第 2 章

スーパーコンピュータによる数値解析

本章では，線形方程式に対する代表的な数値解法について説明し，次いでスーパーコンピュータのアーキテクチャ特性について説明する．そして，スーパーコンピュータを用いて数値計算を行う際に生じる問題点と，それに対する前処理の役割について説明する．また，本論文では前処理を二つの観点から考えた．

2.1 準備

まず最初に，本章の説明中に適宜取り上げる例題について説明する．

2 次元正方領域 $\Omega = [0,1] \times [0,1]$ におけるポアソン方程式の全周ディリクレ問題

$$-\Delta u = f \quad \text{in } \Omega, \tag{2.1}$$

$$u = g \quad \text{on } \partial\Omega \tag{2.2}$$

を考える．領域 Ω を x, y 方向にそれぞれ $(n+1)$ 等分した離散格子を作る．きざみ幅 h は $h = 1/(n+1)$ である．この離散格子において，$x = ih, y = jh$ の格子点 (i, j) における u の値 $u(ih, jh)$ を u_{ij} と表す．境界上の値は式 (2.2) の境界条件で与えられているので，Ω の内部の格子点上の値を求めればよい．図 2.1 に $n = 3$ のときの例を示す．

方程式 (2.1) を 5 点中心差分による差分法 (FDM: finite difference method) で近似すると，格子点 (i, j) における方程式は

$$-u_{i,j-1} - u_{i-1,j} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f_{i,j} \tag{2.3}$$

という漸化式となる．式 (2.3) をすべての内部格子点について書くと，線形方程式が得られる．ベクトル $\boldsymbol{u}$ を

$$\boldsymbol{u} = (u_{11}, u_{21}, \cdots, u_{n1}, u_{12}, u_{22}, \cdots, u_{n2}, \cdots, u_{1n}, u_{2n}, \cdots, u_{nn})^T \tag{2.4}$$

とすると，線形方程式 (linear systems, simultaneous linear equations) は

$$A\boldsymbol{u} = \boldsymbol{b} \tag{2.5}$$

と表すことができる．$n = 3$ のとき，係数行列 A (coefficient matrix) と右辺項ベクトル $\boldsymbol{b}$ (RHS: right hand side vector) は式 (2.6) のとおりである．

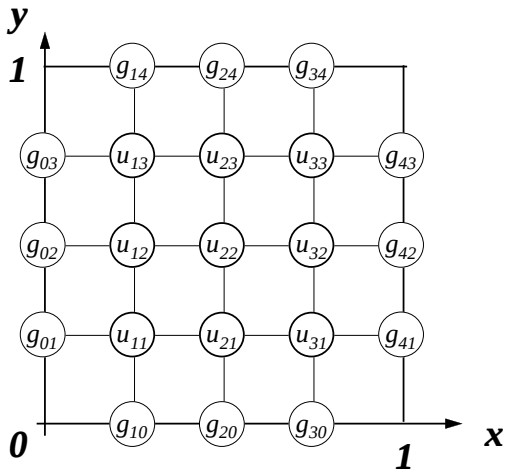

図 2.1: $n = 3$ の離散格子 (ディリクレ条件).

$$A = \begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & \mathbf{0} & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & \mathbf{0} & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} h^2 f_{11} + g_{10} + g_{01} \\ h^2 f_{21} + g_{20} \\ h^2 f_{31} + g_{30} + g_{41} \\ h^2 f_{12} + g_{02} \\ h^2 f_{22} \\ h^2 f_{32} + g_{42} \\ h^2 f_{13} + g_{03} + g_{14} \\ h^2 f_{23} + g_{24} \\ h^2 f_{33} + g_{43} + g_{34} \end{bmatrix} \tag{2.6}$$

このように，離散格子における $u(ih, jh)$ の値を表すベクトル $\boldsymbol{u}$ は，$n \times n (\equiv N)$ 項からなる列ベクトルであり，あらためて，

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_N)^T \tag{2.7}$$

として，

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2.8}$$

と表すことにする．以後，特にことわりの無い限り，係数行列 A は正則 (regular, nonsingular) な N 次正方行列，解ベクトル $\boldsymbol{x}$ (solution vector)，右辺項ベクトル $\boldsymbol{b}$ は N 次ベクトルである．したがって，図 2.1 における u の点を $\boldsymbol{x}$ で表すと，

$$\begin{array}{|ccc|} \hline u_{13} & u_{23} & u_{33} \\ u_{12} & u_{22} & u_{32} \\ u_{11} & u_{21} & u_{31} \\ \hline \end{array} \quad \rightarrow \quad \begin{array}{|ccc|} \hline x_7 & x_8 & x_9 \\ x_4 & x_5 & x_6 \\ x_1 & x_2 & x_3 \\ \hline \end{array}$$

図 2.2: $\boldsymbol{u}$ と $\boldsymbol{x}$ に対するナチュラルオーダリング.

と表される．この一般的な順序付けは，ナチュラルオーダリング (natural ordering) と呼ばれる[1] .

自然界に現れる方程式の離散化から導かれる線形方程式の係数行列 $A = (a_{ij})$ は，一般に M 行列 (M-matrix) か，あるいは適当な操作によって M 行列に帰着できる場合が多い．M 行列とは，$a_{ij} > 0, a_{ij} \leq 0, j \neq i$ かつ逆行列 A^{-1} の全ての成分が正であるような行列 A のことである [37][40][66]．M 行列のとき，様々な数値解法にとってメリットとなる定理がある [34]．一例として，M 行列ならば，ガウス消去法 (Gauss elimination) における軸選択 (pivoting) が生じない，また，反復法の前処理における不完全分解 (incomplete factorization) が可能である．ガウス消去法と不完全分解については，本章の中で説明する．

2.2　代表的な解法

線形方程式の数値解法 [38][42] は，大別して直接解法 (direct method) と反復解法 (iterative method) との二種類がある．直接解法は，係数行列 A に演算を直接ほどこしながら成分を消去してゆき，丸め誤差の無い状況では，有限回の演算で厳密解を得ることができる．反復解法は，線形方程式の解に対する近似値の列を作り出すような解法であり，一般に，アルゴリズムの繰り返しにより近似解に収束してゆき，必要な精度に達したところで打ち切る．これら解法は解くべき問題に応じて選択され，主に，直接解法は係数行列が小規模な密行列 (dense matrix) や帯行列 (band matrix) のとき反復解法は係数行列が大規模な疎行列 (sparse matrix) のときに用いられることが多い．

2.2.1　直接解法

直接解法の代表である LU 分解は，ガウス消去法に基づき係数行列 A を下三角行列 L と上三角行列 U に分解し，分解後の行列 $A = LU$ を用いて前進消去 (forward elimination)，後退代入 (backward substitution) を行い求解する方法である．

ガウス消去法は，係数行列の分解と求解を並行して行う．いま，第 k 段の変換を

M_k :　基本変換 (第 k 列において対角要素より下の要素に乗数を並べた行列),
P_k :　置換変換 (単位行列の第 i 行と第 j 行を入れ換えた行列)

[1] 辞書式オーダリング (dictionaly ordering)，行オーダリング (row ordering) などとも呼ばれる [12].

