Vol.2012-HPC-135 No.34
2012/8/3

通信時間の隠蔽処理を施した並列 SOR 法

中村真輔[1,a)]　小澤一文[1,b)]　渋谷広志[2,c)]

概要：連立一次方程式の反復解法として SOR 法があり，収束が速いことから特に大規模で疎な係数行列を持つ問題に対して頻繁に用いられる解法の一つとなっている．しかしながら，1 反復あたりの計算順序を維持する必要から並列化に不向きであるという欠点も持っている．それでも大規模な問題へ適用する場合には並列化が求められることから，その実装には何らかの工夫を要することとなる．そこで本発表では，係数行列が疎である場合について SOR 法による計算値の依存関係に着目し，プロセス間通信と計算値の更新を重複させる並列化手順について提案し，分散メモリ型の並列計算機において高い効率を得ることができることを数値実験の結果により示す．

1. はじめに

n 次の連立一次方程式

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix},\ \boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix},\ \boldsymbol{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

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

の反復解法の一つとして SOR 法があり，特に大規模で疎な係数行列を持つ問題に対して有効であるとされている．

また近年の計算機の発達で，より大規模な問題を解くことも可能となっている．それでも，より大規模な問題に適用する場合は計算量の増大が避けられない．膨大な計算量に対して計算時間の短縮を図ることができる手法として並列化がある．しかし一方で，並列化は計算の依存関係が複雑なアルゴリズムには不向きである．SOR 法もそのようなアルゴリズムの一つである．そのため，SOR 法を並列化するためには，計算順序を入れ替えることで並列化可能な状態にする処理が必要となる．

そのための手順の一つとして，各変数を互いに直接には依存しない複数の部分集合に分けて，各集合ごとに並列計算する Multicolor ordering がある．その中では，特に偏微分方程式の 5 点差分問題で使われる Red-Black ordering が知られている．

しかし Multicolor ordering では各集合で並列計算した後に同期と通信が必要になり，また一般の問題に適用すると部分集合の数が大きくなるため，特に分散メモリ型の並列計算機において効率が低下する傾向がある．そこで本発表では特に分散メモリ型での並列計算を考え，各変数が各プロセスにどのように割り当てられているかを考慮に入れて，他プロセスに依存しない変数の計算時間中に通信を行なう手順について提案する．

なお以下では，簡単のために係数行列 A は正値対称であると仮定する．

2. SOR 法

反復 k 回目の近似解を $\boldsymbol{x}^{(k)} = (x_1^{(k)}, \ldots, x_n^{(k)})^T$ とすると，初期値 $\boldsymbol{x}^{(0)}$ から開始した SOR 法の k 回目の反復式は

$$\begin{cases} \tilde{x}_i^{(k)} = b_i - \displaystyle\sum_{j=1}^{i-1} a_{i,j} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)}, \\ x_i^{(k)} = x_i^{(k-1)} + \omega(\tilde{x}_i^{(k)} - x_i^{(k-1)}) \end{cases} \tag{2}$$

と表わされる．ここで，式 (2) の右辺第 3 項に反復 k 回目で得られる値 $x_1^{(k)}, \ldots, x_{i-1}^{(k)}$ が含まれることから，$x_i^{(k)}$ を得るためにはこれらの値が先に計算されていなければならず，そのため $x_i^{(k)}$ の計算順序（オーダリング）を反復ごとに一定に保つことが必要となり，並列化の際にはこのことが障害となる．

しかし一般的には SOR 法の適用対象となる問題では係数行列 A は疎である場合が多いため，式 (2) の右辺には $x_1^{(k)}, \ldots, x_{i-1}^{(k)}$ のうちごく少数しか現われていないはずである．このことを利用して，係数行列の疎構造を解析する

[1] 秋田県立大学システム科学技術学部
Faculty of System Science and Technology, Akita Prefectural University, Yuri-Honjo, Akita, Japan
[2] 秋田県立大学大学院システム科学技術研究科
Graduate School of System Science and Technology, Akita Prefectural University, Yuri-Honjo, Akita, Japan
[a)] shinsuke@akita-pu.ac.jp
[b)] ozawa@akita-pu.ac.jp
[c)] shibuya@pna.eis.akita-pu.ac.jp

