An Operator Smoothing with ILU(0) for Aggregation-based Algebraic Multigrid

Jian-ping WU* and Liang SUN
School of Meteorology and Oceanography, National University of Defense Technology, Changsha, China
*Corresponding author

Keywords: Incomplete factorization, Algebraic multigrid, Sparse linear system, Conjugate gradient iteration, Preconditioner.

Abstract. Due to the potential optimal convergence, algebraic multigrid is widely used in solving large sparse linear systems. The aggregation-based version is one of the widely used methods for its cheap cost and easy implementation. But its convergence is often slow compared to other versions. In this paper, a smoothing technique based on incomplete LU factorization without fill-in is presented to the operators on each level. Each operator is approximately factorized, and the derived lower and upper triangular factors are approximately inverted, which are applied to the operator from both sides to improve the diagonal dominance, and then the effectiveness of the smoothing process and the accuracy of the grid transfer operators. The numerical results show that when incorporated into the preconditioned conjugate gradient iteration, the convergence rate is greatly improved, and though the time used for setup is larger, the time used for iteration and the overall time can also be reduced for large-scale systems.

Introduction

Solution of sparse linear systems is the most time-consuming part in many scientific and engineering fields, and with the increasing demand in high resolution simulation, the derived systems are becoming larger and larger. To solve this kind of system efficiently, the multigrid like methods are often used, either as iterations solely, or as preconditioners for Krylov subspace iterations [1].

The effectiveness of multigrid is from the coupling of two processes, the smoothing and the coarse grid correction. The smoothing reduces the error with lower frequencies and the residual corresponding to the remaining error with higher frequencies is transferred to a coarser grid. With the derived residual as the right-hand side, the coarse grid system is solved and its solution is prolonged back to the finer grid, to correct the solution of the original system [2].

Due to potential optimal convergence, multigrid methods attract more and more attention in recent years. Algebraic multigrid methods can achieve similar performance as the traditional geometry versions, without the requirements to know the physical grid or the coordinates in advance, which can be used to solve general sparse linear systems. According to the coarsening process and the construction of the grid transfer operators, algebraic multigrid can be classified into classical and aggregation-based version [2] [3], where the aggregation-based version attracts more attention for its cheap cost and easy implementation. For more details about aggregation-based multigrid methods, we can refer to [4-11].

Though both the setup and the application of aggregation based algebraic multigrid methods are cost-inexpensive, their convergence rates are always slow compared to the classical versions. It is well known that the convergence rate is closely related to the performance of the smoother and the grid transfer operators. The stronger of diagonal dominance of the coefficient matrices, the more efficient of the smoothing [12], and the easier to construct efficient transfer operators too [3]. Based on this consideration, in this paper, we will present a new scheme based on incomplete factorization without fill-in to improve the diagonal dominance of coefficient matrices on each level. In this
scheme, each coefficient matrix is approximately factorized, and the derived lower and upper triangular factors are approximately inverted, which are applied to the coefficient matrix from both sides.

**Operator Smoothing with ILU(0) for Aggregation-Based Multigrid Method**

**Aggregation-Based Multigrid Method**

Consider the linear system $Ax=b$, where $A$ is an $n$ by $n$ positive definite matrix, and $b$ is a vector of dimension $n$. General multigrid with V-cycle can be described as figure 1[3].

In algorithm MG, matrix $S_k$ is a smoothing operator on level $k$, and $S_k^T$ is the adjoint of $S_k$, which assures the positive definiteness of the derived multigrid preconditioner, to apply the preconditioned conjugate gradient iterations. $P_{k,k+1}$ is an interpolation operator, to transfer a vector on the $k$-th level to the $(k+1)$-th level, and $R_{k+1,k}$ is a restriction operator, to transfer a vector on the $(k+1)$-th level to the $k$-th level. In general, we adopt $R_{k+1,k}=P_{k,k+1}^T$ for symmetric positive cases and $A_{k+1}=R_{k+1,k} A_k P_{k,k+1}$.