と表し，これらを用いると，

$$M_{N-1}P_{N-1}\cdots M_2P_2M_1P_1A = U, \tag{2.9}$$
$$(M_k, P_k : N \times N \text{ 行列},\ \ U : N \times N \text{ 上三角行列})$$

のように上三角行列 U に変換される．

LU 分解は，あらかじめ係数行列を分解しておいて求解する方法である．式 (2.9) に対して，

$$(M_{N-1}P_{N-1}\cdots M_2P_2M_1P_1)^{-1} = L$$

とおくと，

$$A \ = \ LU$$

の形になり，これを LU 分解と呼ぶ．ただし，L は P_k が全て単位行列のときには下三角行列となるが，一般には下三角行列ではない．

A を LU 分解しておけば，もとの方程式 (2.8) は，

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

であり，

$$L\boldsymbol{y} \ = \ \boldsymbol{b}, \tag{2.10}$$
$$U\boldsymbol{x} \ = \ \boldsymbol{y} \tag{2.11}$$

と 2 つの方程式と同値である．ここで，式(2.10)は前進代入 (forward substitution)，(2.11) は後退代入と呼ばれる．

この解法を基に作られた変種としては，A が対称正定値 (SPD: symmetric positive definite) のときのコレスキー分解法 (Cholesky factorization)，その他，A の構造に着目したアルゴリズムが数種類提案されている．3 重対角行列をはじめ，帯構造の係数行列に対しては，その特長を活かした LU 分解のアルゴリズムが適用される [24][46]．以下に，コレスキー分解とそのときの求解方法を示す．

アルゴリズム: 2.1 (コレスキー分解[2])

【分解過程】

$$\begin{aligned} A \ &= \ LL^T = (\bar{L}\mathrm{diag}(l_{ii}^{-1}))(\mathrm{diag}(l_{ii}^{-1})\bar{L}^T) \\ &= \ \bar{L}\mathrm{diag}(l_{ii}^{-2})\bar{L}^T = \bar{L}\bar{D}\bar{L}^T \end{aligned} \tag{2.12}$$

となるように分解する．$\bar{D}$ の要素を $\bar{d}_i$，$\bar{L}$ の要素を $\bar{l}_{ij}$ とすると，

$$\bar{d}_i = 1/\bar{l}_{ii}$$

であり,

$$\begin{aligned}\sum_{k=1}^{j} \bar{l}_{ik}\bar{d}_k\bar{l}_{jk} &= a_{ij} \quad \text{から,}\\ \bar{l}_{ij} &= a_{ij} - \sum_{k=1}^{j-1} \bar{l}_{ik}\bar{d}_k\bar{l}_{jk}, \qquad j = 1, 2, \cdots, i-1, \qquad (2.13)\\ \sum_{k=1}^{i} \bar{l}_{ik}\bar{d}_k\bar{l}_{ik} &= a_{ii} \quad \text{から,}\\ \bar{l}_{ii} &= a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\bar{d}_k, \quad \bar{d}_i = 1/l_{ii}, \quad i = 1, 2, \cdots, N \qquad (2.14)\end{aligned}$$

により分解できる.
【求解過程】

$$\begin{aligned}A = \bar{L}\bar{D}\bar{L}^T \rightarrow \quad & \bar{L}\bar{D}\bar{L}^T\boldsymbol{x} = \boldsymbol{b},\\ & \bar{L}\boldsymbol{y} = \boldsymbol{b}, \quad \bar{D}\bar{L}^T\boldsymbol{x} = \boldsymbol{y}\end{aligned}$$

として,

$$\begin{aligned}y_k &= b_k - \sum_{i=1}^{k-1} l_{ki}y_i, \qquad k = 1, 2, \cdots, N, \qquad (2.15)\\ x_k &= d_k^{-1}y_k - \sum_{i=k+1}^{N} l_{ik}x_i, \quad k = N, N-1, \cdots, 1 \qquad (2.16)\end{aligned}$$

と求められる.

2.2.2　反復解法

反復解法を大別すると，逐次過緩和法 (SOR 法: successive over relaxation method)[66] に代表される定常反復法 (stationary iterative methods) と共役勾配法 (CG 法: conjugate gradient method)[23] に代表される非定常反復法 (nonstationary iterative methods) とがある.

前者は解ベクトルに対して，反復過程で同一の操作を施し収束させる反復法であり，ヤコビ法 (Jacobi method)，ガウスーザイデル法 (Gauss-Seidel method) もこちらに属する.

係数行列 A を

$$A = K + R \qquad (2.17)$$

[2] 細かく分類すると，“改訂コレスキー分解” や “修正コレスキー分解” などと呼ばれる.

と分離して，元の線形方程式 (2.8) を

$$K\boldsymbol{x}_{k+1} = -R\boldsymbol{x}_k + \boldsymbol{b}$$

の形にする．ここで，k は反復回数である．この方程式を求解の形にしてまとめると，

$$\begin{aligned} \boldsymbol{x}_{k+1} &= -K^{-1}R\boldsymbol{x}_k + K^{-1}\boldsymbol{b} \\ &\equiv M\boldsymbol{x}_k + N\boldsymbol{b} \end{aligned} \tag{2.18}$$

と表される．一般に，定常反復法は式 (2.18) の形で表され，行列 M を反復行列という．この M の作り方により，以下の**アルゴリズム** 2.2∼ 2.4 が導出される．ここでは，

$$\begin{aligned} A &= D + L + U, \\ & \quad D : A \text{ の対角要素のみからなる対角行列,} \\ & \quad L : \text{狭義下三角行列,} \qquad U : \text{狭義上三角行列} \end{aligned}$$

である．

a)　$M = -D^{-1}(U + L)$ のとき

アルゴリズム: 2.2 (ヤコビ法)

