Jacobi-type Gauss-Seidel Smoother with Error Compensation for Parallel Multigrid ⋆

He Cangping [a,b,*], Zhao Yanmin [c], Tang Yifa [a]

[a] *LSEC, ICMSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China.*

[b] *Graduate University of Chinese Academy of Sciences, Beijing 100190, China.*

[c] *School of Mathematical Sciences, Xuchang University, Xuchang 461000, China.*

Abstract

Smoother is the most important component of parallel multigrid methods, however, the widely used Gauss-Seidel smoother is difficult to be parallelized. This paper proposes a new parallel smoother, Jacobi-type Gauss-Seidel with error compensation(JGSEC), by compensating the error caused by Jacobi-type Gauss-Seidel(JGS) smoother. Requiring no more inter-processor communication than JGS, JGSEC can eliminate the main part (more than ninety-five percent) of error arising from JGS at cost of a little more work amount. Numerical experiment has verified the good performance of JGSEC.

Key words: parallel multigrid, parallel Gauss-Seidel, JGS, JGSEC
1991 MSC: 65N55

1 Introduction

Smoother is the most important component of multigrid methods[3,5], and impacts the convergence rate and stability of multigrid. Gauss-Seidel(GS),

⋆ Supported by open project (No. 47549P0) of the State Key Laboratory of Scientific and Engineering Computing, Chinese Academy of Sciences, and by National Natural Science Foundation of China (Grant Nos. 10471145, 10672143 and 10872037), Natural Science Research Project of Henan Province (Grant No.2009A110017).

* Corresponding author.
Email addresses: cphe@lsec.cc.ac.cn (He Cangping), zhaoym@lsec.cc.ac.cn (Zhao Yanmin), tyf@lsec.cc.ac.cn (Tang Yifa).

which is a commonly used smoother, possesses many attractive properties such as good convergence rate and applicability on unstructured grids. However, GS has a crucial shortcoming: update of value of a grid point needs the most recent values of neighboring grid points. Thus, GS smoother is difficult to be parallelized.

As far as the authors know, the only one efficient parallel true Gauss-Seidel algorithm is *Parallel block multi-color Gauss-Seidel*, which is developed by Adams[1]. Unfortunately, this algorithm loses its simplicity on 3D unstructured grid as the number of required colors increases dramatically[2]. Many other parallel versions of Gauss-Seidel smoother have been given in [8–11,7].

Jacobi-type Gauss-Seidel(JGS) smoother[5] is also called as *Processor block Gauss-Seidel* in[6]. JGS applies the GS smoother only within a subgrid of a grid-partitioning application. As a consequence, the resulting iteration procedure is no longer a sequential GS iteration, but a combination of Jacobi-type iteration and GS[5]. Although JGS can be easily implemented on parallel computer systems, its convergence rate is worse than that of sequential GS, and PGS is even divergent under some special situation. An artificial example, on which PGS fails, is given in[2].

In this paper, it is found that the error on the boundary of subgrids cause the failure of JGS. The boundary error from initialization will propagate into interior of subgrids in the procedure of JGS iteration. Since GS scheme is a local relaxation, the boundary error attenuates rapidly in its propagation. Therefore, compensation of the largest components of error will make JGS approximate sequential GS well. That is the main idea of Jacobi-type Gauss-Seidel smoother with Error Compensation(JGSEC) algrorithm.

Compared with JGS, JGSEC needs no more communication and only a little more flops. Numerical experiment shows that JGSEC can reduce about eighty-five(ninety-five) percent of the error caused by JGS when compensating three(six) error terms.

The reminder of this paper is organized as follows. Section 2 introduces the sequential Gauss-Seidel iteration. Section 3 gives an introduction of JGS algorithm and points out its shortcoming. Section 4 is the central part of this paper. Here, JGSEC algorithm is obtained from the analysis of the boundary error propagation in JGS. Section 5 is a numerical experiment. Finally, the extension of JGSEC and future work are given in Section 6.

2 Sequential Gauss-Seidel iteration

Consider two-dimensional Poisson equation with Dirichlet conditions

$$\begin{aligned} -\triangle u(x,y) &= f(x,y), \quad (x,y) \in \Omega, \\ u(x,y) &= 0, \qquad\quad (x,y) \in \partial\Omega, \end{aligned} \tag{1}$$

where $\Omega = (0,1) \times (0,1)$. Given positive integer n and let $h = 1/n$, domain of the problem $\bar{\Omega} = [0,1] \times [0,1]$ is partitioned into $n \times n$ subsquares by introducing the grid points $(x_i, y_j) = ((i-1)h, (j-1)h)$.

Definition 1 *The* original grid *is denoted by* $\bar{\Omega}_h$ *and given by* $\bar{\Omega}_h = \{(x_i, y_j) | 1 \leq i, j \leq n+1\}$. *The* original interior grid *is denoted by* Ω_h *and given by* $\Omega_h = \{(x_i, y_j) | 2 \leq i, j \leq n\}$. *And the* original boundary grid *is denoted by* $\partial\Omega_h$ *and given by* $\partial\Omega_h = \bar{\Omega}_h \backslash \Omega_h = \{(x_i, h_j) | i, j = 1, n+1\}$.