Before applying MG to solve a sparse linear system, the smoother $S_k$, the interpolation operator $P_{k,k+1}$ and a coarsing scheme should be given, and the coarse grid operator $A_{k+1}$ should be derived at the setup first. Any kind of simple iteration can be selected as a smoother. And once the interpolation operator is given, the coarse grid transfer operator can be computed from the finer grid operator. Therefore, the classic algebraic multigrid and aggregation based versions differ mainly in the construction of the grid transfer operator $P_{k,k+1}$, and the coarsing scheme.

Assume that matrix $A_k$ is symmetric positive definite, it is corresponding to an undirected graph $G_k=(V_k, E_k)$, where each vertex in $V_k$ relates to a row/column of $A_k$, and each edge in $E_k$ relates to a nonzero element of $A_k$. In aggregation based multigrid algorithm, the vetexes in $V_k$ are classified into different aggregates, with each aggregate corresponding to a vetex in the coaser grid. The classic scheme to construct the transfer operator is to set $P_{k,k+1}(i,j)=1$ if vertex $i$ belongs to the $j$-th aggregate, and $P_{k,k+1}(i,j)=0$ otherwise.

**Operator Smoothing with ILU(0)**

To investigate the convergence of multigrid, we usually consider the simplified case of two-level grids, where the iteration matrix can be written as[7]

$$T_k = (I - S_k A_k)(I - P_{k,k+1} A_k^{-1} P_{k,k+1}^T A_k)(I - S_k^T A_k).$$

(1)

Therefore, the convergence rate is influenced by two factors. Firstly, the more accurate $S_k$ approximates the inverse of $A_k$, the more rapid the convergence rate will be. It is well known that the more diagonally dominant the matrix $A_k$ is, the easier we construct related efficient smoother, that is, $S_k A_k\approx I$ is more accurate. Then, the corresponding two-grid iteration will converge more rapidly.

Secondly, we know that when multigrid is used to solve partial differential equations, the stronger the anisotropy is, the less efficient the solver will be. From the point of discrete solution, anisotropy
relates to differences of off-diagonal elements in the coefficient matrix of the sparse linear system. If the off-diagonal non-zeros are not far different from each other, the more convenient for us to construct an efficient interpolation operator, which is crucial to rapid convergce of the derived multigrid method.

With the above considerations, it is clear that if we can transform \( A_kx_k = b_k \) to another linear system \( A_{k+1}x_{k+1} = b_{k+1} \) with \( A_{k+1} \) approaches the unity \( I \) and off-diagonals almost the same in magnitude, the modified multigrid will be more efficient.

In this paper, it is assumed that \( A_k \) is symmetric positive and ILU(0)[1][12] is used for the above transformation, then, we have \( A_kL_kU_k \) with \( U_k U_k^T \), and \( L_k A_k L_k^T \approx I \). In algebraic multigrid method, the coefficient matrix on each level should be given explicitly, thus, we should approximate constructing an efficient interpolation operator, which is crucial to rapid convergce of the derived multigrid method.

Now, if we denote \( L_k \approx L_k^{-1} \), and \( A_k = L_k A_k L_k^T \), we can apply MG algorithm to \( A_k \) recursively, and the modified multigrid algorithm can be described as figure 2, where \( S_k, R_k, \) and \( P_k \) are similarly defined as \( S_k, R_k, \) and \( P_k \) respectively, and are all derived from \( A_k \).

From figure 2, we can see that once \( L_k \) is derived, the computation can be performed similar to figure 1. The only added operations are the application of \( L_k \) and \( L_k^T \) to a vector respectively.

