	2.12	1.49	1.40	1.38
0.4	1.64	2.45	1.49	1.63	1.59
0.5	1.96	2.92	1.48	1.93	1.87
0.6	2.43	3.61	1.48	2.38	2.27
0.7	3.20	4.72	1.47	3.09	2.88
0.8	4.68	6.81	1.45	4.41	3.94
0.85	6.09	8.75	1.43	5.60	4.83
0.9	8.69	12.24	1.40	7.69	6.25
0.925	11.05	15.28	1.38	9.44	7.31
0.95	15.19	20.33	1.33	12.24	8.82
0.975	24.24	30.38	1.25	17.39	11.11
1.0	60.00	60.00	1.00	30.00	15.00
演算器数	4	4	—	2	1

ル計算が並列となりベクトル演算時間がほぼ $1/t$ に減る．演算器の個数 t のときの性能向上比 P を P_t と書けば，ベクトル化率 α と性能向上比 P_t の関係は**表 12** と**図 6** のとおりになる．

表 12 から，ベクトル化率 α が 0.9 を越えないと演算器を並行させても大した効果のないことがわかる．p 個の演算器をもつスーパコンピュータで，m 個の並列演算を行うときを考える ($p \geqq m$)．いま，次のような m 個の要素からなる内積計算を実行しよう．

$$a_i = b_i^{(1)} \times c_i^{(1)} + b_i^{(2)} \times c_i^{(2)} + \cdots + b_i^{(m)} \times c_i^{(m)} \quad (i = 1, \cdots, n)$$

この並列計算をスカラプロセッサとベクトルプロセッサで実行したときの図を図 3 に示してある．$p < m$ のときは

$$a_i = (b_i^{(1)} \times c_i^{(1)} + \cdots + b_i^{(p)} \times c_i^{(p)}) + \cdots$$

というように p 個ずつ区切って順番に計算する．

並列演算器が 4 個のスーパコンピュータで LU 分解するときを考える．ガウス法で 1 回の消去計算において枢軸列を 2 列，被消去列を 2 列選んだアルゴリズムを採用すれば性能はもっとよくなる．枢軸列を列 r と列 $r+1$，被消去列を列 j と列 $j+1$ にすれば，1 対

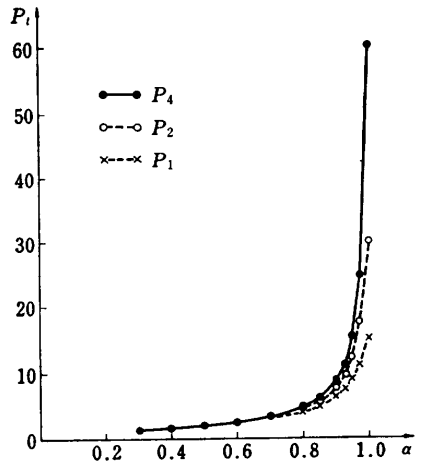


図 6　ベクトル化率と性能向上比 P_t
Fig. 6 Ratio of vectorization and ratio of improved performance.

の消去計算は次式により行われる．

$$\begin{cases} a_{rj}^{(r)} = -a_{rj}^{(r-1)} / a_{rr}^{(r-1)} = t_j^{(r)} \\ a_{r+1\,j}^{(r)} = -a_{r+1\,j}^{(r)} / a_{r+1\,r+1}^{(r)} = t_j^{(r+1)} \\ a_{ij}^{(r+1)} = a_{ij}^{(r-1)} + t_j^{(r)} \cdot a_{ir}^{(r-1)} + t_j^{(r+1)} \cdot a_{ir+1}^{(r)} \\ a_{rj+1}^{(r)} = a_{rj+1}^{(r-1)} / a_{rr}^{(r-1)} = t_{j+1}^{(r)} \\ a_{r+1\,j+1}^{(r+1)} = a_{r+1\,j+1}^{(r)} / a_{r+1\,r+1}^{(r)} = t_{j+1}^{(r+1)} \\ a_{ij+1}^{(r+1)} = a_{ij+1}^{(r-1)} + t_{j+1}^{(r)} \cdot a_{ir}^{(r-1)} + t_{j+1}^{(r+1)} \cdot a_{ir+1}^{(r)} \end{cases} \quad (2), (3)$$

つまり，主要な DO ループ内の処理は式(2)と(3)になる．この両式の中に現われる $t_j^{(r)}, t_j^{(r+1)}, t_{j+1}^{(r)}$，および $t_{j+1}^{(r+1)}$ は j ないし $j+1$ を固定すると，スカラとみなせる．したがって，式(2)と(3)は

$$\begin{cases} \boldsymbol{A} = \boldsymbol{A} + s \times \boldsymbol{C} + t \times \boldsymbol{D} & (2)' \\ \boldsymbol{B} = \boldsymbol{B} + u \times \boldsymbol{C} + v \times \boldsymbol{D} & (3)' \end{cases}$$