ことで，並列化が容易になるオーダリングを構成することができる．

今回の発表では，PC クラスタ等の分散メモリ型並列計算機による並列化を考え，特にプロセス間の通信によるオーバーヘッドを解消するためのオーダリングの構成を目的とする．

3. 係数行列のグラフ表現

SOR 法の計算式における $x_1^{(k)}, \ldots, x_n^{(k)}$ の依存関係を明確にするために，係数行列 A の隣接グラフを用いる．すなわち，節点集合 $V = \{v_1, \ldots, v_n\}$ について，辺集合を

$$E = \{e_{i,j} \mid a_{i,j} \neq 0,\ i > j\}$$

と定義したグラフを行列 A の隣接グラフと呼ぶ．ここで A は対称であるとの仮定から，グラフ $\{V, E\}$ は無向グラフとなる．例えば，下の行列 A をグラフで表わすと**図 1** のようになる．

$$A = \begin{pmatrix}
3 & 1 & & & & & & 1 & & \\
1 & 4 & 1 & & & & 1 & 1 & & \\
 & 1 & 3 & 1 & 1 & & & & & \\
 & & 1 & 3 & & 1 & & & & \\
 & & 1 & & 2 & 1 & & & & \\
 & & & 1 & 1 & 2 & & & & 1 \\
 & 1 & & & & & 3 & & 1 & 1 \\
1 & 1 & & & & & & 3 & & \\
 & & & & & & 1 & & 3 & 1 \\
 & & & & & 1 & 1 & & 1 & 3
\end{pmatrix}$$

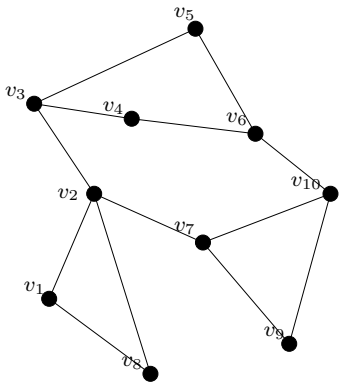

図 1 行列 A の隣接グラフ
Fig. 1 Adjacency graph of matrix A.

以下で節点 v_i に隣接する節点の集合を表わす記号として

$$\mathrm{Adj}\{v_i\} = \{v_j \mid e_{i,j} \in E \text{ または } e_{j,i} \in E\}$$

を用いる．

4. SOR 法の並列化

4.1 Multicolor ordering による並列化

SOR 法の並列化手法としては Multicolor ordering[1] を用いたものが一般的である．その手順をここで説明する．

事前に節点を複数の色に分類する．ここで，互いに隣接する節点は同じ色に属してはならない．よって，$e_{i,j} \in E$ であれば v_i と v_j は違う色に分類する．色分けの手法としては，例えば greedy multicoloring algorithm がある．

Algorithm.1: Greedy multicoloring

$i = 1, \ldots, n$ について，
$\mathrm{Color}(i) := \min \{k > 0 \mid k \neq \mathrm{Color}(j) \forall j \in \mathrm{Adj}\{v_i\}\}$.

ここで，分類に用いる色数を $c_{\max} = \max_i \mathrm{Color}(i)$ とおく．

Multicoloring を用いてオーダリングを変更した後に，SOR 法のオーダリングを $\mathrm{Color}(i)$ の若い順に変更するすなわち，反復 k 回目の処理は以下のようになる．

Algorithm.2: Multicolor SOR 法

$c = 1, \ldots, c_{\max}$ の順に，
　$\mathrm{Color}(i) = c$ を満たす全ての i について，
　　$x_i^{(k)}$ を，$x_j^{(k)}$ $(\mathrm{Color}(j) < c)$ と
　　$x_j^{(k-1)}$ $(\mathrm{Color}(j) > c)$ を用いて更新．