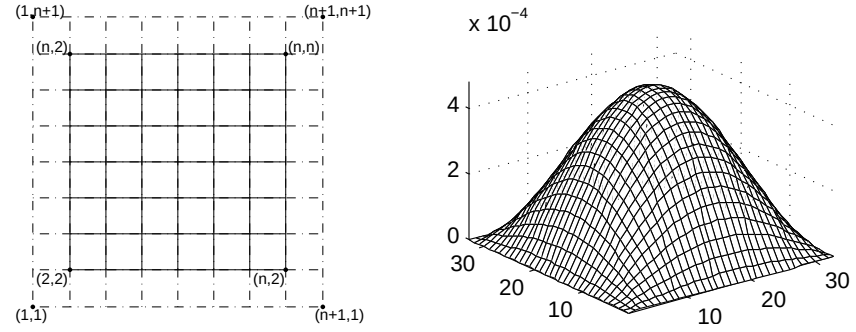

Fig. 1. Left: two-dimensional grids on the unit square. Points on the dashed lines are in original boundary grid $\partial\Omega_h$, while points on the solid lines are in original interior grid Ω_h. Right: v^1 on original grid $\bar{\Omega}_h$

These grids, Ω_h, $\bar{\Omega}_h$, and $\partial\Omega_h$, are shown in the left picture of Fig.1. The original grid $\bar{\Omega}_h$ contains all points in the picture. The original interior grid Ω_h contains all points on the solid lines, while the original boundary grid $\partial\Omega$ contains all points on the dashed lines.

At each of the $(n-1)^2$ interior grid points $(x_i, y_j) \in \Omega_h$, replace the original equation (1) with a second-order finite difference approximation. In making this replacement, v_{ij} is introduced as an approximation to the exact solution $u(x_i, y_j)$. Then the discrete Poisson equation is obtained:

$$\begin{aligned} (4v_{ij} - v_{i-1,j} - v_{i,j-1} - v_{i,j+1} - v_{i+1,j})/h^2 &= f_{ij}, \qquad (x_i, y_j) \in \Omega_h \\ v_{ij} &= 0, \qquad (x_i, y_j) \in \partial\Omega_h, \end{aligned} \tag{2}$$

where $f_{ij} = f(x_i, y_j)$.

Given initial values v_{ij}^0 for all points $(x_i, y_j) \in \Omega_h$, the sequential Gauss-Seidel iteration can be written in component form as

$$\begin{aligned} v_{ij}^{m+1} &= (h^2 f_{ij} + v_{i-1,j}^{m+1} + v_{i,j-1}^{m+1} + v_{i,j+1}^{m} + v_{i+1,j}^{m})/4, \quad (x_i, y_j) \in \Omega_h \\ v_{ij}^{m+1} &= v_{ij}^{m}, \quad (x_i, y_j) \in \partial\Omega_h, \end{aligned} \tag{3}$$

where $m = 0, 1, ...$

Discrete Poisson equation (2) implies that, for any boundary grid points $(x_i, y_j) \in \partial\Omega_h$, v_{ij}^m has the same value as v_{ij}^0, that is $v_{ij}^m \equiv v_{ij}^0 = 0$. Let set $v^m = \{v_{ij}^m | (x_i, y_j) \in \bar{\Omega}_h\}$ and denote GS iteration (3) by $v^{m+1} = \mathrm{GS}(v^m, f)$ for $m = 0, 1, ...$

Example 1. In Poisson equation (1), let $f(x, y) = \sin(\pi x)\sin(\pi y)$, and let $v_{ij}^0 = 0$, then v^1 is obtained by $v^1 = \mathrm{GS}(v^0, f)$(see the right picture of Fig. 1).

3 Jacobi-type Gauss-Seidel iteration

In order to implement GS iteration (3) on parallel computers with distributed memory, *grid partitioning* is a natural approach. In this approach, the original grid $\bar{\Omega}_h$ is split into P subgrids, such that P available processes can jointly solve the discrete Poisson eqution (2). Each subgrid is assigned to a different process such that each process is responsible for the computation in its part of the problem[5]. Each process not only stores the data belonging its subgrid but also a copy of data located in neighbor subgrids in a thin overlap area.

For simplicity, assuming $P = 4$, we split the original interior grid Ω_h into 2×2 subgrids to illustrate the idea of grid partitioning.

Definition 2 (subgrids) *subgrid 1:* $\Omega_h^1 = \{(x_i, y_j) | 2 \le i, j \le \frac{n}{2} + 1\}$, *subgrid 2:* $\Omega_h^2 = \{(x_i, y_j) | \frac{n}{2} + 2 \le i \le n, 2 \le j \le \frac{n}{2} + 1\}$, *subgrid 3:* $\Omega_h^3 = \{(x_i, y_j) | 2 \le i \le \frac{n}{2} + 1, \frac{n}{2} + 2 \le j \le n\}$, *subgrid 4:* $\Omega_h^4 = \{(x_i, y_j) | \frac{n}{2} + 2 \le i, j \le n\}$.