$\boldsymbol{x}_0$: 初期値,
for $k = 1, 2, \cdots,$ until　$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \le \varepsilon\|\boldsymbol{b}\|$ do
begin

$$\boldsymbol{x}_{k+1} = -D^{-1}(U + L)\boldsymbol{x}_k + D^{-1}\boldsymbol{b}, \tag{2.19}$$

end

---

b)　$M = -(D + L)^{-1}U$ のとき

アルゴリズム: 2.3 (ガウスーザイデル法)

$\boldsymbol{x}_0$: 初期値,
for $k = 1, 2, \cdots,$ until　$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \le \varepsilon\|\boldsymbol{b}\|$ do
begin

$$\boldsymbol{x}_{k+1} = D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}_{k+1} - U\boldsymbol{x}_k\right), \tag{2.20}$$

end

---

c)　$M = (D + \omega L)^{-1} \{(1 - \omega)D - \omega U\}$ のとき

アルゴリズム: 2.4 (SOR 法)

$\boldsymbol{x}_0$: 初期値,　ω : 加速パラメタ,
for $k = 1, 2, \cdots,$ until　$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leq \varepsilon \|\boldsymbol{b}\|$ do
begin

$$\tilde{\boldsymbol{x}}_{k+1} = D^{-1} \left(\boldsymbol{b} - L\boldsymbol{x}_{k+1} - U\boldsymbol{x}_k \right), \tag{2.21}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \omega \left(\tilde{\boldsymbol{x}}_{k+1} - \boldsymbol{x}_k \right), \tag{2.22}$$

end

また，SOR 法による解は次式 (2.23) のようにも表される．

$$\boldsymbol{x}_{k+1} = (D + \omega L)^{-1} \{(1 - \omega)D - \omega U\} \boldsymbol{x}_k + \omega (D + \omega L)^{-1} \boldsymbol{b} \tag{2.23}$$

後者の反復解法，つまり，非定常反復法は，反復に依存した係数をもつ反復法で，共役勾配法 (CG 法) の他に，BiCG 法 [15][33]，CGS 法 [56]，BiCGSTAB 法 [65]，GPBiCG 法 [68][69] や，GCR 法 [14]，GMRES 法 [51] などはこちらに属する．これらの代表例として，共役勾配法のアルゴリズムを示す．

アルゴリズム: 2.5 (共役勾配法)

$\boldsymbol{x}_0$: 初期値,
$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,　$\boldsymbol{p}_0 = \boldsymbol{r}_0$,
for $k = 0, 1, \cdots,$ until　$\|\boldsymbol{r}_k\| \leq \varepsilon \| \boldsymbol{b} \|$
do:
begin

$$\begin{aligned}
\alpha_k &= \frac{(\boldsymbol{r}_k, \boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)}, \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k, \\
\beta_k &= \frac{(\boldsymbol{r}_{k+1}, \boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k, \boldsymbol{r}_k)}, \\
\boldsymbol{p}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k,
\end{aligned}$$

end

2.2.3　前処理付き反復解法

これら反復法の収束は，線形方程式の係数行列のスペクトル特性 (spectral properties) に依存することが知られている．多くの場合，スペクトル特性は前処理 (preconditioning) を施すことで改善され，反復法の収束性は高められる [62].

ここで，対称正定値な系に対する前処理付き共役勾配法の導出について説明する．行列 K を，$K \approx A$ かつ $K = LL^T$ (L: 下三角行列) という性質をもつように生成する．この行列 K は，一般に前処理行列 (preconditioner) と呼ばれる．このとき，解くべき線形方程式 (2.8) は,

$$(L^{-1}AL^{-T})(L^T\boldsymbol{x}) \;=\; (L^{-1}\boldsymbol{b}) \tag{2.24}$$

と変換される．ここで，**アルゴリズム** 2.5 で用いられるベクトルは，$K = LL^T$ を用いて,

$$\boldsymbol{x} \Rightarrow L^T\boldsymbol{x},\;\; \boldsymbol{p} \Rightarrow L^T\boldsymbol{p},\;\; \boldsymbol{r} \Rightarrow L^{-1}\boldsymbol{r} \tag{2.25}$$

と書き換えられることから，次の前処理付き共役勾配法のアルゴリズムが得られる．

アルゴリズム: 2.6 (前処理付き共役勾配法)

$\boldsymbol{x}_0$: 初期値,
$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\;\; \boldsymbol{p}_0 = K^{-1}\boldsymbol{r}_0,$
for $k = 0, 1, \cdots,$ until $\quad \|\boldsymbol{r}_k\| \le \varepsilon \| \boldsymbol{b} \|$
do:
begin

$$\begin{aligned}
\alpha_k &= \frac{(K^{-1}\boldsymbol{r}_k, \boldsymbol{r}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)},\\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k,\\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k,\\
\beta_k &= \frac{(K^{-1}\boldsymbol{r}_{k+1}, \boldsymbol{r}_{k+1})}{(K^{-1}\boldsymbol{r}_k, \boldsymbol{r}_k)},\\
\boldsymbol{p}_{k+1} &= K^{-1}\boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k,
\end{aligned}$$

end

ここで，前処理演算は反復ステップ k において

$$\boldsymbol{r'}_k \;=\; K^{-1}\boldsymbol{r}_k \tag{2.26}$$

を解くことに帰着される．

具体的にKを生成する方法の一つに不完全コレスキー分解(IC分解: incomplete Cholesky factorization) による方法がある [41][62]．前述のコレスキー分解 $A = LL^T$ (式 (2.12)) で

は，行列 A が疎であっても L は *fill–in* のために一般に密行列になる．*fill–in* とは，分解前の係数行列ではゼロであった要素が，厳密な分解により非ゼロの値になることである．そこで，$A = (a_{ij})$ の非零要素のインデックス集合を

$$G_A \;=\; \{(i,j)\ ;\ a_{ij} \neq 0\}$$

とし，

$$G \;\supseteq\; G_A$$

なる集合 G をあらかじめ決めておいて，A をコレスキー分解するときに L の要素中 G に属するインデックスをもつものだけを計算し，他のものは 0 のままにする．不完全コレスキー分解では，

$$\begin{aligned} A &= \bar{L}\bar{D}\bar{L}^T + R \\ &\equiv LL^T + R \\ &\equiv K + R \end{aligned} \tag{2.27}$$

と書ける．R は A と LL^T との差を表す行列である．前処理行列として不完全コレスキー分解によって生成された行列を用いる方法を，ICCG 法 (incomplete Cholesky conjugate gradient method) という．