このオーダリングであれば，同じ色の節点は互いに隣接していないために，上のアルゴリズムの内側の繰り返しが i について並列化可能である．ただしこの手法の場合，c を 1 進めるごとにプロセス間の同期・通信が必要となるため，色数 $c_{\max}$ が多くなればなるほど，並列化によるオーバーヘッドが増大し，効率が低下するという問題もある．

4.2 提案手法による並列化

次に SOR 法の計算を P 個のプロセスへの割り当てを考えるために，グラフを P 個の部分グラフ $G_1 = \{V_1, E_1\}, \ldots, G_P = \{V_P, E_P\}$ に分割する．ここで $x_i^{(k)}$ の計算をプロセス p が担当する場合は $v_i \in V_p$ となるように分割する．

さらに，v_i に隣接する節点の集合 $\mathrm{Adj}\{v_i\}$ について $\mathrm{Adj}\{v_i\} \subset V_p$ である節点の集合を内部集合，それ以外の節点の集合を接続集合と呼ぶことにする．内部集合に属する変数は他プロセスに直接は依存せずに計算できるが，接続集合に属する変数は他プロセスの計算結果を直接用いる．そのため，接続集合の計算は全てのプロセスに渡ってオーダリングを守る必要があるが，内部集合の計算はプロセス間で前後しても問題はない．よって，プロセス p の 1 反復あたりの処理を以下のようにする．

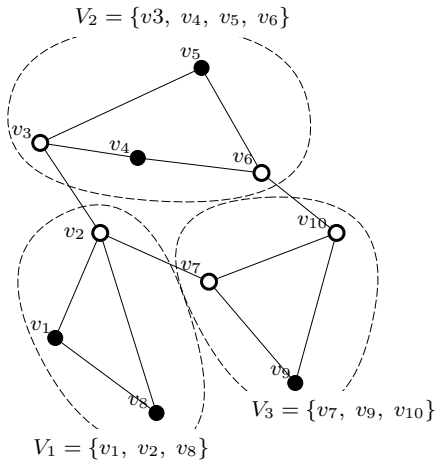

図 2 内部集合と接続集合
●は内部集合　○は接続集合

Fig. 2 Inner vertices and connecting vertices.
● are inner vertices, ○ are connecting vertices.

Algorithm.3: 通信隠蔽処理を施した並列 SOR 法

(1) プロセス 1, ..., $p-1$ から反復 k 回目の変数値を，プロセス $p+1$, ,..., P から反復 $k-1$ 回目の変数値を，それぞれ接続集合の計算に必要なものだけ受信.
(2) 接続集合の変数値を更新.
(3) 接続集合の計算結果を送信.
(4) 内部集合の変数値を更新.

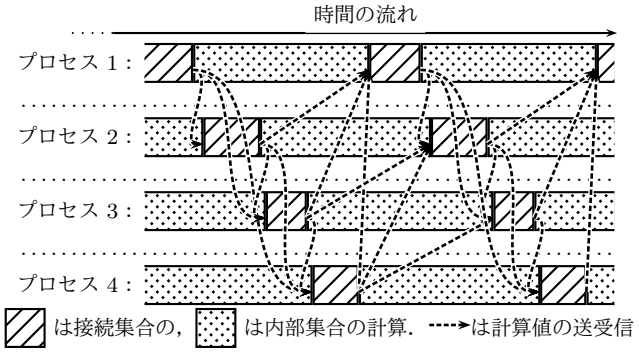

図 3 通信隠蔽を施した計算の流れ

Fig. 3 Computation order with overlapping communication

ここで，各送受信に非同期通信を用いることにより，通信時間を内部集合の計算時間によって隠蔽することができる [2].

図 1 の例について，内部集合／接続集合の分類を**図 2** に，計算の流れを **図 3** に示す.

5. 数値実験

5.1 実験条件

提案手法の性能を検証するために，Florida 大学の Sparse Matrix Collection [3] からダウンロードした 4 個の疎行列を係数行列 A に持つ連立一次方程式を対象に数値実験を行なった．**表 1** に用いた疎行列のリストを示す．なお，SOR 法の加速パラメータ ω は 1.0 から 1.9 まで 0.1 刻みで最も収束が速かったものを用いている.