The definition of $\bar{\Omega}_h^k$ and $\partial\Omega_h^k (k = 1, 2, 3, 4)$ is similar to Definition 1. Obviously, $\cup_{k=1}^4 \Omega_h^k = \Omega_h$ and $\cup_{k=1}^4 \bar{\Omega}_h^k = \bar{\Omega}_h$, but $\cup_{k=1}^4 \partial\Omega_h^k \neq \partial\Omega_h$. When $n = 16$, Fig.2 illustrates the 4 subgrids. Grid points of the original boundary grid $\partial\Omega_h$ are marked with circles($\circ$), overlap points are marked with squares($\square$) and interior gird points of subgrids are marked with bullets($\bullet$).

The *left boundary* of Ω_h^2 is $\{(x_i, y_j) | i = \frac{n}{2} + 1, 1 \le j \le \frac{n}{2} + 2\}$, and the *bottom boundary* of Ω_h^3 is $\{(x_i, y_j) | 1 \le i \le \frac{n}{2} + 2, j = \frac{n}{2} + 1\}$. The *left boundary* and *bottom boundary* of Ω_h^4 can defined in the same way.

Using GS iteration (3), an update of unknowns depends on previously calcu-

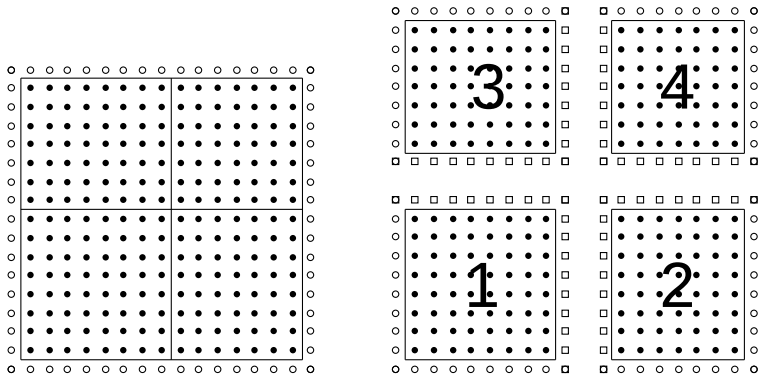

Fig. 2. Grid partitioning with $n = 16$. Left: original grid $\bar{\Omega}_h$. Grid points of the original boundary grid $\partial\Omega_h$ are marked with circles($\circ$). Right: Subgrids $\bar{\Omega}_h^k (k = 1, 2, 3, 4)$. Overlap points are marked with squares($\square$) and interior gird points of subgrids are marked with bullets($\bullet$).

lated values, so GS iteration is not suitable for parallel computing. A modification of GS, which better suits the grid partitioning concept is to apply the GS iteration only within a subgrid of $\bar{\Omega}_h^k$[5].

Let a_{ij}^m, b_{ij}^m,c_{ij}^m, d_{ij}^m be approximation of $u(x_i, y_j)$ on subgrids $\bar{\Omega}_h^1$, $\bar{\Omega}_h^2$, $\bar{\Omega}_h^3$, $\bar{\Omega}_h^4$ respectively, and let w_{ij}^m be approximation of $u(x_i, y_j)$ on original grid $\bar{\Omega}_h$. Their initial values are given as

$$a_{ij}^0 = v_{ij}^0,\ b_{ij}^0 = v_{ij}^0,\ c_{ij}^0 = v_{ij}^0,\ d_{ij}^0 = v_{ij}^0. \tag{4}$$

Definition 3 *Define sets a^m, b^m, c^m, c^m, w^m as follows:*

$$\begin{aligned}
a^m &= \{a_{ij}^m | (x_i, y_j) \in \bar{\Omega}_h^1\}, \quad b^m = \{b_{ij}^m | (x_i, y_j) \in \bar{\Omega}_h^2\}, \\
c^m &= \{c_{ij}^m | (x_i, y_j) \in \bar{\Omega}_h^3\}, \quad d^m = \{d_{ij}^m | (x_i, y_j) \in \bar{\Omega}_h^4\}, \\
w^m &= \{w_{ij}^m\} = \{a_{ij}^m | (x_i, y_j) \in \Omega_h^1\} \cup \{b_{ij}^m | (x_i, y_j) \in \Omega_h^2\} \cup \{c_{ij}^m | (x_i, y_j) \in \Omega_h^3\} \\
&\quad \cup \{d_{ij}^m | (x_i, y_j) \in \Omega_h^4\} \cup \{v_{ij}^m | (x_i, y_j) \in \partial\Omega_h\}, m = 0, 1, 2, ...
\end{aligned}$$

It is obvious that $w^0 \equiv v^0$.

The procedure of JGS iteration is described as following. Starting with some approximation w^m on these subgrids, each process computes a new GS iterate w^{m+1} for each point in the interior of its subgrid, then each process exchanges data with its neighboring subgrids.