式 (2.6) で示した A の場合には，

$$G_A \;=\; \{(i,j)\ ;\ j = i,\ i = i \pm 1,\ i = i \pm n\}$$

である．A の対角要素を a_i，副対角要素を b_i, c_i とする．また，式 (2.27) の第 1 式による $\bar{L}$ の対角要素を $\bar{a}_i$，副対角要素を $\bar{b}_i$,$\bar{c}_i$ とし，$\bar{D}$ の対角要素を $\bar{d}_i$ とする．不完全分解した結果は，

$$\begin{aligned} \bar{d}_i^{-1} &= \bar{a}_i = a_i - b_{i-1}^2\bar{d}_{i-1} - c_{i-n}^2\bar{d}_{i-n}, && 1 \le i \le N, \\ \bar{b}_i &= b_i, && 1 \le i \le N-1, \\ \bar{c}_i &= c_i, && 1 \le i \le N-n \end{aligned} \tag{2.28}$$

となる．上式中，インデックスが範囲外のものは 0 とみなす．$\bar{L}$ の非対角要素 $\bar{b}_i$，$\bar{c}_i$ は A と同じ値なので，計算は不要である [13].

このような不完全コレスキー分解から派生する様々な前処理が，これまでに提案されている [6][40].

2.3　スーパーコンピューティング

数値シミュレーションを行うとき，大規模な問題に対してはスーパーコンピュータが利用される．スーパーコンピュータの代表的なものとしては，浮動小数点演算に特化した演

算パイプラインを持つプロセッサによるベクトル計算機と，複数のプロセッサを高速ネットワークで結合した並列計算機がある．さらに，これら両者の混合形態であるベクトル並列計算機や，ワークステーション，パーソナルコンピュータ等を汎用ネットワークで結合したクラスタ型のシステムも，並列計算機の一形態である．本研究では，これら全てをスーパーコンピュータと考える．

2.3.1　ベクトルプロセッシング

ベクトルプロセッサの基本的なアーキテクチャは，図 2.3 のとおりである．

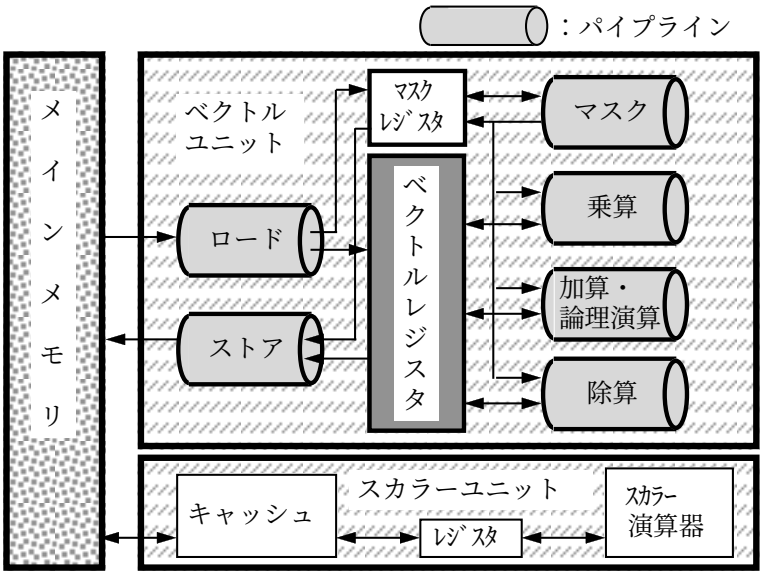

図 2.3: ベクトルプロセッサのアーキテクチャ．

被演算データの浮動小数点数は，メインメモリからベクトルレジスタへロードされる．そして，各演算に応じたパイプラインにより演算され，結果は一旦ベクトルレジスタに格納される．最後に，メインメモリにストアされる．

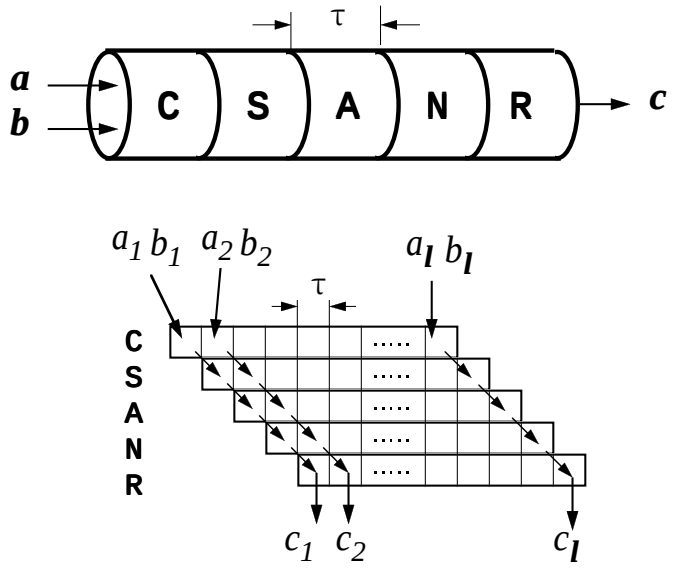

図 2.4: ベクトル処理のパイプライン.

パイプラインによる演算イメージは，図 2.4 のとおりである [43]．ここでは，

$$
\begin{aligned}
\boldsymbol{c} &= \boldsymbol{a} + \boldsymbol{b}, \\
\boldsymbol{a} &= [a_1, a_2, \cdots, a_l]^T, \\
\boldsymbol{b} &= [b_1, b_2, \cdots, b_l]^T, \\
\boldsymbol{c} &= [c_1, c_2, \cdots, c_l]^T, \\
l &: \text{ベクトルレジスタを介して一括して処理される要素数 (ベクトル長)}
\end{aligned}
\tag{2.29}
$$

に対してベクトル演算を行っている [10][19].

図 2.4 の上の図は，クロック単位 (τ) のパイプライン命令で,「C:指数比較，S:桁合わせ，A:加算演算，N:正規化，R:結果をベクトルレジスタに格納」を表す．図 2.4 の下の図はベクトルデータの演算フローである.

本論文では，特にこのようなベクトル処理に対する数式表現として，

$$
\boldsymbol{c} \begin{bmatrix} 1 \\ 2 \\ \vdots \\ l \end{bmatrix} = \boldsymbol{a} \begin{bmatrix} 1 \\ 2 \\ \vdots \\ l \end{bmatrix} + \boldsymbol{b} \begin{bmatrix} 1 \\ 2 \\ \vdots \\ l \end{bmatrix}
\tag{2.30}
$$

と表すことにする.