表 1 A に用いた疎行列
Table 1 Examples of sparse matrices A

行列名	n	非零要素数	ω
thermomech_TK	102158	711558	1.9
shipsec8	114919	6653399	1.5
G2_circuit	150102	726674	1.9
parabolic_fem	525828	3674625	1.9

また，方程式の右辺 $\boldsymbol{b}$ は

$$\boldsymbol{b} = A(0,1,2,0,1,2,\ldots)^T$$

として，反復の初期値 $\boldsymbol{x}^{(0)}$ と反復終了条件は

$$\boldsymbol{x}^{(0)} = \boldsymbol{0}, \quad \max_{i=1,\ldots,n} \left| a_{i,i}(\tilde{x}_i^{(k)} - x_i^{(k)}) \right| \leq \|\boldsymbol{b}\|_1 \times 10^{-8}$$

を用いる.

計算環境は以下のものを用いている.

CPU Intel Xeon Processor E5440（2.83 GHz，4 コア）.
ノード構成 1 ノードあたり 2 CPU，4 ノード.
コンパイラ intel C compiler version 10.1 + MPICH2 version 1.4.1p1.

並列化については $P = 4, 8, 16, 32$ としてプロセスが 4 ノード全てに均等に割り当てられるようにしている．また，並列化の際のグラフ分割については Scotch[4] を用いている.

比較対象として，オーダリングをもとのままで計算した逐次 SOR 法の他に，Greedy multicoloring algorithm によってオーダリングを決定した Multicolor SOR 法を適用する．ここで，Multicolor SOR 法についても通信時間をできる限り隠蔽するために，各色番号 c ごとの計算手順を以下のように分ける.

Algorithm.4: 通信隠蔽処理つき Multicolor SOR 法

$c = 1, \ldots, c_{\max}$ について，
(1) Color(i) = c である x_i のうち接続集合に属するものを更新.
(2) 接続集合のうち Color(i) = c である x_i を送信.
(3) Color(i) = c である x_i のうち内部集合に属するものを更新.
(4) 他プロセスの接続集合に属する x_i のうち，Color(i) = c のものを受信.

この手順を用いることで，(3) の計算時間が充分であれば，(2) の送信から (4) の受信までの通信時間を隠蔽できる.

5.2 結果

まず並列化のためのオーダリング変更による反復回数への影響を確認するために，逐次 SOR 法，Multicolor SOR 法（MC-SOR），提案手法の反復回数を **表 2** に示す．ここで，提案手法のみ反復回数がグラフの分割数に依存するため，各 P ごとに反復回数を示す．

表 2 反復回数（×100）
Table 2 Iteration numbers (×100).

行列名	逐次 SOR	MC-SOR	提案手法（$P=4,\ldots,32$） P=4	8	16	32
thermomech_TK	309	333	256	303	317	270
shipsec8	235	202	258	238	240	234
G2_circuit	201	200	201	201	201	201
parabolic_fem	279	299	279	279	279	279

Multicolor SOR 法，提案手法ともに，オーダリングの変更によって逐次 SOR 法と比較して反復回数が増減しているが，その変化の傾向は一様ではなく，また大きく変化しているとも言えない．そのため，両手法の性能の優劣は反復回数よりも 1 反復あたりの計算時間によって決定されるものと考えられる．

そこで，逐次 SOR 法を基準とした効率を，反復回数の増減による影響を排除するために 1 反復あたりの計算時間をもとに算出した．その結果を **表 3** に示す．

表 3 1 反復あたりの計算時間による効率
Table 3 Efficiency by computational time per iteration

行列名	解法	$P=4$	8	16	32
thermomech_TK	MC-SOR	1.13	0.91	0.64	0.32
($c_{\max}=7$)	提案手法	1.13	1.12	0.98	0.67
shipsec8	MC-SOR	0.75	0.49	0.31	0.13
($c_{\max}=54$)	提案手法	0.94	0.71	0.54	0.32
G2_circuit	MC-SOR	1.20	0.94	0.46	0.24
($c_{\max}=4$)	提案手法	1.20	1.05	0.65	0.26
parabolic_fem	MC-SOR	1.12	1.10	1.11	0.72
($c_{\max}=5$)	提案手法	1.14	1.12	1.34	1.17

