応用数理工学特論
第 3 回授業ノート

廣田悠輔, 程飛

2008 年 5 月 1 日

目次

1 **分散メモリ型並列計算機のプログラミング** 3

1.1 通信ライブラリ MPI 3

1.2 MPI のプログラミングモデル 3

1.3 MPI の関数 4

1.4 行列乗算の例 4

2 **高性能化の技法** 5

2.1 基本的な考え方 5

2.2 実行時間の予測 6

2.3 行列乗算の例 7

2.4 通信の隠蔽 8

3 **Fox のアルゴリズム** 8

3.1 アルゴリズム 8

3.2 Fox のアルゴリズムの MPI による実装 10

3.3 Fox のアルゴリズムの性能モデル 11

3.4 より効率的な行列乗算 11

4 **連立一次方程式の高性能解法（密行列の場合）** 11

4.1 LU 分解 11

4.2 LU 分解を用いた連立一次方程式の求解 12

4.3 ガウスの消去法よる LU 分解 12

1　分散メモリ型並列計算機のプログラミング

1.1　通信ライブラリ MPI

通信ライブラリ MPI

MPI（Message Passing Interface）とは並列コンピューティング利用するための標準化された規格である．実装自体を指すこともある．複数の CPU が情報をバイト列からなるメッセージとして送受信することで協調動作を行えるようにする．

MPI の機能

まず，1 対 1 通信が可能である．2 つのプロセス間での通信を行う．1 対 1 通信サブルーチンは，本当に 2 つのプロセス間のみで通信を行なうために使用する場合の他に，集団通信サブルーチンの機能を補う目的で，集団通信サブルーチンの代用として使用する場合がある．そして，分散した処理結果をまとめるためのブロードキャスト機能がある．まとめたデータの総和演算・最大値のような演算機能も完備されている．他にも実用的な機能を備えている．

オペレータ	操作
MPI_MAX	最大値
MPI_MIN	最小値
MPI_SUM	和
MPI_PROD	積
MPI_LOR	論理和
MPI_BOR	ビット論理和
MPI_LAND	論理積
MPI_BAND	ビット論理積
MPI_LXOR	排他的論理和
MPI_BXOR	ビット排他的論理和
MPI_MAXLOC	最大値とその位置
MPI_MINLOC	最小値とその位置

1.2　MPI のプログラミングモデル

SPMD（Single Program Multiple Data）

SPMD とは並列プログラミングの一つのスタイルで，シンプルかつ汎用的なので広く用いられている．SPMD は Single Program Multiple Data の略である．その意味するところは，複数のプロセッサに個々別々のデータが乗っているが，それを処理するプログラムはすべて同一であるということである．しかし，プログラムが同一だからといって処理内容が同一であるとは限らない．条件分岐を用いれば，プロセッサ毎に全く異なる処理を行うプログラムを SPMD の枠内で記述することは難しくないことは明らかである．各プロセッサは固有の番号 0, 1, …, $p-1$ を持ち，プログラム中で自分の番号を参照して処理を行う．

分散メモリ

分散メモリとは各プロセッサは自分の持つローカルメモリのみをアクセス可能なモデルである．他プロセッ

サのメモリ上にあるデータが必要な場合は，プロセッサ間のネットワークに依存することとなるので，演算性能が低下してしまう．

1.3 MPI の関数

起動／終了	
MPI_INIT	MPI の初期化
MPI_FINALIZE	MPI の終了
環境情報の取得	
MPI_COMM_SIZE	全プロセッサ台数の取得
MPI_COMM_RANK	プロセッサ番号の取得
1 対 1 の送受信	
MPI_SEND	データの送信
MPI_RECV	データの受信
集団通信	
MPI_BCAST	データのブロードキャスト
MPI_REDUCE	リダクション演算（総和／最大値など）

1.4 行列乗算の例

問題

$N \times N$ 行列 A, B に対し，P 台のプロセッサを使って $C = AB$ を計算し，A はブロック行分割，B はブロック列分割されているとする．結果の C はブロック列分割の形で求めるとする．(図 1)