![Figure 2. Modified V-cycle multigrid \( x^{(k)} = \text{ILUMG}(x^{(k)}, b^{(k)}, k) \).](image)

### Some Implementation Details in Setup

Now we investigate the computation of \( L_k \) and \( A_k \). For \( L_k \) is from ILU(0) of \( A_k \), it is suitable to compute \( L_k \) from \( L_0 \) with the same non-zero structure. Then, if there is a nonzero \( \ell_0(i,j) \) in \( L_k \), the corresponding element in \( L_0 \) is given by

\[
L_0(i,j) = \begin{cases} \ell_0(i,i), & i = j, \\ -\ell_0(i,j) \ell_0(j,i)/\ell_0(j,j), & i > j. \end{cases}
\]

(2)

Assume that matrix \( L_k \) is of order \( n_k \) and it is stored in MSR format[12] to vectors \( \ell' \) and \( \ell \), where the first \( n_k \) elements of \( \ell' \) are the diagonals one by one, the remaining off-diagonal non-zeros are followed row by row. When \( i \) is not larger than \( n_{k+1} \), \( \ell'(i) \) record the position of the first element in the \( i \)-th row in vector \( \ell \). Otherwise, it record the column index of the element \( \ell(i) \). In addition, we store \( L \) in CSR format[12] to the vectors \( \ell', \ell' \), and \( \ell \) with the non-zeros stored in \( \ell \) row by row, the column index of \( \ell(k) \) stored in \( \ell'(k) \), the position of the first element in the \( i \)-th row in vector \( \ell \) denoted as \( \ell'(i) \), then the computation is very direct and efficient, and only one traversal of the related vectors is required. The algorithm can be described in detail as figure 3.

To compute \( A_k \), the kernel is the multiplication of two sparse matrices. Assume that it is needed to compute \( S = AW \). Since the \( i \)-the row of \( S \) can be computed as

\[
s(i,*) = \sum_{j=1}^{n} a(i,j)w(j,*) = \sum_{j \in \text{Adj}(A,i)} a(i,j)w(j,*) ,
\]

(3)
where \( \text{adj}(A,i) \) denotes the column indices of the non-zeros in the \( i \)-th row of matrix \( A \). Then, we can compute the vector \( s(i,*\) with sparse data structures.

If matrix \( A \) is of order \( n \), we store it in CSR format to the vectors \( a', a', \) and \( a \), with the non-zeros stored in a row by row, the column index of \( a(k) \) stored in \( a'(k) \), the position of the first element in the \( i \)-th row in vector \( a \) denoted as \( a'(i) \), and matrix \( W \) is stored in the vectors \( w', w' \), and \( w \) similarly, the algorithm to compute \( S=AW \) can be described as figure 4. In figure 4, vector \( t \) is used to identify whether an element in the current row is already generated. If \( S_{ij} \) is generated, it is updated latter when needed. On output, the result matrix \( S \) is stored in vectors \( s', s' \), and \( s \).

```
label=1
for i=1, n_k
   f'(i)=label
   for j=f'(i), f'(i+1)-2
      f'(label)=f'(j)
      l(label)=l(j)/l(f'(j))/l(i)
      label=label+1
   endfor
   f'(n_k+1)=label
```

**Figure 3.** Compute ILU(0) inverse of a lower triangular matrix.

```
t(1:n)=0
label=1
for i=1, n
   s'(i)=label; n_i=0
   for k=a'(i), a'(i+1)-1
      for j=w'(a'(k)), w'(a'(k)+1)-1
         if(t(w'(j))=0 then
            v(t(w'(j)))= v(t(w'(j)))+a(k)*w(j)
         else
            n_i = n_i + 1
            v(n_i)= a(k)*w(j)
            v'(n_i)= w'(j)
            t(w'(j))= n_i
         endif
      endfor
   endfor
   for j=1, n,
      s(label)=v(j)
      s'(label)= v'(j)
      t(v'(j))=0
      label=label+1
   endfor
   s'(n+1)=label
```

**Figure 4.** Sparse matrix multiplication \( S=AW \) in CSR format.

For \( A_k=L_kA_kL_k^T \), it can be computed as \( B_k=L_kA_k \) first, and then \( A_k=B_kL_k^T \), where \( L_k^T \) can be stored as \( U_k \) in CSR format, which is transformed from the CSR format of \( L_k \) with sparse data structures again.