Concretely, four processes compute a new GS iterate w^1 from initial approximation w^0 on the four subgrids $\bar{\Omega}_h^k (k = 1, 2, 3, 4)$ simultaneously, then exchange data between subgrid $\bar{\Omega}_h^1$ and $\bar{\Omega}_h^2$, between subgrid $\bar{\Omega}_h^3$ and $\bar{\Omega}_h^4$, between subgrid $\bar{\Omega}_h^1$ and $\bar{\Omega}_h^3$, between subgrid $\bar{\Omega}_h^2$ and $\bar{\Omega}_h^4$. The following JGS algorithm

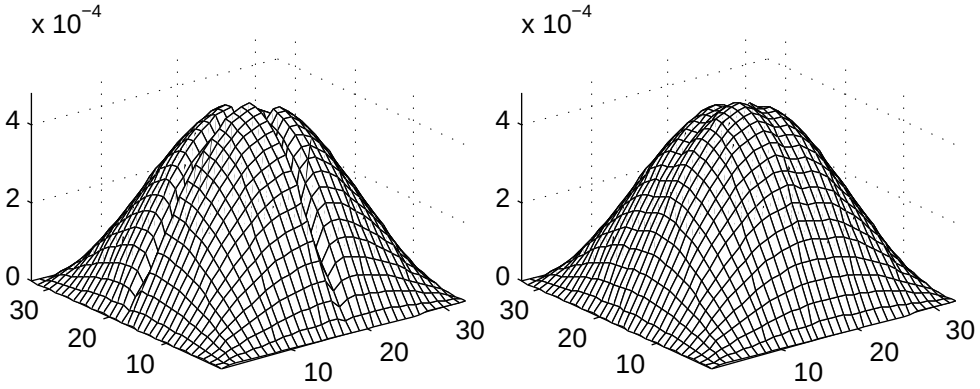

Fig. 3. Left: w^1 resulted from JGS. Right: w^1 resulted from JGSEC.

is written in sequential pseudo code, which must be implemented by parallel code in practice.

Algorithm 1: Jacobi-type Gauss-Seidel
$\forall (x_i, y_j) \in \bar{\Omega}_h^1, a_{ij}^0 = v_{ij}^0$; $\forall (x_i, y_j) \in \bar{\Omega}_h^2, b_{ij}^0 = v_{ij}^0$;
$\forall (x_i, y_j) \in \bar{\Omega}_h^3, c_{ij}^0 = v_{ij}^0$; $\forall (x_i, y_j) \in \bar{\Omega}_h^4, d_{ij}^0 = v_{ij}^0$;
`for m = 0:1`

$$\begin{aligned} a^{m+1} &= \mathrm{GS}(a^m, f); b^{m+1} = \mathrm{GS}(b^m, f); \\ c^{m+1} &= \mathrm{GS}(c^m, f); a^{m+1} = \mathrm{GS}(a^m, f); \end{aligned} \tag{5}$$

$$\begin{aligned} &a_{\frac{n}{2}+2,j}^{m+1} = b_{\frac{n}{2}+2,j}^{m+1},\ b_{\frac{n}{2}+1,j}^{m+1} = a_{\frac{n}{2}+1,j}^{m+1},\ j = 2, 3, .., \tfrac{n}{2}+2. \\ &a_{i,\frac{n}{2}+2}^{m+1} = c_{i,\frac{n}{2}+2}^{m+1},\ c_{i,\frac{n}{2}+1}^{m+1} = a_{i,\frac{n}{2}+1}^{m+1},\ i = 2, 3, .., \tfrac{n}{2}+2. \\ &b_{i,\frac{n}{2}+2}^{m+1} = d_{i,\frac{n}{2}+2}^{m+1},\ d_{i,\frac{n}{2}+1}^{m+1} = b_{i,\frac{n}{2}+1}^{m+1},\ i = \tfrac{n}{2}+2, ..., n. \\ &c_{\frac{n}{2}+2,j}^{m+1} = d_{\frac{n}{2}+2,j}^{m+1},\ d_{\frac{n}{2}+1,j}^{m+1} = c_{\frac{n}{2}+1,j}^{m+1},\ j = \tfrac{n}{2}+2, ..., n. \end{aligned}$$

`end`

JGS algorithm implies that $w^m \equiv w^0 \equiv v^0$ on $\partial\Omega_h$, but $w^m \not\equiv v^m$ on Ω_h. So JGS is not algorithmically equivalent to sequential GS(3). In multigrid method, only one or two JGS iterations is needed on each grid level, thus the range of variable `m` in JGS algorithm is `0:1`. The left picture of Fig. 3 is the imagine of w^1 on original grid $\bar{\Omega}_h$. Comparing this picture with the right picture of Fig. 1, one can see that w^1 approximates v^1 well on the interior points of subgrids, but the approximation is bad on the points near the interface of subgrids.

Define the error of a parallel iteration as $e^m = w^m - v^m$. Imagine of error e^1 of JGS algorithm(the left picture of Fig. 4) also shows that the main part of error is located at grid points near subgrid boundary. So the *norm* of error e^m can be defined as following:

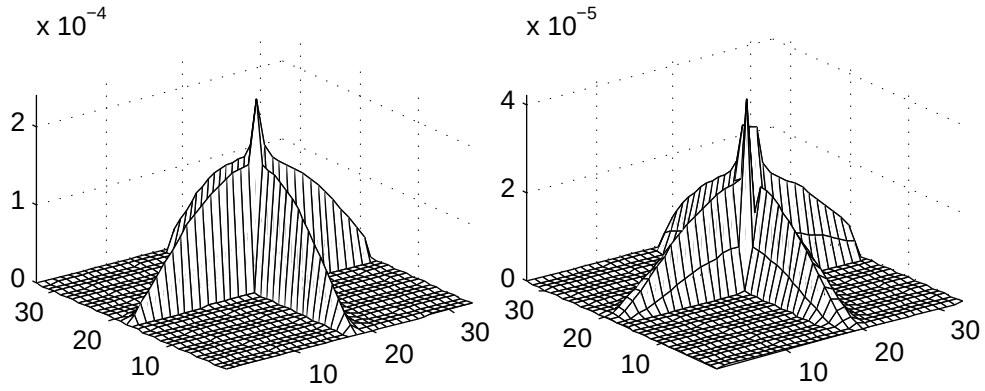

Fig. 4. Left: error $-e^1$ of JGS. Right: error $-e^1$ of JGSEC.