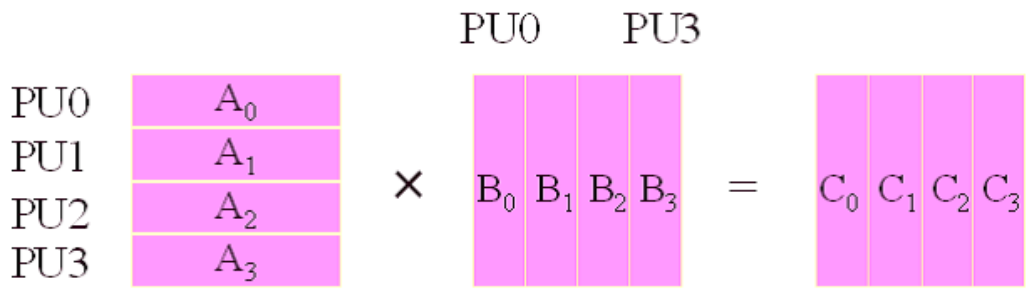

図 1　ブロック列分割の形

各プロセッサは，行列の一部のみを持つ（分散メモリ）．また，自分の持つ部分のみにアクセスできる．計算の方針は各 C_J を縦方向に P 個のブロック $C_{0J}, C_{1J}, \cdots, C_{P-1,J}$ に分ける．まず，自分の持つ部分行列のみで計算できる C_{JJ} を計算する．その後，他のプロセッサからデータをもらいながら，他の C_{IJ} を計算す

る．(図 2)

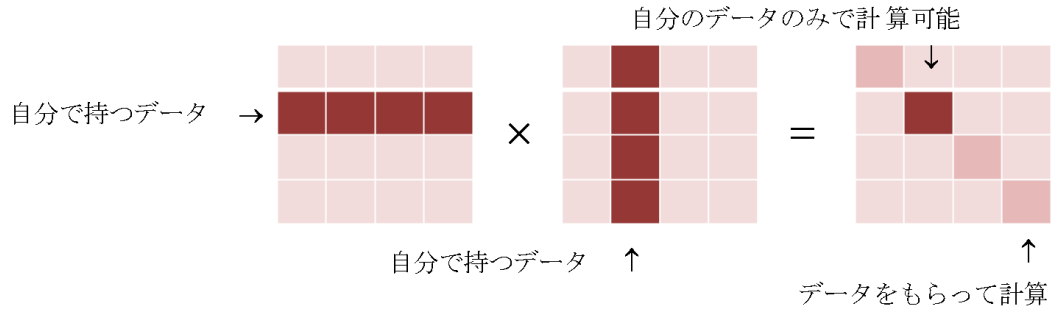

図 2　計算の仕組み

program matmult
(配列の確保：A_J, B_J, C_J, ATMP)
(MPI の初期化と環境情報取得．自分のプロセッサ番号を J とする．)
(配列 A_J, B_J の初期化)
$C_{JJ} = A_J \times B_J$
do $I = 1, P-1$
　(プロセッサ MOD(J-I+P,P) に A_J を送る．)
　(プロセッサ MOD(J+I,P) から $A_{\mathrm{MOD}\ (J+I,P)}$ を受け取り，ATMP に格納．)
　$C_{\mathrm{MOD}\ (J+I,P)}$ ATMP $\times B_J$
end do
(配列 C_J の出力)
(MPI の終了)
stop
End

2　高性能化の技法

線形計算において，一般的な計算方法でなく他のプロセッサからデータを調達する必要があり，分散メモリ型並列計算機のアーキテクチャに合わず，計算性能が大幅に低下する．高性能化のために，様々な工夫が必要となる．

2.1　基本的な考え方

PU 間の負荷分散均等化

各 PU の処理量が均等になるよう処理を分割する．均等に分割できないと，一部のプロセッサが次のステップに入る．(図 3)