It is clear that we can apply any incomplete factorizations, but we adopt ILU(0) here for two reasons. First when ILU(0) is used, it is cost-effective to compute \( L_k \) and \( U_k \) from \( L_k \) and \( U_k \) respectively. Second, for the non-zero pattern of \( L_k \) and \( U_k \) are similar to that of \( A_k \), then the number of non-zeros in \( A_k \) can be controlled in some sense, leading to affordable additional computation costs. When the preconditioned conjugate gradient method is used with the described multigrid as a preconditioner, if the convergence is greatly accelerated, the number of iterations will be largely
reduced. Then the overall computation cost will be reduced too, which will be seen from the numerical results from next section.

**Numerical Experiments**

In this section, we will give some of the test results. All results are derived on a server of Intel(R) Xeon(R) CPU E5-2692 v2 @ 2.20GHz (cache 30720KB). The operating system is Red Hat Linux 2.6.32-220-TH, and the compiler used is Intel FORTRAN Version 11.1. All the test cases here are symmetric positive, and the preconditioned conjugate gradient is used with multigrid method as a preconditioner. When solving a linear system, the initial guess is the zero vector. For multigrid, we adopt original aggregation version here, with the coarsening scheme presented by Braess [3] and the Jacobi smoothing.

The iteration is stopped once the Euclid norm of the residual vector is reduced by $10^{10}$ in all tests. The time elapsed is recorded in seconds and in all the results, $DOF$ denotes the degree of freedoms, MG and ILUMG denote the aggregation based preconditioners as in figure 1 and 2 respectively. $C(grid)$ denotes the complexity of grid, that is, the ratio of the number of total vertexes in the multigrid method divided by the order of the original linear system. $C(oper)$ denotes the complexity of the operator, that is, the ratio of the number of total non-zeros of coefficient matrices on each level divided by the number of non-zeros of the coefficient matrix of the original linear system. $Nlev$ denotes the number of multigrid levels. $Cptime$ denotes the time elapsed for multigrid setup. $Iters$ and $Sttime$ denote the number of iterations and the iteration time to solve the linear system respectively.

The linear systems are discretized from Dirichlet boundary value problems of the 2D PDE

$$
-\frac{\partial}{\partial x} \left( \rho \frac{\partial u}{\partial x} \right) - \frac{\partial}{\partial y} \left( \rho \frac{\partial u}{\partial y} \right) + \rho u = f,
$$

which is defined on region $(0,c) \times (0,c)$, and the function $f$ and the boundary values are all defined through the assumed true solution $u=1$. All the linear systems considered here are obtained with finite difference scheme. If we select $n+2$ points in each direction, and for a continuous function $z$, denote $z(x, y)$ as $z_{ij}$, where

$$
x_i = ih, y_j = jh, j = 0, 1, \cdots, n + 1,
$$

and $h=cl/(n+1)$, we can discretize the PDE as

$$
-\rho_{i,j} u_{i,j+1} - \rho_{i+1,j} u_{i+1,j} - \rho_{i,j-1} u_{i,j-1} - \rho_{i-1,j} u_{i-1,j} + \lambda_{ij} u_{ij} = h^2 f_{ij},
$$

$$
\lambda_{ij} = \delta_{ij} h^2 + \rho_{i,j+1} + \rho_{i+1,j} + \rho_{i,j-1} + \rho_{i-1,j}.
$$

In system 1, $\rho=1$, $\delta=0$, $c=1$. In system 2 and 3,

$$
\rho = \begin{cases}
\rho_1, & x \in (2,2.1] \& y \in (1,2.1], \\
\rho_2, & x \in (1,2) \& y \in (1,2], \\
\rho_3, & x \in [0,1) \& y \in [0,2.1], \\
\rho_4, & y \in [0,1) \& x \in [0,2.1],
\end{cases}
$$

$$
\delta = \begin{cases}
\delta_1, & x \in (2,2.1] \& y \in (1,2], \\
\delta_2, & x \in (1,2) \& y \in (1,2], \\
\delta_3, & x \in [0,1) \& y \in [0,2.1],
\end{cases}
$$