$$\|e^m\| = \frac{1}{4n-8}\sum_{j=2}^{n}\left(|e^m_{\frac{n}{2}+2,j}| + |e^m_{\frac{n}{2}+3,j}|\right) + \frac{1}{4n-8}\sum_{i=2}^{\frac{n}{2}+1}\left(|e^m_{i,\frac{n}{2}+2}| + |e^m_{i,\frac{n}{2}+3}|\right) + \frac{1}{4n-8}\sum_{i=\frac{n}{2}+4}^{n}\left(|e^m_{i,\frac{n}{2}+2}| + |e^m_{i,\frac{n}{2}+3}|\right).$$

For JGS, $\|e^1\|$=7.17e-3 when $n = 32$.

4 Jacobi-type Gauss-Seidel with error compensation

In order to overcome the shortcoming of JGS, we have to answer two question2. The first question is what cause the error e^1? For subgrid 1, $a^1_{ij} \equiv v^1_{ij}$ on Ω^1_{ij} since $a^0_{ij} \equiv v^0_{ij}$. For subgrid 2, if $b^1_{ij} \equiv v^0_{ij}$ on the left boundary of Ω^2_h, then $b^1_{ij} \equiv v^1_{ij}$ on $\bar{\Omega}^2_h$. Similarly, if $c^1_{ij} \equiv v^0_{ij}$ on the bottom boundary of Ω^3_h, then $c^1_{ij} \equiv v^1_{ij}$ on $\bar{\Omega}^3_h$; if $d^1_{ij} \equiv v^0_{ij}$ on the left boundary and bottom boundary of Ω^4_h, then $d^1_{ij} \equiv v^1_{ij}$ on $\bar{\Omega}^4_h$. In JGS algorithm, however, GS iterations on the four subgrids take place at the same time, so $b^1_{ij} \not\equiv b^0_{ij} \equiv v^0_{ij}$ on left boundary of Ω^2_h, $c^1_{ij} \not\equiv c^0_{ij} \equiv v^0_{ij}$ on the bottom boundary Ω^3_h and $d^1_{ij} \not\equiv d^0_{ij} \equiv v^0_{ij}$ on the left boundary and the bottom boundary of Ω^4_h.

Definition 4 *The* boundary error *is denoted by* $\tilde{e}$ *and given by* $\tilde{e} = \{\tilde{e}_{ij}\} = \{b^0_{ij} - v^1_{ij} | \textit{on the left boundary of } \Omega^2_h\} \cup \{c^0_{ij} - v^1_{ij} | \textit{on the bottom boundary of } \Omega^3_h \cup \{d^0_{ij} - v^1_{ij} | \textit{on the left and bottom boundary of } \Omega^4_h\}$.

It is clear that boundary error causes error e^1 on Ω_h.

The second equation is how boundary error propagates from the boundary subgrid points to the interior subgrid points? Take subgrid 2 for example. The GS iteration (3) implies that

$$v^1_{ij} = (h^2 f_{ij} + v^1_{i-1,j} + v^1_{i,j-1} + v^0_{i,j+1} + v^0_{i+1,j})/4, \quad (x_i, y_j) \in \Omega_h, \qquad (6)$$

and JGS algorithm implies that

$$b^1_{ij} = (h^2 f_{ij} + b^1_{i-1,j} + b^1_{i,j-1} + b^0_{i,j+1} + b^0_{i+1,j})/4, \quad (x_i, y_j) \in \Omega^2_h. \tag{7}$$

(7) subtracted from (6) gives the *error propagation formula*

$$\begin{aligned} e^1_{ij} &= b^1_{ij} - v^1_{ij} \\ &= \frac{1}{4}[(b^1_{i-1,j} - v^1_{i-1,j}) + (b^1_{i,j-1} - v^1_{i,j-1}) + (b^0_{i,j+1} - v^0_{i,j+1}) + (b^0_{i+1,j} - v^0_{i+1,j})] \\ &\overset{(4)}{=} \frac{1}{4}[(b^1_{i-1,j} - v^1_{i-1,j}) + (b^1_{i,j-1} - v^1_{i,j-1})], \quad (x_i, y_j) \in \Omega^2_h. \end{aligned} \tag{8}$$

In order to illustrate the propagation of boundary error, assume $n = 12$ and

$$\tilde{e}_{73} = 1, \qquad \tilde{e}_{7,j} = 0, \quad j = 1, 2, 4, ..., 8. \tag{9}$$