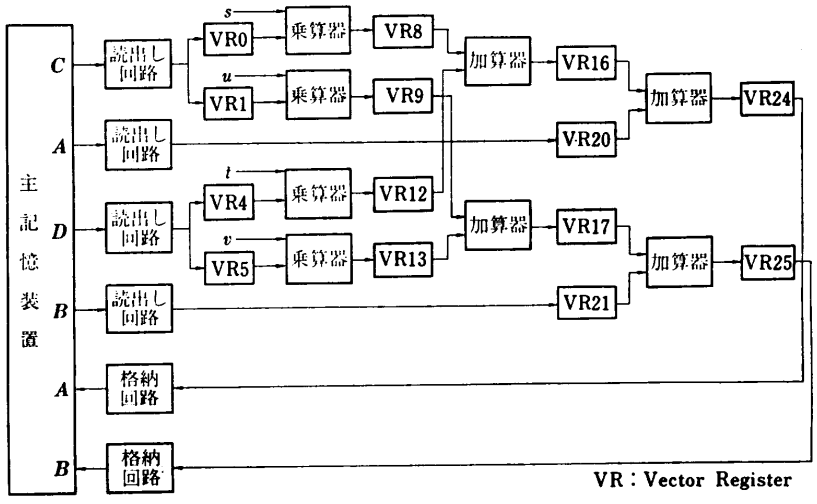


図 7　2 列同時 2 列消去 ($\boldsymbol{A} = \boldsymbol{A} + s \times \boldsymbol{C} + t \times \boldsymbol{D}$, $\boldsymbol{B} = \boldsymbol{B} + u \times \boldsymbol{C} + v \times \boldsymbol{D}$)
Fig. 7 Two columns elimination with two pivotal columns.

表 13 同時2列消去による効果 (単位：秒)
Table 13 Effect with simultaneous elimination by two pivotal columns.

アルゴリズム	VP 化	次数 n = 312 CPU	比	400 CPU	比	500 CPU	比	600 CPU	比
(1)標準	有	0.32		0.56		0.91		1.43	
	無	4.16	13.0	8.84	15.7	17.2	18.9	29.8	20.8
(2)ベクトル向	有	0.19		0.35		0.59		0.96	
	無	4.22	22.2	8.93	25.5	17.4	29.4	30.0	31.2
(3)ベクトル向2列同時	有	0.13		0.23		0.38		0.62	
	無	3.41	26.2	7.23	31.4	14.1	37.1	24.2	39.0
(4)ベクトル向2列同時2列	有	0.10		0.20		0.32		0.52	
	無	2.90	29.0	6.12	30.6	11.9	37.1	20.4	39.2

CPU：CPU 時間

と書ける．ここで，太字はベクトルを示す．式(2)′と(3)′の計算と演算器およびベクトルレジスタとの関係を示すと，**図7**のようになる．

つまり，主記憶装置からの読出し回路と乗算器がおのおの4個，加算器と格納回路がおのおの2個あれば，計算はすべて並行して実行できる．読出し回路や並列処理できる演算器がもっと多いハードウェアならばさらに性能を向上できるが，プログラムはそれだけ面倒になる．この2列同時2列消去はスカラプロセッサの場合もDOループの回数を1/4に減らすのできわめて有効になる．S-810のスカラプロセッサおよびベクトルプロセッサで各種のガウス法を実行したときの結果を**表13**に示す．S-810はハードウェアで自動的に並列演算を行う機能をもっているが，ここでは並列処理の性能を比較するために各演算器を独立に動作させて測定を行った．

この表で，アルゴリズムの違いは次のとおりである．

(1) ベクトルプロセッサを意識しない標準のアルゴリズム

(2) ベクトルプロセッサを意識したアルゴリズムで，(3)と(4)の基礎となるアルゴリズム

(3) a_{ij} 要素を2列の枢軸列で同時に消去するアルゴリズム

(4) a_{ij}, a_{ij+1} 要素を2列の枢軸列で同時に消去するアルゴリズム

8. むすび

本論文では，大規模な連立1次方程式を直接法で解くためのコンピュータプログラムを作る上で，ハードウェアやFORTRANコンパイラの諸特性をどう考慮すべきかを示した．これらの諸特性を活かした連立1次方程式用ルーチンはすでに数値計算副プログラム集MSLⅡおよびS-810用行列演算副プログラム集MATRIX/HAPとして一部実用化済みである．

本論文では直接解法を扱ったが，スーパコンピュータの出現により可能となりつつある数値流体力学関係ではICCG法をはじめとする反復法が重要視されている．これらの反復法についての評価も現在続行中であり，まとまり次第発表したいと考えている．

謝辞 本論文のテーマ全般について長期にわたってご指導をいただいた図書館情報大学村田健郎教授にお礼申し上げる．

参考文献

1) Brayton, R. K. et al.: Some Results on Sparse Matrices, *Math. Comput.*, Vol. 24, No. 112, pp. 937-954 (1970).
2) Bunch, J. R.: Analysis of Sparse Elimination, *SIAM*, Vol. 11, No. 5, pp. 847-873 (1981).
3) Faddev, D. K. et al.: *Computational Methods of Linear Algebra*, Moscow (1963).
4) George, A. et al.: *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs (1981).
5) Harano, S.: Improvements in Sparse Matrix Operations of NASTRAN, 9th NASTRAN User's colloquim, pp. 14-48 (1980).
6) McCormick, C. W.: *Sparse Matrix Operations in NASTRAN, Theory and Practice in Finite Element Structural Analysis*, pp. 611-631, 東京大学出版会, 東京 (1973).
7) 村田健郎：仮想メモリ方式と FORTRAN 最適化機能に適した大次元行列演算プログラム技法，情報処理研修センター，東京 (1979).
8) Orchard-Hays, W.: *Advanced Linear-Programming Computing Techniques*, McGraw-Hill, New York (1968).
9) Tewarson, R. P.: *Sparse Matrices*, Academic Press, New York (1973).
10) Westlake, J. R.: *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations*, John Wiley & Sons, New York (1972).

(昭和59年2月9日受付)
(昭和59年3月6日採録)