ベクトルプロセッシング（ここではベクトル計算機における数値演算を指す）の効率を上げるために着目すべき点は，次のとおりである [61]．(i) 回帰参照を回避すること，(ii)

ベクトル長を十分に長く取ること (図 2.5)．そして，(iii) ベクトル変数に対する連続アクセスをはじめ，十分にベクトル化されるようにプログラミングを行うことである．また，(iv) 1 つの数式内で同一のデータを参照させるようにして再利用すると，データのアクセス効率が向上する．各々を解決する基本方法は，元の物理方程式あるいは線形方程式を念頭にした数学的な裏付けが必要である．

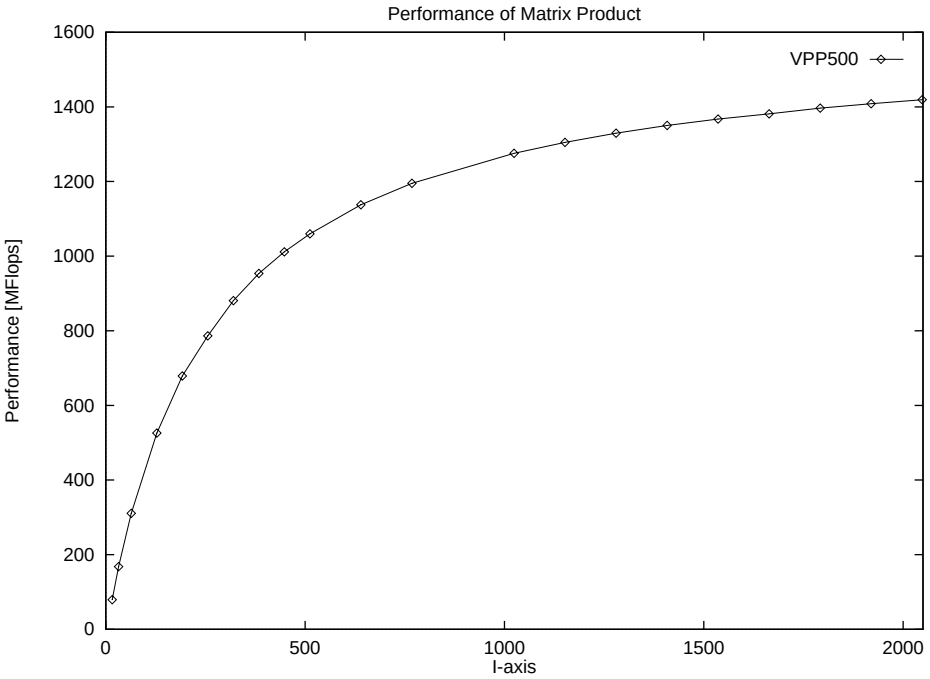

図 2.5: ベクトル長に対する計算性能.

本論文の**第 3 章**および文献 [25] では，(ii) 番目に着目して，ベクトル計算機を用いて問題を解く場合に，ベクトル長を十分に長くすると計算機性能を引き出すことができる特性を引き出すアルゴリズムを提案した.

2.3.2　パラレルプロセッシング

パラレルプロセッサ（並列計算機）は，Flynn,M. により 4 つのクラスに分類された．それは，命令の流れ (Instruction Stream) とデータ流 (Data Stream) が各々単一 (Single) か多重 (Multiple) かによって，以下のように分類される [24][46][60].

SISD：単一命令単一データ流 (Single Instruction stream Single Data stream) 従来の直列のフォン・ノイマン型の計算機，すなわち単一スカラー計算機はこのクラスである.

SIMD：単一命令多重データ流 (Single Instruction stream Multiple Data stream) 1 つの命令により複数の演算器が制御される方式である．ベクトルの各要素は別個のデータ

ストリームを構成するとみなして，ベクトル命令をもつ計算機はこのクラスである．ベクトル処理はパイプライン方式とプロセッサアレイ（1 つの命令が複数の演算器にブロードキャストされる）がある．

MISD : 多重命令単一データ流 (Multiple Instruction stream Single Data stream) 理論上のクラスであり，実用的には意味がなく，実例もなく．

MIMD : 多重命令多重データ流 (Multiple Instruction stream Multiple Data stream) 多重の命令ストリームは複数の命令処理装置を意味する．したがって，複数のデータストリームも意味する．このクラスは，すべてのマルチプロセッサ・システムを含む．

本論文において，パラレルプロセッサは MIMD のクラスに属するものを指す．

パラレルプロセッシング（ここでは並列計算機における数値演算を指す）の効率を上げるために着目すべき点は，次のとおりである．(i) 回帰参照を回避すること，(ii) 全プロセッサのロードバランスのばらつきを抑えること，そして，(iii) 並列化におけるデータ分割方法の設計や (iv) プロセッサ内におけるデータの再利用などにより，(v) プロセッサ間のデータ転送量を最小限に留めることである．これらにおいても，ベクトルプロセッシングと同様の数学的裏付けが必要である．

2.3.3　スーパーコンピュータによる数値解析の問題点

これらのスーパーコンピュータを用いて数値計算を行うとき，数学上の観点のみから開発された前処理をそのまま適用することは，しばしば，それらの計算手順がスーパーコンピュータの特化されたアーキテクチャに適合せず，結果として，前処理とスーパーコンピュータともにその効果が発揮されなくなる例が少なくない．また，アルゴリズムそのものにおいても同様の問題が生ずるものもある．

その様な問題が生ずるプログラム例について説明する．

プログラム: 2.1 (回帰参照)

```
      do  10  I = 2, 50
        a(I) = a(I-1) + b(I)
   10 continue
```

この**プログラム** 2.1 では，変数 “a” は，更新された直後にその値が参照されている．これをプログラムにおける回帰参照[3] と呼ぶ．このとき，計算機のアーキテクチャに応じて，以下のとおり計算される．

[3] プログラム上のデータ参照や演算では，回帰参照 (演算)，再帰参照 (演算) や巡回演算など様々な呼び方があり，数式についても同じように回帰演算や再帰演算などと呼ばれる．本論文では，プログラムなどアーキテクチャ面を問題にする場合を “回帰参照” ，アルゴリズムなど数式面を問題にする場合を “再帰関係式” とした．