Equations (5),(8) and (9) give $e^1_{i,2} = 0$, $i = 8, 9, ..., 12$, and

$$\begin{aligned} e^1_{83} &= \frac{1}{4}\left[(b^1_{73} - v^1_{73}) + (b^1_{82} - v^1_{82})\right] = \frac{1}{4}(b^0_{71} - v^1_{71}) = \frac{1}{4}\tilde{e}_{73} = \frac{1}{4}, \\ e^1_{93} &= \frac{1}{4}\left[(b^1_{83} - v^1_{83}) + (b^1_{92} - v^1_{92})\right] = \frac{1}{4}[e^1_{83} + e^1_{92}] = \frac{1}{16}, \\ e^1_{84} &= \frac{1}{4}\left[(b^1_{74} - v^1_{74}) + (b^1_{83} - v^1_{83})\right] = \frac{1}{4}[\tilde{e}_{74} + e^1_{83}] = \frac{1}{16}, \\ e^1_{10,3} &= \frac{1}{64},\ e^1_{94} = \frac{2}{64},\ e^1_{85} = \frac{1}{64},\ e^1_{11,4} = \frac{1}{256} \cdots . \end{aligned} \tag{10}$$

The propagation procedure on subgrid 2 is shown in the left picture of Fig. 5. If let

$$\bar{b}^1_{83} = b^1_{83} + \frac{1}{4}\tilde{e}_{73},\ \bar{b}^1_{93} = b^1_{93} + \frac{1}{16}\tilde{e}_{73},\ \bar{b}^1_{84} = b^1_{84} + \frac{1}{16}\tilde{e}_{73},$$

then the three largest error terms are compensated:

$$e^1_{83} = \bar{b}^1_{83} - v^1_{83} = 0,\ e^1_{93} = \bar{b}^1_{93} - v^1_{93} = 0,\ e^1_{84} = \bar{b}^1_{84} - v^1_{84} = 0.$$

Error compensation on subgrid 3 and subgrid 4 are similar(see the right picture of Fig. 5). Following this compensation idea is the algorithm of JGSEC. The following JGSEC algorithm is written in sequential pseudo code, which must be rewritten in parallel code in practice.

Algorithm 2: Jacobi-type Gauss-Seidel with Error Compensation
$\forall (x_i, y_j) \in \bar{\Omega}^1_h, a^0_{ij} = v^0_{ij}$; $\forall (x_i, y_j) \in \bar{\Omega}^2_h, b^0_{ij} = v^0_{ij}$;
$\forall (x_i, y_j) \in \bar{\Omega}^3_h, c^0_{ij} = v^0_{ij}$; $\forall (x_i, y_j) \in \bar{\Omega}^4_h, d^0_{ij} = v^0_{ij}$;

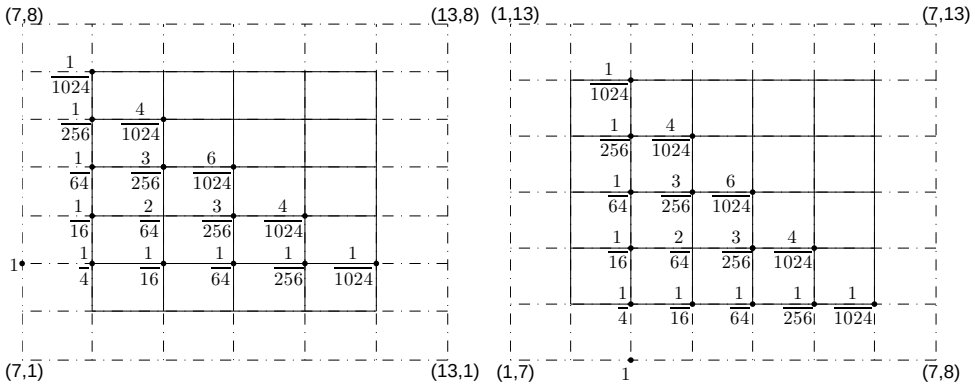

Fig. 5. Error propagation from boundary to interior. Left: subgrid 2, right: subgrid 3.

```
for m = 0:1
```

$a^{m+1} = \mathrm{GS}(a^m, f)$; $b^{m+1} = \mathrm{GS}(b^m, f)$; $c^{m+1} = \mathrm{GS}(c^m, f)$; $d^{m+1} = \mathrm{GS}(d^m, f)$;

```
    for j = 2 : n/2+1
```

$$b^{m+1}_{\frac{n}{2}+2,j} = b^{m+1}_{\frac{n}{2}+2,j} - \tfrac{1}{4}(b^{m+1}_{\frac{n}{2}+1,j} - a^{m+1}_{\frac{n}{2}+1,j}) - \tfrac{1}{16}(b^{m+1}_{\frac{n}{2}+1,j-1} - a^{m+1}_{\frac{n}{2}+1,j-1});$$
$$b^{m+1}_{\frac{n}{2}+3,j} = b^{m+1}_{\frac{n}{2}+3,j} - \tfrac{1}{16}(b^{m+1}_{\frac{n}{2}+1,j} - a^{m+1}_{\frac{n}{2}+1,j});$$
$$b^{m+1}_{\frac{n}{2}+1,j} = a^{m+1}_{\frac{n}{2}+1,j}; \quad a^{m+1}_{\frac{n}{2}+2,j} = b^{m+1}_{\frac{n}{2}+2,j};$$

```
    end
    for j = n/2+2 : n
```

