Parallel Implementation of the Coordinates-partitioning Based Aggregation-type Algebraic Multigrid Preconditioners

Jian-ping WU*, Jun ZHAO and Shu-chang WANG

College of Meteorology and Oceanography, National University of Defense Technology, Changsha, China

*Corresponding author

Keywords: Aggregation based algebraic multigrid, Sparse linear system, Coordinates partitioning, Preconditioner, Conjugate gradient method, Parallel algorithm.

Abstract. The coordinates-partitioning based aggregation-type algebraic multigrid preconditioners have been proven to be very efficient in the solution of sparse linear systems with conjugate gradient iterations. In this paper, a parallel algorithm for the setup is provided and the parallelization of the preconditioning process is also considered. The parallel algorithm is based on a good property of the coordinates-partitioning based aggregation, that is, the aggregation process is performed in a fashion from the coarsest level to the finest step by step. Thus, the original adjacent graph is partitioned into a number of sub-graphs, where each sub-graph is related to a node in the coarsest level and is assigned to a processor. The aggregation process can then proceed forward on each processor independently from the assigned sub-graph. When this kind of multigrid preconditioner is applied in Krylov subspace iterations, only the computation on the coarsest level and the matrix-vector multiplications related to the smoothing on each level require communication for V- and W-cycle versions, and require only some extra communication related to dot products for the K-cycle version. The size of the derived linear system on the coarsest level can be controlled by the adjustable arguments and this system can be solved again in parallel with some preconditioned Krylov subspace iterations. The structure information related to the matrix-vector multiplications is invariable to the iterations and can be derived in the setup and be stored, and is used in the latter iterations. Finally, the parallel algorithm is validated for the Gauss-Seidel smoother and some kinds of popular multigrid cycles in solving sparse linear systems from some two-dimensional model partial equations with preconditioned conjugate gradient iterations. The results show that the parallel efficiencies of both the setup and the iteration processes are satisfied.

Introduction

The solution of sparse linear systems occupies much part of the whole run time in many scientific and engineering applications, and many methods have been presented to solve this kind of system. Among them, the Krylov subspace iteration [1] and the multigrid method [2] are the two most famous. Especially, multigrid methods can be incorporated with Krylov subspace iterations to exploit both the potential of optimal convergence rate of the former and the robustness of the latter, where the multigrid method is used as the preconditioner.

In a multigrid method, several level of grids are constructed firstly and a coefficient matrix related to each level is formed, where the one related to the finest level is just that of the original linear system. Given an initial guess to the solution on some level, after smoothing, an approximate solution can be derived, and then the residual vector is restricted to the coarser level to compute a correction. The correction is interpolated back to the finer level to correct the approximate solution on that level. On each level, this solution process is called recursively, until the size of the grid of the coarser level is small enough. On the coarsest level, the related system can be solved with any kind of method. When the grid levels are constructed mainly with algebraic information from the sparse linear system, the derived method can be regarded as an algebraic one [3].
There are several types of algebraic multigrid methods, among which the classical-type and the 
aggregation-type are two most frequently used ones. The classical type converges more rapidly in 
general, but is more expensive. In addition, the setup process is hardly to be parallelized. The 
aggregation type is simpler and cheaper for the construction process. Up to now, there are two ways 
to determine the interpolation operators, the classical method [3] and the method based on 
smoothing[4]. The former is very simple and cheaper. The latter can lead to more rapid convergence 
at the cost of more non-zero entries and more complex non-zero structures. Whichever way is used, it 
is required to construct the grid hierarchy first.

The most famous aggregation scheme is based on the concept of strong connection, where two 
end-points of the edge with maximum weight are aggregated into a coarse point [5]. It can be 
repeatedly applied to reduce the number of levels. The frequently used scheme is to repeat the process 
two or three times [6], that is, there are four or eight points put into an agglomeration at most 
respectively. To maintain the element structure, a coarsening scheme is provided based on 
concentrating three points each time [7]. To improve the quality of the grid hierarchy, each point not 
assigned to any aggregation can be added to an adjacent one as soon as possible [4]. Braess provided 
a scheme based on grouping each two points with maximum edge weights into one subset first, and 
then aggregating each two subsets with the maximum number of edges into an aggregation [8].

Wu et. al. provided a two-level grid correction for parallel incomplete factorization based on 
partitioning of the adjacent graph, with each part corresponding to a coarse grid point [9]. Kumar also 
provided a two-grid algorithm based on graph partitioning, with each sub-graph corresponding to an 
aggregation, where ILU(0) and ILUT are selected as the smoother and the solution scheme for the 
coarsest problem [10]. In reference [11], several frequently used aggregation schemes are tested and 
the experiments show that the best is always among the two-point scheme and its variants.

While the application process of algebraic multigrids is relatively ready to be parallelized, the 
parallelization of the setup process is much more difficult. In classical algebraic multigrids, it can be 
mainly reduced to the parallel section of the coarser grid points [12]. In aggregation based versions, the 
focus is the parallel section of the aggregations. Adams provided an aggregation scheme based on the 
maximal independent set algorithm [13]. Tuminaro et al provided three parallel schemes [14], a scheme 
coarsening in each processor independently. In [15], several parallel aggregation strategies are 
compared, and the results suggest that the types aggregating from the borders to the interiors are the best. 
In [16], parallelization techniques for multigrid solvers are surveyed, and parallel coarsening is one of the 
focuses.

In aggregation based algebraic multigrids, the aggregation process is in general based on local 
connectivity. Though for parallel computing, the original adjacent graph can be partitioned into 
sub-graphs and then apply aggregating schemes to them individually. The graph based aggregation 
has been used for two-level grid methods in [9] and [10], and in [17], such kind of method is provided 
for algebraic multigrid methods, where the construction starts from the coarsest grid to the finest one 
step by step. Each coarse-grid point is an aggregation of several fine-grid points and the number can 
be selected as needed. The test results have validated its effectiveness. In this paper, the 
parallelization for this scheme will be considered.