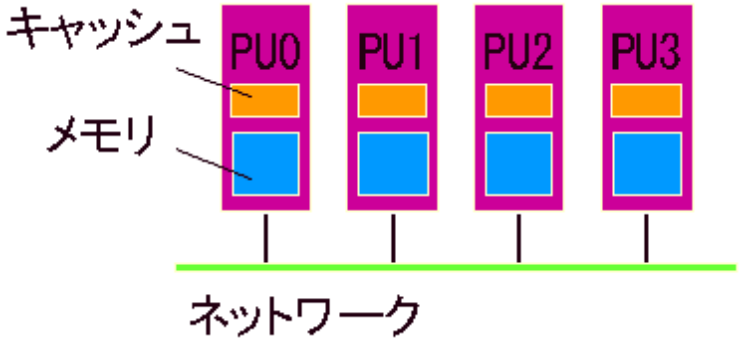

図 3　分散メモリ型並列計算機の通信

PU 間通信量の削減

通信には，データ 1 個あたり数十サイクルの時間が必要で，データ分割の最適化による通信量の削減と通信の隠蔽が重要である．

全データを適切な形で分割できれば，計算と通信が同時に進行できることで通信時間の隠蔽はできる．

PU 間通信回数の削減

1 回の通信には通常，数百〜数千サイクルの起動時間が必要で，なるべく通信の回数を減らすことは効果的である．1 回あたりの並列計算規模を拡大し，並列粒度の増大による通信回数の削減が重要である．(図 4)

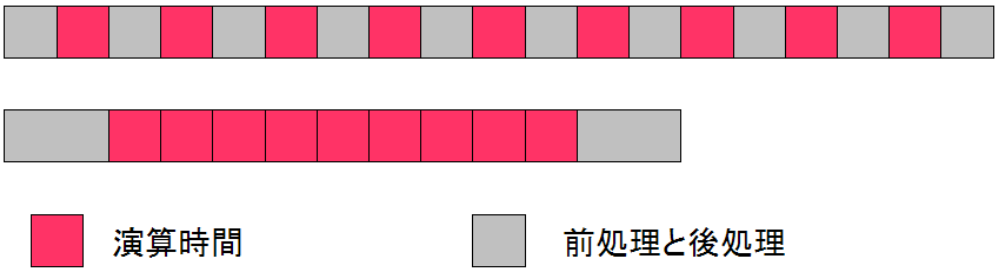

図 4　演算と通信のイメージ

キャッシュメモリの有効利用

ほかには，コンピュータの特徴により，メインメモリと比べてキャッシュメモリがかなりの速度でデータの処理ができ，有効利用が重要な手段である．(図 5)

2.2　実行時間の予測

演算時間

T_{comp} ＝ 演算量／演算速度

但し，以上の計算式は単純なモデルでより精密な計算キャッシュの影響などを考慮する必要ある．

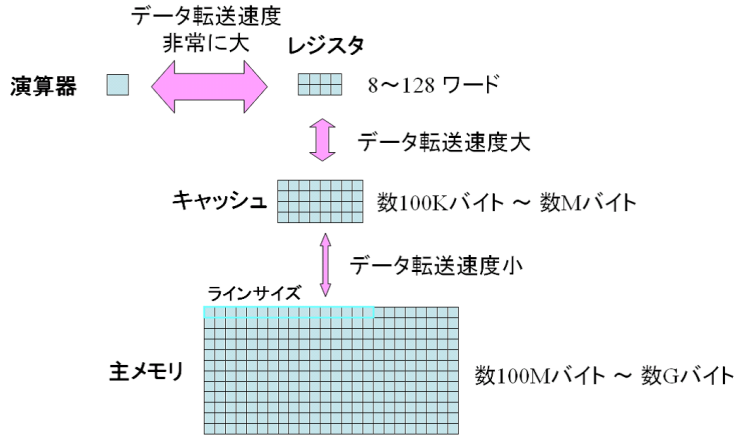

図 5　レジスタとキャッシュ，メモリの関係

通信時間

T_{comm} ＝ 転送回数×起動時間 ＋ 転送量／転送速度

待ち時間

T_{idle}

全実行時間

$T_{\mathrm{exec}} = T_{\mathrm{comp}} + T_{\mathrm{comm}} + T_{\mathrm{idle}}$

2.3　行列乗算の例

アルゴリズム

A，C をブロック行分割，B をブロック列分割とした行列乗算である．(図 6)

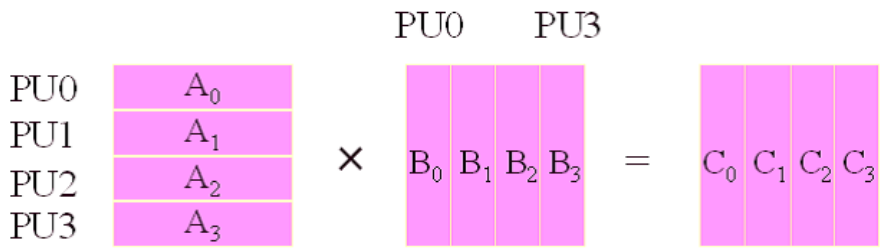

図 6　ブロック列分割の形

実行時間の予測

$T_{\mathrm{comp}} = (2N^3/P)/s$　　：P に反比例して減少

$T_{\mathrm{comm}} = (P-1) \times (T_{\mathrm{setup}} + (N/P) \times N \times 8/b)$