In system 2,

$$
\rho_1 = 1, \rho_2 = 2 \times 10^3, \rho_3 = 3 \times 10^5,
$$

$$
\delta_1 = 0.02, \delta_2 = 3, \delta_3 = 500.
$$
In system 3,

\[ \rho_1 = x, \rho_2 = 2 \times 10^3 x, \rho_3 = 3 \times 10^4 x, \]
\[ \delta_1 = 0.02 x, \delta_2 = 3 x, \delta_3 = 500 x. \]  

Table 1. Results from model system 1.

| DOF  | Method | C(grid) | C(oper) | NLev | Sttime | Iter | Cptime |
|------|--------|---------|---------|------|--------|------|--------|
| 4096 | MG     | 1.3320  | 1.3244  | 5    | 0.0046 | 55   | 0.0534 |
|      | ILUMG  | 1.3379  | 8.1438  | 5    | 0.0466 | 23   | 0.0803 |
| 16384| MG     | 1.3330  | 1.3290  | 6    | 0.0139 | 103  | 0.3799 |
|      | ILUMG  | 1.3354  | 9.1999  | 6    | 0.2152 | 29   | 0.4315 |
| 65536| MG     | 1.3333  | 1.3312  | 7    | 0.0528 | 191  | 2.8260 |
|      | ILUMG  | 1.3346  | 9.9864  | 7    | 0.9823 | 35   | 2.2050 |
| 262144| MG    | 1.3333  | 1.3323  | 8    | 0.2038 | 356  | 21.530 |
|      | ILUMG  | 1.3340  | 10.674  | 8    | 4.7567 | 42   | 11.179 |
| 1048576| MG   | 1.3333  | 1.3328  | 9    | 0.7949 | 662  | 163.18 |
|      | ILUMG  | 1.3337  | 11.211  | 9    | 23.066 | 50   | 55.721 |

Table 2. Results from model system 2.

| DOF  | Method | C(grid) | C(oper) | NLev | Sttime | Iter | Cptime |
|------|--------|---------|---------|------|--------|------|--------|
| 4096 | MG     | 1.3333  | 1.3378  | 5    | 0.0047 | 100  | 0.0975 |
|      | ILUMG  | 1.3352  | 8.0014  | 5    | 0.0453 | 22   | 0.0761 |
| 16384| MG     | 1.3334  | 1.3345  | 6    | 0.0143 | 159  | 0.5882 |
|      | ILUMG  | 1.3346  | 9.2060  | 6    | 0.2129 | 31   | 0.4602 |
| 65536| MG     | 1.3335  | 1.3366  | 7    | 0.0538 | 184  | 2.7331 |
|      | ILUMG  | 1.3342  | 10.195  | 7    | 1.0054 | 43   | 2.7331 |
| 262144| MG    | 1.3335  | 1.3363  | 8    | 0.2050 | 323  | 19.342 |
|      | ILUMG  | 1.3338  | 10.787  | 8    | 4.7784 | 57   | 15.170 |
| 1048576| MG   | 1.3334  | 1.3363  | 9    | 0.7910 | 580  | 141.05 |
|      | ILUMG  | 1.3335  | 11.390  | 9    | 23.627 | 77   | 85.929 |

Table 3. Results from model system 3.

| DOF  | Method | C(grid) | C(oper) | NLev | Sttime | Iter | Cptime |
|------|--------|---------|---------|------|--------|------|--------|
| 4096 | MG     | 1.3347  | 1.4255  | 5    | 0.0051 | 94   | 0.0965 |
|      | ILUMG  | 1.3347  | 8.1453  | 5    | 0.0462 | 22   | 0.0769 |
| 16384| MG     | 1.3343  | 1.3875  | 6    | 0.0172 | 130  | 0.5029 |
|      | ILUMG  | 1.3347  | 9.1937  | 6    | 0.2164 | 31   | 0.4606 |
| 65536| MG     | 1.3344  | 1.4155  | 7    | 0.0598 | 263  | 4.0801 |
|      | ILUMG  | 1.3343  | 10.183  | 7    | 1.0244 | 42   | 2.6706 |
| 262144| MG    | 1.3343  | 1.4037  | 8    | 0.2300 | 490  | 30.722 |
|      | ILUMG  | 1.3338  | 10.827  | 8    | 4.8817 | 56   | 14.920 |
| 1048576| MG   | 1.3349  | 1.3649  | 22   | 0.8830 | 966  | 238.29 |
|      | ILUMG  | 1.3335  | 11.421  | 9    | 23.852 | 75   | 83.903 |