$$d^{m+1}_{\frac{n}{2}+2,j} = d^{m+1}_{\frac{n}{2}+2,j} - \tfrac{1}{4}(d^{m+1}_{\frac{n}{2}+1,j} - c^{m+1}_{\frac{n}{2}+1,j}) - \tfrac{1}{16}(d^{m+1}_{\frac{n}{2}+1,j-1} - c^{m+1}_{\frac{n}{2}+1,j-1});$$
$$d^{m+1}_{\frac{n}{2}+3,j} = d^{m+1}_{\frac{n}{2}+3,j} - \tfrac{1}{16}(d^{m+1}_{\frac{n}{2}+1,j} - c^{m+1}_{\frac{n}{2}+1,j});$$
$$d^{m+1}_{\frac{n}{2}+1,j} = c^{m+1}_{\frac{n}{2}+1,j}; \quad c^{m+1}_{\frac{n}{2}+2,j} = d^{m+1}_{\frac{n}{2}+2,j};$$

```
    end
    for i = 2 : n/2+1
```

$$c^{m+1}_{i,\frac{n}{2}+2} = c^{m+1}_{i,\frac{n}{2}+2} - \tfrac{1}{4}(c^{m+1}_{i,\frac{n}{2}+1} - a^{m+1}_{i,\frac{n}{2}+1}) - \tfrac{1}{16}(c^{m+1}_{i-1,\frac{n}{2}+1} - a^{m+1}_{i-i,\frac{n}{2}+1});$$
$$c^{m+1}_{i,\frac{n}{2}+3} = c^{m+1}_{i,\frac{n}{2}+3} - \tfrac{1}{16}(c^{m+1}_{i,\frac{n}{2}+1} - a^{m+1}_{i,\frac{n}{2}+1});$$
$$c^{m+1}_{i,\frac{n}{2}+1} = a^{m+1}_{i,\frac{n}{2}+1}; \quad a^{m+1}_{i,\frac{n}{2}+2} = c^{m+1}_{i,\frac{n}{2}+2};$$

```
    end
    for i = n/2+2 : n
```

$$d^{m+1}_{i,\frac{n}{2}+2} = d^{m+1}_{i,\frac{n}{2}+2} - \tfrac{1}{4}(d^{m+1}_{i,\frac{n}{2}+1} - b^{m+1}_{i,\frac{n}{2}+1}) - \tfrac{1}{16}(d^{m+1}_{i-1,\frac{n}{2}+1} - b^{m+1}_{i-1,\frac{n}{2}+1});$$
$$d^{m+1}_{i,\frac{n}{2}+3} = d^{m+1}_{i,\frac{n}{2}+3} - \tfrac{1}{16}(d^{m+1}_{i,\frac{n}{2}+1} - b^{m+1}_{j,\frac{n}{2}+1});$$
$$d^{m+1}_{i,\frac{n}{2}+1} = b^{m+1}_{i,\frac{n}{2}+1}; \quad b^{m+1}_{i,\frac{n}{2}+2} = d^{m+1}_{i,\frac{n}{2}+2};$$

```
    end
end
```

The w^1 is obtained after two JGSEC iterations. From Fig. 3, one can see that w^1 resulting from JGSEC is smoother than w^1 resulting from JGS. Comparing the two error pictures in Fig. 4, one can get an intuitive conclusion that error of JGSEC is much smaller than error of JGS. For JGSEC, $\|e^1\| = 1.16\text{e-}5$ when $n = 32$.

Algorithm 2 only compensates the three largest error terms, it can be easily modified to compensate the six or nine largest errror terms.

Work amount: analysis of algorithm 1 and algorithm 2 shown that JGSEC needs no more inter-processes communication than JGS. If define N as the number of subgrid points of the set

$$\cup_{k=1}^{P}\{\textit{left boundary and bottom boundary of } \bar{\Omega}_h^k\}\backslash\bar{\Omega}_h,$$

then $N = 2n-2$ for 2×2 grid partitioning. One JGSEC sweep needs $3N(6N)$ more addition operations and $2N(4N)$ more multiplication operations when the three(six) largest error terms are compensated.

5 Numerical Experiment

Example 2. Let $n = 256$ and let $f(x,y) = \sin(k\pi x)\sin(l\pi y)$ in the Poisson equation (1), where $k, l = 1, 2, ..., n-1$. With initial value $v_{ij}^0 = 0$, the error norms of JGS and JGSEC are illustrated in Fig. 6. If the ratio of $\|e^1\|^{\mathrm{JGSEC}}$ to $\|e^1\|^{\mathrm{JGS}}$ is denoted by r and given by $r = \|e^1\|^{\mathrm{JGSEC}}/\|e^1\|^{\mathrm{JGS}}$, the left(right) picture of Fig. 7 shows that the ratio is less than 0.158(0.052) when the three(six) largest error terms are compensated. The two pictures demonstrate that JGSEC is much better than JGS.