$= (P-1) \times T_{\mathrm{setup}} + 8(1-1/P)N^2/b)$　　：P とともに減少しない

$T_{\mathrm{idle}} = 0$ ：負荷は完全に均等

以上の式より，プロセッサ数 P を増やすと，並列化効率が大きく低下する可能性が出てくる．

2.4 通信の隠蔽

行列 A のデータを，計算に必要な 1 ステップ前に送ってもらう．演算とのオーバーラップにより，通信時間を隠蔽できる．

program matmult
（配列の確保： A_J, B_J, C_J,ATMP）
（MPI の初期化と環境情報取得．自分のプロセッサ番号を J とする．）
（配列 A_J, B_J の初期化）
（プロセッサ MOD(J-1+P,P) に A_J を送る．）
$C_{JJ} = A_J \times B_J$
do $I = 1, P-1$
（プロセッサ MOD(J+I,P) から $A_{\mathrm{MOD}\ (J+I,P)}$ を受け取り，ATMP に格納．）
IF $(I < P-1)$（プロセッサ MOD(J-I+1+P,P) に A_J を送る．）
$C_{\mathrm{MOD}\ (J+I,P),J} = \mathrm{ATMP} \times B_J$
end do
（配列 C_J の出力）
（MPI の終了）
stop
end

但し，演算時間が通信時間より短いと完全な隠蔽が不可能である．

3 Fox のアルゴリズム

3.1 アルゴリズム

Fox のアルゴリズムとはプロセッサを 2 次元に並べ，A, B, C を縦横両方向にブロック分割する方法である．

プロセッサ I, J は，第 K ステップで

$$C_{IJ} += A_{I\ \mathrm{MOD}\ (I+K-1,\sqrt{P})}\ B_{\mathrm{MOD}\ (I+K-1,\sqrt{P}),J}$$ で計算できる．(図 7，8)

program matmult
（配列の確保： A_{IJ}, B_{IJ}, C_{IJ},ATMP, BTMP）
（MPI の初期化と環境情報取得．自分のプロセッサ番号を I, J とする．）
（配列 A_{IJ}, B_{IJ} の初期化）