【スカラー計算】

$$\begin{array}{rcl} a(2) & = & a(1)+b(2) \\ a(3) & = & a(2)+b(3) \\ \vdots & & \vdots \quad \vdots \\ a(50) & = & a(49)+b(50) \end{array}$$

【ベクトル計算】

$$\boldsymbol{a}\begin{bmatrix} 2 \\ 3 \\ \vdots \\ 50 \end{bmatrix} = \boldsymbol{a}\begin{bmatrix} 1 \\ 2 \\ \vdots \\ 49 \end{bmatrix} + \boldsymbol{b}\begin{bmatrix} 2 \\ 3 \\ \vdots \\ 50 \end{bmatrix}$$

【並列計算 (CPU1)】

$$\begin{array}{rcl} a(2) & = & a(1)+b(2) \\ a(3) & = & a(2)+b(3) \\ \vdots & & \vdots \quad \vdots \\ a(25) & = & a(24)+b(25) \end{array}$$

【並列計算 (CPU2)】

$$\begin{array}{rcl} a(26) & = & a(25)+b(26) \\ a(27) & = & a(26)+b(27) \\ \vdots & & \vdots \quad \vdots \\ a(50) & = & a(49)+b(50) \end{array}$$

スカラー計算機では，逐次実行されるため，特に問題は生じない．ベクトル計算機では，演算を実施する前に，右辺のベクトル変数 $\boldsymbol{a}, \boldsymbol{b}$ をベクトルレジスタにロードしてしまう．そのため本来ならば，右辺のベクトル変数 $\boldsymbol{a}(2) \sim \boldsymbol{a}(49)$ は演算により更新された値を用いるべきところが，予めロードされた，更新前の値を参照することになる．したがって，本来とは異なる演算結果となる．並列計算機では，例えば，CPU2 における $a(25)$ の値は，CPU1 で計算された $a(25)$ を参照しなければなない．したがって，結果として逐次計算に他ならず，並列実行は不可能となる．

科学技術計算に現れる偏微分方程式を離散化すると，多くの場合，式 (2.3) で示したような漸化式となる．このとき，アルゴリズムによっては，更新された値を参照する再帰関係式[3] となる．また，アルゴリズムの計算手順そのものが再帰関係式となるものもある．以下に，このような再帰関係式となる数値解析手法の一例を挙げる．下線を引いたものは，アルゴリズムも示した．

- ガウス消去法，LU 分解，コレスキー分解
- ガウスーザイデル法，SOR 法

$$\boldsymbol{x}^{(k+1)} = D^{-1}(-L\boldsymbol{x}^{(k+1)} - U\boldsymbol{x}^{(k)} + \boldsymbol{b}), \quad k : \text{反復回数} \tag{2.31}$$

このときの，2 次元問題における格子点の参照を表したものが図 2.6 である．図中の L, U が，上式の L, U を意味する．

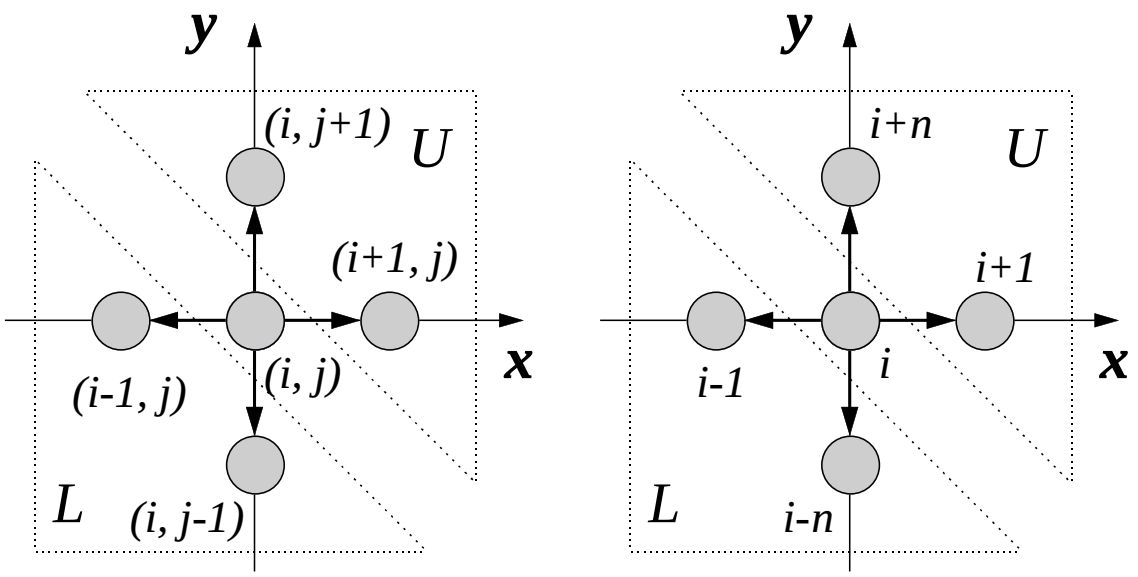

図 2.6: データ参照.

これら，格子点のインデックス表現 (式 (2.32)，図 2.6 左) およびベクトルのインデックス表現 (式 (2.33)，図 2.6 右) を用いると，

$$x_{i,j}^{(k+1)} = \frac{1}{4}(x_{i-1,j}^{(k+1)} + x_{i,j-1}^{(k+1)} + x_{i+1,j}^{(k)} + x_{i,j+1}^{(k)} + b_{i,j}) \tag{2.32}$$

または，

$$x_{i}^{(k+1)} = \frac{1}{4}(x_{i-1}^{(k+1)} + x_{i-n}^{(k+1)} + x_{i+1}^{(k)} + x_{i+n}^{(k)} + b_{i}) \tag{2.33}$$

という漸化式で表される．これらから，ベクトル $\boldsymbol{x}^{(k+1)}$ に関する要素が，再帰関係式となっていることが分かる．

- 不完全 LU，不完全コレスキー分解 による前処理
  【前処理行列の分解】

$$K = LL^T = \bar{L}\bar{D}\bar{L}^T \tag{2.34}$$

となるように分解する (**第** 2.2.3 **節**参照).

$$\begin{aligned} \bar{d}_i^{-1} &= a_i - b_{i-1}^2\bar{d}_{i-1} - c_{i-n}^2\bar{d}_{i-n}, \quad 1 \le i \le N, \\ \bar{b}_i &= b_i, \qquad\qquad\qquad\qquad\quad 1 \le i \le N-1, \\ \bar{c}_i &= c_i, \qquad\qquad\qquad\qquad\quad 1 \le i \le N-n \end{aligned} \tag{2.35}$$