The test results for solving the three systems with different sizes are listed in table 1 to 3 respectively. From the tables, we can see that, when the problem size is small, ILUMG has no superiority. But ILUMG is less sensitive to the problem size, and with the increase of the size, the convergence rate of ILUMG is faster and faster than MG. For system 2 with discontinuous coefficients, and for system 3 with different strength in different directions, the acceleration is more significant. It is also clear that the convergence rate for any one of the three systems is nearly the same when ILUMG is used, which is very different from MG. When MG is used and the size is unchanged, the number of iterations to solve system 3 is apparently much more than that to system 1 and 2.

Though ILUMG converges more rapidly and is more efficient as a whole, its operator complexity is very large, which leads to relatively more time for the setup and more average time for each iteration. If appropriate approximation is used further, the operator complexity may be reduced further, which is the main focuses in future.
Summary

In this paper, we present a technique based on ILU (0) to smooth the coefficient matrices on each level for the aggregation based multigrid methods. The coefficient matrices on each level are factorized with ILU (0) and the triangular factors are approximately inverted with unchanged non-zero patterns. Numerical results show that the modified aggregation based multigrid is superior to the non-modified version. The relatively large operator complexity shows that there are potentials to improve further.

Acknowledgement

This research was financially supported by the National Science Foundation of China (61379022).

References

[1] M. Benzi, Preconditioning techniques for large linear systems: a survey, J. Phys. Comput., 182 (2002) 418-477.
[2] R. D. Falgout, An introduction to algebraic multigrid, Computing In Science & Engineering, 8:6 (2006) 24-33.
[3] C. Wagner, Introduction to algebraic multigrid, Course Notes. University of Heidelberg, 1998/1999; available at: http://www.iwr.uni-heidelberg.de/~Christian.Wagner/.
[4] P. Vanek, J. Mandel, M. Brezina, Algebraic multigrid based on smoothed aggregation for second and fourth order problems, Computing, 56 (1996) 179-196.
[5] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, J. Ruge, Adaptive smoothed aggregation (aSA) multigrid, Siam Review, 47:2 (2005) 317-346.
[6] R. Blaheta, Algebraic multilevel methods with aggregations: An overview, In I. Lirkov, S. Margenov & J. Wasniewski (Eds.), Large-Scale Scientific Computing, 3743 (2006) 3-14.
[7] Y. Notay, Aggregation-based Algebraic Multigrid for Convection-Diffusion Equations. Siam Journal On Scientific Computing, 34:4 (2012) A2288-A2316.
[8] E. Treister, I. Yavneh, On-the-fly adaptive smoothed aggregation multigrid for Markov chains, Siam Journal On Scientific Computing, 33:5 (2011) 2927-2949.
[9] I. Pultarova, Fourier analysis of the aggregation based algebraic multigrid for stochastic matrices, Siam Journal On Matrix Analysis and Applications, 34:4 (2013) 1596-1610.
[10] M. Brezina, A. Doostan, T. Manteuffel, S. McCormick, J. Ruge, Smoothed aggregation algebraic multigrid for stochastic PDE problems with layered materials, Numerical Linear Algebra With Applications, 21:2 (2014) 239-255.
[11] A. Frommer, K. Kahl, S. Krieg, B. Leder, & M. Rottmann, Adaptive Aggregation-Based Domain Decomposition Multigrid for the Lattice Wilson-dirac Operator. Siam Journal on Scientific Computing, 36:4 (2014) A1581-A1608.
[12] Y. Saad, Iterative methods for sparse linear systems. Boston: PWS Pub. Co., 1996.