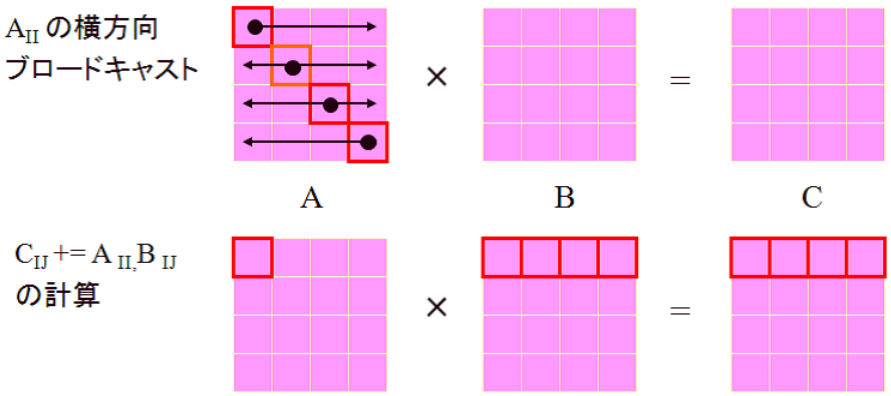

図 7　第 1 ステップ

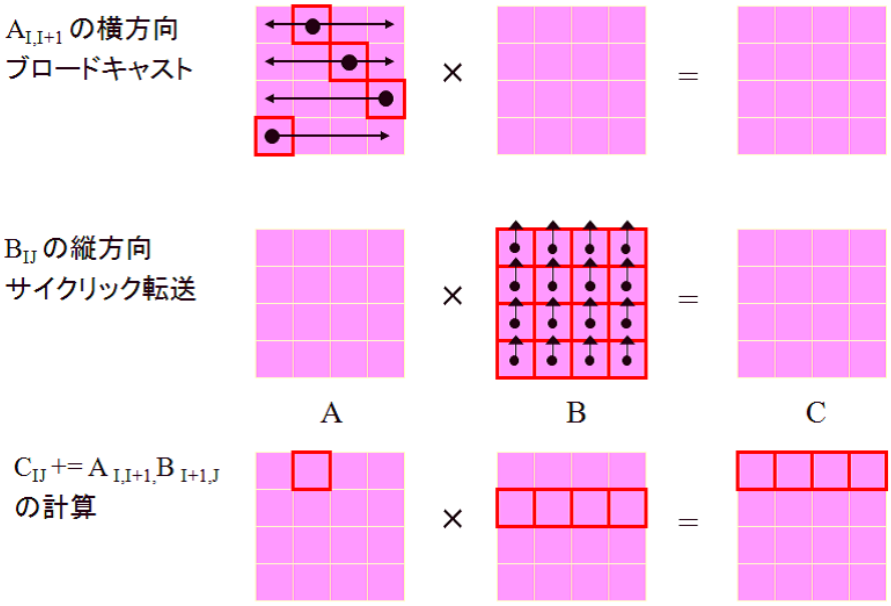

図 8　第 2 ステップ

do $K = 1, \sqrt{P}$

IF $(J = \mathrm{MOD}(I + K, \sqrt{P}))$（配列 A_{IJ} を ATMP にコピー）

配列 ATMP を行方向のプロセッサ間でブロードキャスト

IF $(K > 1)$ THEN

（プロセッサ $(\mathrm{MOD}(I - 1 + P, P), J)$ に BTMP を送る.）

（プロセッサ $(\mathrm{MOD}(I + 1, P)$ から BTMP を受け取る.）

ELSE

（B_{IJ} を配列 BTMP にコピー）

END IF
　C_{IJ} += ATMP × BTMP
end do
(配列 C_{IJ} の出力)
(MPI の終了)
stop
end

3.2　Fox のアルゴリズムの MPI による実装

同じ I の値を持つ PU 群に対してコミュニケータ（PU のグループ）を定義コミュニケータを使用する．
横方向ブロードキャストとは `MPI_BCAST` によるブロードキャストである．(図 9)
縦方向サイクリック転送とは `MPI_SEND` を用いて 1 つ上の PU にデータ送信である．(図 10)

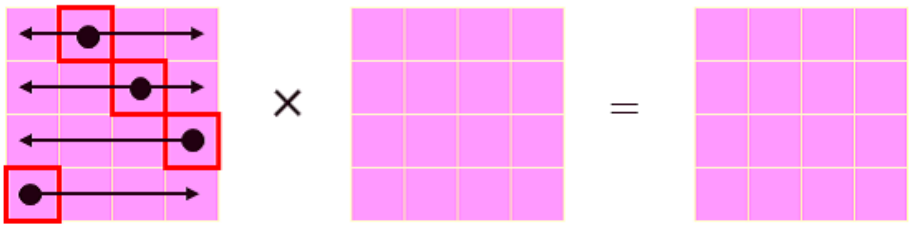

図 9　横方向ブロードキャスト

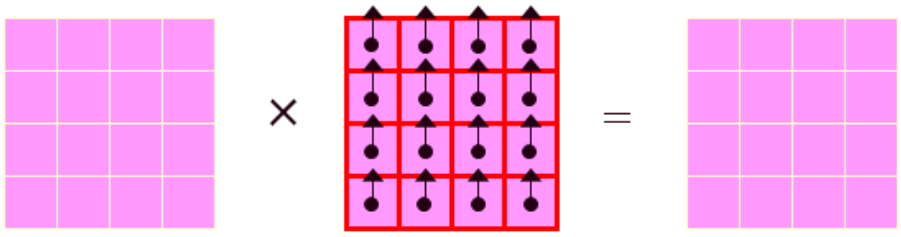

図 10　BTMP の縦方向サイクリック転送

3.3 Fox のアルゴリズムの性能モデル

実行時間の予測

$T_{\mathrm{comp}} = 2N^3/P$ ：P に反比例して減少

$T_{\mathrm{comm}} = \sqrt{P} \times \log_2(\sqrt{P}) \times (T_{\mathrm{setup}} + N^2/P \times 8/b)$ ：ATMP の転送