【前処理の演算】

$$K\boldsymbol{s} = \boldsymbol{r} \quad \rightarrow \quad \bar{L}\bar{D}\bar{L}^T\boldsymbol{s} = \boldsymbol{r}, \tag{2.36}$$

$$\text{として,} \qquad \bar{L}\boldsymbol{y} = \boldsymbol{r}, \quad \bar{D}\bar{L}^T\boldsymbol{s} = \boldsymbol{y} \tag{2.37}$$

の 2 つの線形方程式を解く．これらの計算手順は以下のとおりである．

$$\check{y}_i = r_i - b_{i-1}\check{y}_{i-1} - c_{i-n}\check{y}_{i-n}, \qquad i = 1, 2, \cdots N, \tag{2.38}$$

$$\hat{s}_i = \bar{d}_i^{-1}\check{y}_i - (b_i\hat{s}_{i+1} + c_i\hat{s}_{i+n}), \qquad i = N, N-1, \cdots, 1. \tag{2.39}$$

ここでは，分解過程 (2.35) の $\bar{d}_i$ と演算過程 (2.38) の $\check{y}_i$ と (2.39) の $\hat{s}_i$ がそれぞれ再帰関係式となっている．

次節において，これらの問題点に対する従来の解決法を示す．

2.4　前処理の役割

線形方程式を反復法を用いて解く場合，解くべき方程式 $A\boldsymbol{x} = \boldsymbol{b}$ に対して変換を施すと，その収束性や計算精度を高めるなどの影響を与える．通常，前処理 (preconditioning) とはこの変換手法を指す．しかし，これらの前処理の効果は，個々の問題の解くべき性質に応じてしばしば変わる．したがって，元の物理方程式やその離散化方法に応じて，線形方程式の係数行列の性質や構造は大きく異なるため，適した前処理も異なる．さらに，組合わせる解法によっても適した前処理が異なる場合も少なくない．これらの選択方法は，経験によるところが大きく，様々な問題に応じた前処理が必要である．また一方で，スーパーコンピュータを用いた数値計算においては，そのアーキテクチャに適合するような前処理手法も数多く提案されており，これらは計算効率を向上させる．

本研究において，これら前処理を二つの観点から考えることにする．

2.4.1　数学的な観点からの前処理

一つは，数学的な観点から提案される前処理であり，これまで一般的に考えられてきた収束性や計算精度および計算効率の向上を念頭にしている．本論文では，この前処理を “数学の観点からの前処理” と呼ぶ．代表的なものでは，直接解法に対しては，LU 分解の計算誤差を抑えるための “スケーリング (scaling)” や分解過程において発生する誤差を抑えるための “軸選択 (pivoting)”，また，反復解法においては，その収束を速めるために用いられる “不完全分解” により得られる前処理行列を作用させる方法が挙げられる．また，非定常反復法の前処理演算 $\boldsymbol{r'} = K^{-1}\boldsymbol{r}$ を含めて線形方程式そのものにおける “演算方法” についても考えられる．これは，丸め誤差の影響を無視すれば，

- 代数的に同値な求解を行ない，

- より安定な求解を行い,
- 演算量が少なくなる

ことを目的に変形する方法を指す．具体的な例については，**第 2.2.3 節 前処理付き反復解法** に示したとおりである．

2.4.2　アーキテクチャの観点からの前処理

もう一つは，計算機アーキテクチャの観点から提案される前処理で，主に計算効率の向上を念頭にしている．本論文では，この前処理を “アーキテクチャの観点からの前処理” と呼ぶ．この前処理では，アルゴリズムをベクトル化，並列化する上での問題点（主に回帰参照）と同様の問題が生ずるが，反復解法における前処理では，“不完全” あるいは “近似” であっても十分にその役割を担うことが可能であり，また前述した，解くべき問題の性質にも依存することから，結果として直接解法に対するベクトル，並列化の手法以上に様々な手法が考えられる．代表的なものでは，“ブロック化” や “再順序付け” などがあり，これまでに様々な手法が提案されている [7]．またそれ以外にも，非定常反復法の前処理演算 $\boldsymbol{r'} = K^{-1}\boldsymbol{r}$ も含めて，線形方程式そのものにおける演算量の減少を主な目的とした “演算方法” 自体が挙げられる．

“ブロック化” の主な目的は，データアクセスの局所性を高めることであり，以下の特長がある [2][9]．

- 反復法に対して，不完全分解などの “数学の観点からの前処理” をほどこす場合，対角ブロックから構成される前処理行列を生成することで，強制的に回帰参照を解消できる．これは，係数行列 A に対して，前処理行列 K が不完全な近似でも良い利点を活かし，かつ，その効果が見込まれるためである．最も簡単な例は，対角要素のみからなるスケーリングである．
- データ再利用によるアクセス効率の向上やデータ転送量を減少させる．

代数的には，行列の対角上に形成される互いに重複しない任意のサイズのブロックが対角ブロックと呼ばれるが，前処理においては，主に物理領域に基づいて分割（2次元の場合では，x 軸に平行に並ぶ格子点ごとに分割）されたものを対象として研究がなされてきている．この対角ブロックから構成される前処理は “ブロック前処理” と呼ばれ，この前処理による収束性に関する報告 [2][5][9][63] やベクトル計算や並列計算に関する報告 [3][36] が行われている．本論文においても，これに基づいて “ブロック前処理” を定義する．

本論文の**第 5 章**および文献 [27] では，物理問題に起因する対角ブロックに特徴のある係数行列に対して，新しいブロック前処理を提案した．

“再順序付け” の主な目的は回帰参照の回避であり [12]，細かく分類すると次の二種類がある．一つは，元の方程式と同値であり，結果を変えない “計算順序の入れ換え” ともう一つは，互いに漸化式上直接の参照関係を持たない全ての変数を選び出して，最初に番

号付けする “未知数の再順序付け” である [10]．後者では，元の方程式と同値とはならず計算結果も変わる．ここでは，回帰参照の回避という点のみを考え，これら二種類をまとめて “再順序付け” と呼ぶ．以下に，これに基づく代表的な方法とそれらの特長を示す．

- ツイスト分解：計算順序入れ換えの一つである．3 重対角行列やブロック 3 重対角行列に対して，対角要素の第 1 番目と最後の要素の両側から中央の要素まで分解を行ってもそれは通常の分解と同値であり，並列性が生じる [63][64]．これを利用して，不完全分解においても並列性を高めることができる．

  本論文の**第 3 章**および文献 [25] では，この分解方法をさらに改良し，ベクトル計算機における計算効率を向上させる前処理を提案した．ツイスト分解の詳細については，**第 3 章**で説明する．

- ハイパープレーン (超平面) 法：計算順序入れ換えの一つである．不完全分解において，回帰参照の回避やベクトル長を長くする．

6	8	9
3	5	7
1	2	4

21	26	30	33	35	36
15	20	25	29	32	34
10	14	19	24	28	31
6	9	13	18	23	27
3	5	8	12	17	22
1	2	4	7	11	16

図 2.7: ハイパープレーンオーダリング ($n = 3$ と $n = 6$).