全ての結果において，提案手法は Multicolor SOR 法よりも高い効率が得られたことが 表 3 より確認できる．また，特にプロセス数を多くとった場合に差が大きくなっている．

Multicolor SOR 法では 1 反復あたりの通信を色ごとに分けて行なうために，一方で通信によるオーバーヘッドがより大きくなっており，他方でそれを隠蔽するための内部集合の計算も小分けにされるため隠蔽処理による効果も低下するものと考えられる．特にその傾向は shipsec8 のように N_c が大きくなった場合に顕著である．

それに対して提案手法では，あるプロセスから別のプロセスへの計算値の送信は 1 反復あたり 1 回のみであるため，並列化によるオーバーヘッドも最小限（行列 - ベクトル積と同程度）に抑えられる．またそれを隠蔽するための内部集合の計算も各反復ごとにまとめて行なうため，隠蔽処理による効果も Multicolor SOR 法よりは高く，それゆえ 表 3 のような結果が得られたものと考えることができる．

しかし提案手法でも，shipsec8 や G2_circuit を A として $P=32$ で解いた場合のように効率が大きく低下することもあり，その改善が必要であると考えられる．

6. 効率低下の原因と改善

まずは，プロセス数をより多くとった場合に発生しうる現象について考えてみる．

図 3 から，通信時間の隠蔽が充分に機能するためには，全プロセス $p=1,\ldots,P$ について少なくとも

$$\begin{pmatrix}\text{プロセス } p \text{ の内部集合}\\ \text{の計算時間}\end{pmatrix} > \sum_{q\neq p}\begin{pmatrix}\text{プロセス } q \text{ の}\\ \text{接続集合の}\\ \text{計算時間}\end{pmatrix}$$

が成り立っていることが必要である．しかし，プロセス数を多くすることでグラフの分割数も増えるため，それに伴ない接続集合に属する節点数は増え，内部集合に属する節点数は減ってしまう．よって，接続集合の計算が長くなり内部集合の計算は短くなるため，Algorithm.3 の 3 行目で接続集合の値を送信してから次の反復の 1 行目で他プロセスの接続集合の値を受信するよりも早く内部集合の計算が終了してしまうことになり，その結果として **図 4** のように受信待ちが発生してしまう．

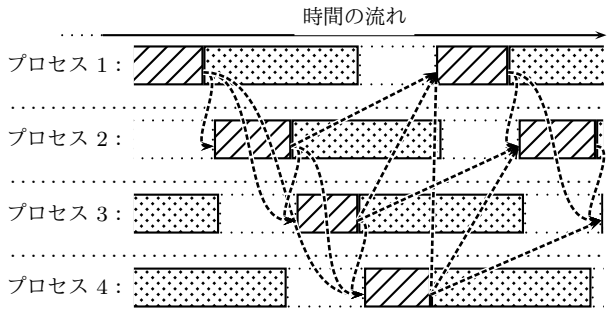

図 4 通信隠蔽が不十分な場合
Fig. 4 An example when the overlapping communication is unsatisfactory.

そこで，接続集合の計算の高速化を考えてみる．その手法としては，以下の 2 通りが考えられる．

(1) 接続集合に属する節点を減らす．
(2) 接続集合の計算も並列化する．

このうち (1) については今回用いている SCOTCH のように一般的なグラフ分割ツールがもともと目指しているところであり，これ以上の改善は困難である．そこで，今回は