$\quad +(\sqrt{P}-1) \times (T_{\mathrm{setup}} + N^2/P \times 8/b)$ ：BTMP の転送

$T_{\mathrm{idle}} = 0$ ：負荷は完全に均等

P を増やすと通信時間も $1/\sqrt{P}$ で減少し，P を増やしたときの並列化効率の低下が小さい．

3.4 より効率的な行列乗算

通信の隠蔽

ATMP の転送を，計算の 1 ステップ前に行うが，通信用バッファがもう 1 個必要となる．ノンブロッキング通信（`MPI_ISEND`，`MPI_IRECV`）を使うと，より効果的である．

より効率的なアルゴリズムの利用

SUMMA (Scalable Universal Matrix Multiplication),LAPACK Working Notes 96 のような完成したアルゴリズムの利用も計算速度を上げる手段の 1 つである．

4 連立一次方程式の高性能解法（密行列の場合）

n 個の未知数 $x_1, x_2, \ldots, x_n$ をもつ連立一次方程式は以下のように表される．

$$\begin{cases} a_{1,1}\,x_1 + a_{1,2}\,x_2 + \cdots + a_{1,n}\,x_n = b_1 \\ a_{2,1}\,x_1 + a_{2,2}\,x_2 + \cdots + a_{2,n}\,x_n = b_2 \\ \qquad\qquad\qquad \vdots \\ a_{n,1}\,x_1 + a_{n,2}\,x_2 + \cdots + a_{n,n}\,x_n = b_n \end{cases}$$

係数行列を A，未知ベクトルを $\boldsymbol{x}$，既知ベクトルを $\boldsymbol{b}$ とおくと，この連立一次方程式は次のように表すことができる．

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

以下では A は正則であると仮定する．本節では，(1) の形式で表される連立一次方程式の解法で，特に A が密行列である場合に向く，LU 分解による連立一次方程式の解法について説明を行う．

4.1 LU 分解

LU 分解とは，行列 A を $A = LU$ となるように対角要素がすべて 1 である下三角行列

$$L = \begin{bmatrix} 1 & & & \\ l_{2,1} & 1 & & \\ \vdots & \ddots & \ddots & \\ l_{n,1} & \cdots & l_{n,n-1} & 1 \end{bmatrix}$$

および上三角行列

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ & u_{2,2} & \cdots & u_{2,n} \\ & & \ddots & \vdots \\ \multicolumn{2}{c}{\smash{\raisebox{.5em}{0}}} & & u_{n,n} \end{bmatrix}$$

に分解することである．このように LU 分解する場合，L および U は一意に決定される．行列 A が正則であっても A は一般に LU 分解可能な行列ではないが，以下では LU 分解を行う対象の行列は LU 分解可能であると仮定する．

4.2 LU 分解を用いた連立一次方程式の求解

連立一次方程式 (1) の係数行列 A を LU 分解することにより，容易に (1) を解くことが可能である．(1) は，

$$LU\boldsymbol{x} = \boldsymbol{b}$$

と書くことができるから，まず，

$$L\boldsymbol{y} = \boldsymbol{b} \tag{2}$$

の解 $\boldsymbol{y}$ を求め，$\boldsymbol{y}$ を既知ベクトルとする連立一次方程式

$$U\boldsymbol{x} = \boldsymbol{y} \tag{3}$$

を解くことで (1) の解が得られる．(2)，(3) の係数行列はそれぞれ下三角行列，上三角行列になっているので，(2) は前進代入によって，(3) は後退代入によって少ない演算量で解くことが可能である．したがって，A が LU 分解されていれば，(1) の解を少ない演算量で計算することが可能である．

4.3 ガウスの消去法よる LU 分解

ガウスの消去法によって連立一次方程式 (1) を解くとき，第 k ステップで，第 $k-1$ ステップまでの操作によって変形された行列を $A^{(k)}$ の，第 k 行の $a_{i,k}^{(k)}/a_{k,k}^{(k)}$ 倍を第 i 行から引くという操作を繰り返し行う ($k = 1, 2, \ldots, n-1$)．この操作によって，A を，対角要素を含まない行列の下三角要素を 0 にし，上三角行列 U に変形する．第 k ステップで，$A^{(k)}$ の第 k 行の $a_{i,k}^{(k)}/a_{k,k}^{(k)}$ 倍を第 i 行から引くという操作は，

$$L_k = \begin{bmatrix} 1 & & & & & & & \\ 0 & 1 & & & & & \multicolumn{2}{c}{\smash{\raisebox{.5em}{0}}} \\ \vdots & \ddots & \ddots & & & & & \\ \vdots & & 0 & 1 & & & & \\ \vdots & & 0 & -a_{k+1,k}^{(k)}/a_{k,k}^{(k)} & 1 & & & \\ \vdots & & 0 & -a_{k+2,k}^{(k)}/a_{k,k}^{(k)} & 0 & 1 & & \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & -a_{n,k}^{(k)}/a_{k,k}^{(k)} & 0 & \cdots & 0 & 1 \end{bmatrix}$$