図 2.7 の左図 ($n = 3$) と図 2.2 のナチュラルオーダリングとを比較すると，ハイパープレーンでは計算順序が左斜め上へ順番となっていることが確認できる．各々の斜めに並ぶ番号ごとに，“{1}{2,3}{4,5,6}{7,8}{9}” とグループ分けする “{1}” の点の計算が完了していれば，“{2,3}” はどのような順序で計算してもその結果を変えない．同様に，“{2,3}” のグループの計算が完了していれば，“{4,5,6}” はどのような順序で計算しても良い．これらの計算では再帰関係式とはならず，さらに，同一グループ内の計算は互いに独立しており並列計算が可能である．ベクトル計算の場合，右図のとおり n の値が大きいとベクトル長が大きくなる．

- Red-Black [4] などマルチカラーオーダリング：未知数の再順序付けの一つである．SOR 法やガウス－ザイデル法の演算が再帰関係式であるため，ベクトル計算機や並列計算機でそのままのアルゴリズムを実行すると計算機の特長を損ねてしまう．これらの計算機に解法を適合させるために改良したアルゴリズムである [18][60]．これ

[4] Odd-Even，チェッカーボードオーダリングとも呼ばれる．

はまた，非定常反復法のための “数学の観点からの前処理” （不完全分解など）をこれらの計算機に適合させるときにも用いられる．

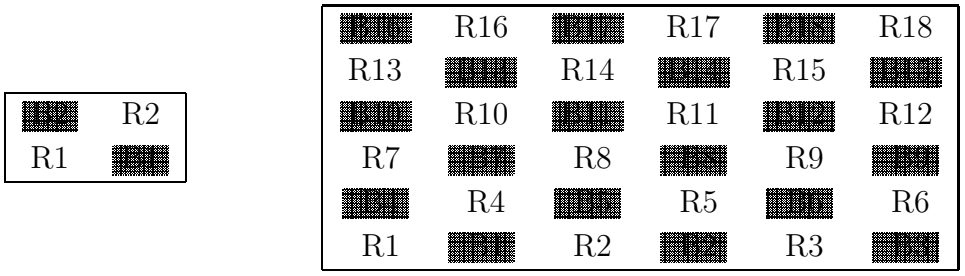

図 2.8: レッドブラックオーダリング ($n = 2$ と $n = 6$).

図 2.8 は $n = 2$ と $n = 6$ のときのオーダーリングである．R は Red の要素，網掛けされている B は Black の要素を表し，数字は各々の色における番号である．ここでは，説明を簡単にするために，**第** 2.1 **節**で取り上げた例題を用い，$n = 2$ について説明する．元の線形方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を，式 (2.40) と表す．

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_{R1} \\ x_{B1} \\ x_{B2} \\ x_{R2} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{2.40}$$

これを Red-Black オーダリングによる再順序付けを行うと，次式 (2.41) となる．

$$\begin{bmatrix} 4 & 0 & -1 & -1 \\ 0 & 4 & -1 & -1 \\ -1 & -1 & 4 & 0 \\ -1 & -1 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_{R1} \\ x_{R2} \\ x_{B1} \\ x_{B2} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_4 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.41}$$

この式に対して，

$$\begin{aligned} D_R &= D_B = 4I, \quad C = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}, \\ \boldsymbol{x}_R &= \begin{bmatrix} x_{R1} \\ x_{R2} \end{bmatrix}, \quad \boldsymbol{x}_B = \begin{bmatrix} x_{B1} \\ x_{B2} \end{bmatrix}, \quad \boldsymbol{b}_1 = \begin{bmatrix} b_1 \\ b_4 \end{bmatrix}, \quad \boldsymbol{b}_2 = \begin{bmatrix} b_2 \\ b_3 \end{bmatrix} \end{aligned}$$

とすると，線形方程式は次式として表される．

$$\begin{bmatrix} D_R & C \\ C^T & D_B \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_R \\ \boldsymbol{x}_B \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix} \tag{2.42}$$

これをガウス－ザイデル法で解くと，

$$\begin{bmatrix} D_R & 0 \\ C^T & D_B \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_R^{(k+1)} \\ \boldsymbol{x}_B^{(k+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix} - \begin{bmatrix} 0 & C \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_R^{(k)} \\ \boldsymbol{x}_B^{(k)} \end{bmatrix} \tag{2.43}$$

の形で表される．これより，次の 2 式が得られる．

$$\boldsymbol{x}_R^{(k+1)} = D_R^{-1}(\boldsymbol{b}_1 - C\boldsymbol{x}_B^{(k)}), \tag{2.44}$$

$$\boldsymbol{x}_B^{(k+1)} = D_B^{-1}(\boldsymbol{b}_2 - C^T\boldsymbol{x}_R^{(k+1)}). \tag{2.45}$$

このとき，第 $k+1$ 番目反復において，Red の点を求める計算 (2.44) では，第 k 番目で求めた Black の点のみを参照し，次いで，Black の点を求める計算 (2.45) では，第 $k+1$ 番目で求めた Red の点のみを参照する．したがって，再帰関係式は生じない．

しかし，以上挙げた “アーキテクチャの観点からの前処理” の手法は，スーパーコンピュータに適合した計算方法を実現する一方で，反復法自体や反復法の収束を速めるための “数学の観点からの前処理” に応用した場合には，前処理本来の収束性に劣る場合も少なくない．

このとき，解くべき問題に応じた “演算方法” の提案により，演算量減少の観点で，さらに計算効率を向上させることで，線形方程式全体の求解時間を短縮できる．

また，本論文の**第 4，5 章**および文献 [26][27] では，元の演算と同値な，より計算効率の高い “演算方法” による前処理を提案した．