(2) を試みる．

接続集合の計算が並列化できないのはプロセスを跨いだ依存関係があるためだが，プロセスの組み合わせ全てについて直接的な依存関係が存在するとは限らない．そこで，例えばプロセス 2 と 3 の間に依存関係が無いものと仮定してみる．この場合，プロセス 3 の接続集合の計算はプロセス 2 のそれを待たずに開始できるため，計算の流れは**図 5** のようにプロセス 2 と 3 の接続集合の計算が並列になる．もちろん，プロセス数 4 に対して 2 並列では並列性が高いとは言い難いが，プロセス 2 と 3 が接続集合を計算する間を他プロセスが内部集合の計算で消化することができれば，通信待ち時間も解消できると考えられる．

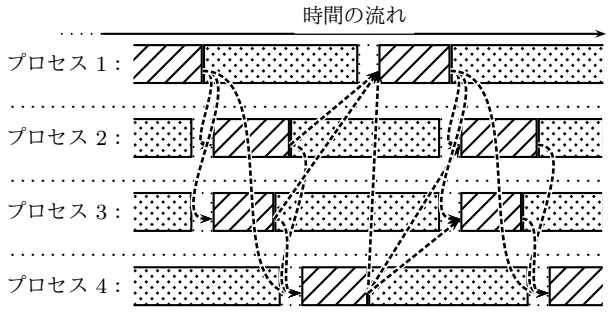

図 5 接続集合の並列化

Fig. 5 Parallelization of the computation of the connecting vertices

しかし，直接的な依存関係を持たないプロセスの組み合わせが必ずしも存在するとは限らず，またたとえそのような組み合わせが存在したとしてもその 2 個のプロセスが相前後する順番になっていなければこの事実は活用されない．例えば，直接的な依存関係がプロセス 2 と 3 の間には有ってプロセス 1 と 3 の間に無い場合は，プロセス 3 の接続集合の計算はプロセス 2 からの受信を待つが，プロセス 2 のそれはプロセス 1 からの受信を待つため，プロセス 1 と 3 の接続集合の計算は並列にはできず，依存関係がないという特徴を活用できない．

そこで，互いに依存関係を持たないプロセスの集合を形作り，その集合の順番でプロセスの計算順序を決定することで，接続集合の計算時間を削減する方法が考えられる．依存関係を持たないプロセスの集合を得るための手順としては，例えば Algorithm.1 が節点番号 $i = 1, \ldots, n$ をプロセス番号 $p = 1, \ldots, P$ に置き換えることで利用できる．この改善方法による 1 反復あたりの効率を改善前と併せて**表 4** に示す．

以上の結果から，特に効率が大きく低下している場合について改善が得られたことが確認できる．

7. まとめ

一般的に並列化が困難であるとされる SOR 法について，高い効率が得られる並列化手法を提案し，数値実験によってその性能を実証した．また，プロセス数を多くとった場合に発生する効率低下についても，プロセスの計算順序を制御することで抑制することができることを確認した．

今後の課題としては，共役勾配法に対する前処理としての実装や，提案手法に適したグラフ分割アルゴリズムの開発などを考えている．

表 4 1 反復あたりの計算時間による効率

Table 4 Efficiency by computational time per iteration

行列名	提案手法	$P = 4$	8	16	32
thermomech_TK	改善前	1.13	1.12	0.98	0.67
	改善後	1.14	1.11	0.93	0.72
shipsec8	改善前	0.94	0.71	0.54	0.32
	改善後	0.91	0.70	0.76	0.55
G2_circuit	改善前	1.20	1.05	0.65	0.26
	改善後	1.20	1.08	0.87	0.51
parabolic_fem	改善前	1.14	1.12	1.34	1.17
	改善後	1.14	1.18	1.37	1.12

参考文献

[1] Y. Saad: *Iterative methods for sparse linear systems*, SIAM, 2003.
[2] 関口達也：通信隠蔽手法を用いた SOR 法の並列化，平成 23 年度修士論文，秋田県立大学．
[3] T. Davis: The University of Florida Sparse Matrix Collection, 入手先 ⟨ http://www.cise.ufl.edu/research/sparse/matrices/⟩
[4] SCOTCH: Static Mapping, Graph, Mesh and Hypergraph Partitioning, and Parallel and Sequential Sparse Matrix Ordering Package, 入手先 ⟨http://www.labri.fr/perso/pelegrin/scotch/⟩