を $A^{(k-1)}$ の左からかけていることに相当する．第 1 ステップから第 $n-1$ ステップまでこの操作を繰り返した結果，A が U に変形されているので，

$$L_{n-1}L_{n-2}\cdots L_1 A = U$$

と書ける．両辺に $L = (L_{n-1}L_{n-2}\cdots L_1)^{-1}$ を左からかけると

$$A = LU$$

となる．対角要素がすべて 1 の下三角行列同士の行列積は対角要素がすべて 1 の下三角行列であり，対角要素がすべて 1 の下三角行列の逆行列は対角要素がすべて 1 の下三角行列であるので，L も対角要素がすべて 1 の下三角行列である．したがって，ガウスの消去法の前身消去過程の A の変形は，A を LU 分解して，対角要素がすべて 1 の下三角行列 L の逆行列を A を計算し，上三角行列 U が得ていること相当している．

上述したガウスの消去法による LU 分解では，第 k 行の $a_{i,k}^{(k)}/a_{k,k}^{(k)}$ 倍を第 i 行から引くという操作が繰り返される．これを数式で表すと，

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} a_{k,j}^{(k)} \quad (k+1 \le i \le n)$$

となる．上式のベクトルの演算をまとめて表すと以下のように書ける．

$$A_k^{(k+1)} = A_k^{(k)} - \frac{1}{a_{k,k}^{(k)}} \boldsymbol{b}_k \boldsymbol{c}_k^{\mathrm{T}}$$

但し，$\boldsymbol{b}_k = [a_{k+1,k}^{(k)}, a_{k+2,k}^{(k)}, \ldots, a_{n,k}^{(k)}]^{\mathrm{T}}$，$\boldsymbol{c}_k = [a_{k,k+1}^{(k)}, a_{k,k+2}^{(k)}, \ldots, a_{k,n}^{(k)}]^{\mathrm{T}}$ であり，図 11 に示す部分の要素をもつ行列およびベクトルになっている．$\tilde{\boldsymbol{b}}_k = (1/a_{k,k}^{(k)})\, \boldsymbol{b}_k$ とおけば，

$$A_k^{(k+1)} = A_k^{(k)} - \tilde{\boldsymbol{b}}_k \boldsymbol{c}_k^{\mathrm{T}}$$

と書ける．これは行列の rank-1 更新である．行列の rank-1 更新は Level-2 BLAS の演算であり，データの再利用性が低くキャッシュ効率が悪い．そのため，上述の LU 分解アルゴリズムは，キャッシュマシン上で高性能が期待できない．

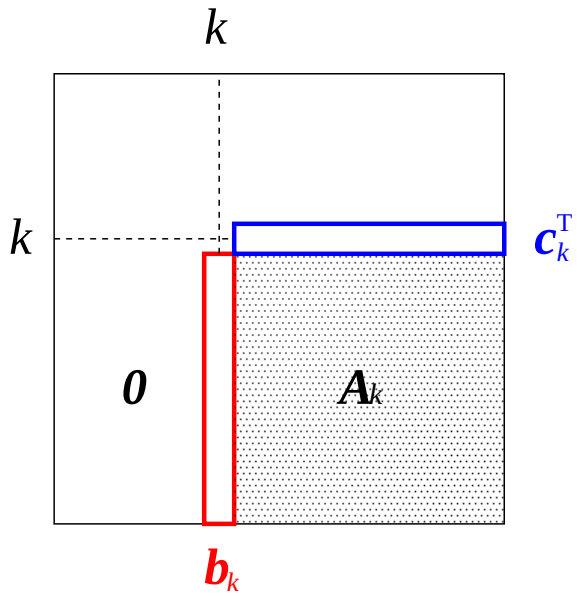

図 11　ガウスの消去法の k ステップ目の変形の計算に使用される行列とベクトル.