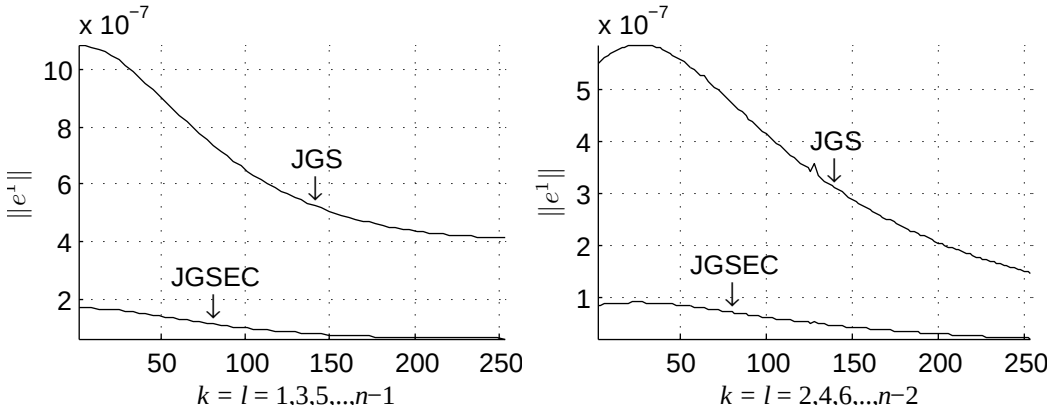

Fig. 6. Comparison of error norm on 2×2 subgrids.

6 Extension and future work

It easy to get corresponding JGSEC algorithm for nine-point stencil. The JGSEC algorithm 2 can be used on three-dimensional structured and unstruc-

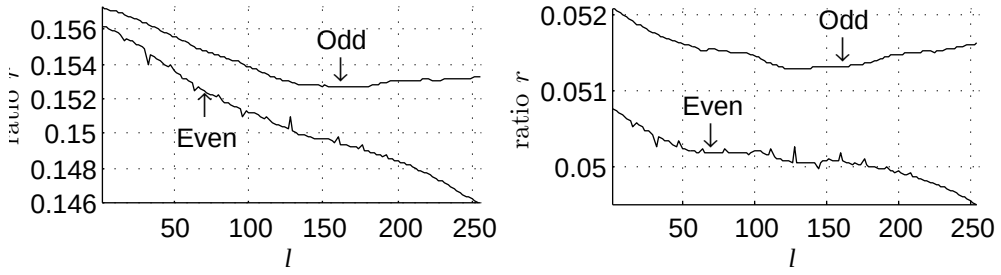

Fig. 7. Ratio of $\|e^1\|^{\mathrm{JGSEC}}$ to $\|e^1\|^{\mathrm{JGS}}$ on 2×2 subgrids. Left: the three largest error terms have been compensated, right: the six largest error terms have been compensated.

tured grids. The idea of compensation can be used to obtain corresponding parallel versions of SOR and SSOR smoothers. For example, SOR smoother

$$v_{ij}^{m+1} = \frac{\omega}{4}(h^2 f_{ij} + v_{i-1,j}^{m+1} + v_{i,j-1}^{m+1} + v_{i,j+1}^{m} + v_{i+1,j}^{m}) + (1-\omega)v_{ij}^{m}$$

has its own *error propagation formula* as

$$e_{ij}^1 = \frac{\omega}{4}[(b_{i-1,j}^1 - v_{i-1,j}^1) + (b_{i,j-1}^1 - v_{i,j-1}^1)], \quad (x_i, y_j) \in \Omega_h^2.$$

References

[1] Mark F. Adams, *A Distributed Memory Unstructured Gauss-Seidel Algrorithm for Multigrid Smoothers*, in: ACM/IEEE Proceedings of SC01: High Performance Networking and Computing, 2001.

[2] Mark Adams, Marian Brezina, Jonathan Hu, Ray Tuminaro, *Parallel multigrid smoothing: polynomial versus Gauss-Seidel* , Journal of Computational Physics 188(2003) 593-610.

[3] William L. Briggs, Van Emden Henson, Steve F. McCormick, A Multigrid Tutorial, SIAM 2000.

[4] L. R. Scott and Dexuan Xie, *Parallel Linear Stationary Iterative Methods*, Math. Comp. (1995)

[5] U. Trottenberg, C.W. Oosterlee, A. Shüller, *Multigrid*, Academic Press, 2001.

[6] Van Emden Henson, Ulrike Meier Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Technical Report UCRL-JC- 139098, Lawrence Livermore National Laboratory, 2000.

[7] Dexuan Xie, *New parallel symmttric SOR preconditioners by mulit-type partitioning*, International Journal of Computer Mathematics, Vol.86, No. 2, February 2009, 287-300.

[8] D. Xie and L. Adams, *New Parallel SOR Method by Domain Partitioning*, SIAM J. on Scientific Computing, Vol. 20, No. 6, pp. 2261-2281, 1999.

[9] D. Xie, *Analysis of a class of parallel multigrid smoothers*, BIT Numerical Mathematics, Vol. 44, pp.813-828, 2004.

[10] D. Xie, *A new block parallel SOR method and its analysis*, SIAM J. Sci. Comput., Vol. 27, No. 5, pp. 1513-1533, 2006.

[11] D. Xie, *New parallel symmetric SOR preconditioners by multi-type partitioning*, Journal of Computer Mathematics, Vol. 86, No. 2, pages 287-300, 2009.