The Coordinates-Partitioning Based Aggregation-Type Algebraic Multigrid Preconditioner

Consider the following sparse linear system

\[ Au = b, \]

where \( A \) is a known \( n \) by \( n \) sparse matrix and is symmetric positive definite, \( b \) is a known vector of 
length \( n \), and \( u \) is the unknown vector. The linear system (1) can be solved with preconditioned 
conjugate gradient (PCG) iterations [1], where the preconditioner is selected as the aggregation based 
algebraic multigrid in this paper.

The algebraic multigrid methods have several types, including the V-cycle, the W-cycle and the 
K-cycle. For more details, we can refer to [3][6]. If we denote \( l \) as the current level number, an
approximation solution can be given by \( u^{(l)} = L_l^T b^{(0)} \) on level \( l \), where \( L_l \) is a smoother. The related residual vector \( r^{(l)} = b^{(0)} - A_l x^{(l)} \) is restricted to level \( l+1 \), multiplying \((P_{l+1})^T \) from the left. For matrix
\( B_l = (P_{l+1})^T A_l \) is invariant in the PCG iterations, it can be pre-computed in the setup process. On level \( l+1 \), the algorithm is invoked again to derive the correction vector \( v^{(l+1)} \). It is interpolated back to level \( l \), to correct the approximation \( u^{(l)} \). And then a post smoothing is applied to \( u^{(l)} \) with the smoother \( L_l^T \).

In this algorithm, we can select Gauss-Seidel iteration as the smoother. In this case, \( L_l \) is a matrix with \( L(i,j) = A(i,j) \) for \( j < i \) to \( i \) and \( i = 1 \) to \( n_l \), and all others zero. Besides \((P_{l+1})^T A_l \), the grid hierarchy, \( A_l \), \( P_{l+1} \), and \( L_l \) are also unchanged during the PCG iteration. Therefore, they can also be pre-computed and put into the set-up process beforehand, and be used directly in latter PCG iterations. In addition, the grid hierarchy is the kernel of this process, which is determined by the aggregation scheme. Once the grid hierarchy is determined, the other parts can be determined from it directly. Specially, when the simple classical interpolation operators are used, the coefficient matrix on a coarser level can be derived by first summation of the corresponding rows and then of the corresponding columns of the matrix on the finer level.

Assume that linear system (1) is derived from a certain partial differential equation and the coordinate information related to each unknown is given. Then the aggregation algorithm based on coordinates partitioning [17] can be described in detail as figure 1, where the input arguments are the coordinates of each unknown, \( m_0 \), \( p \) and \( G_i^{(0)} \), the adjacent graph of the coefficient matrix \( A \). For \( A \) is symmetric, \( G_i^{(0)} \) is undirected. The output is \( l \), array \( f \) and \( g \). The graph \( G_i^{(0)} \) is partitioned into \( m_0 \) parts firstly and then each part is recursively partitioned into \( p \) parts, until the minimal size of some sub-graph on some level is not larger than \( p \).

On output, a grid hierarchy is derived and the number of levels is given by the output argument \( l \). For \( k < l \), each sub-graph \( G_i^{(k)} \) is corresponding to a point in level \( k \). And for level \( l \), each point is corresponding to a point of \( G_i^{(0)} \). The value \( f_i \) records the label of the corresponding node in the coarser level of \( i \), and \( g_i \) is the position of the first value of level \( k \) in vector \( f \). Therefore, the 1st level has \( m_0 \) nodes, the 2nd level has \( p m_0 \) nodes, the 3rd level as \( p^2 m_0 \) nodes, and so on. The \((l-1)\)-th level has \( p^{l-1} m_0 \) nodes and the \( l \)-th level has \( n \) nodes.

Now we can reverse the grid hierarchy to setup the multigrid. The \( k \)-th level in the multigrid can be regarded as the \((l+1-k)\)-th level in figure 1. Then in the multigrid, the 1-st level has \( n \) nodes and the coarsest level is the \( l \)-th one, which has \( m_0 \) nodes. The nodes and the corresponding coefficient matrix in the 1st level have known beforehand, that is \( m=n \) and \( A \) respectively. For level \( k > 1 \), the nodes is given by \( f_{i+\gamma} \) for each node in level \( k-1 \), that is, for \( i=1, 2, ..., m_{i-1} \), where \( \gamma = g_{i+1-k} - 1 \). The nodes with the same \( f_{i+\gamma} \) are aggregated into a node in the coarser level \( k \).

1. Partition \( G_i^{(0)} \) into \( G_i^{(1)} \), \( i=1, 2, ..., m_0 \) with coordinate information
2. For \( i=1, 2, ..., m_0 \), set \( f_i = 1 \)
3. Set \( d^{(1)} = \min \{ G_i^{(1)} : i=1, 2, ..., m_0 \} \)
4. Set \( l=1 \) and \( g_1 = 1, g_2 = m_0 + 1 \)
5. While \( (d^{(1)} > p) \) do
6. Set \( l = l + 1 \) and \( m_{l+1} = pm_{l-2} \)
7. Set \( g_{l+1} = g_l + m_{l-1} \) and \( \gamma = g_{l+1} - 1 \)
8. For \( k=1 \) to \( m_{l-2} \) do
9. Partition \( G_i^{(l-1)} \) into \( G_{i+k-1}^{(l-1)} \), \( i=1, 2, ..., p \) with coordinate information
10. For \( i=1, 2, ..., p \), set \( f_{i+k-1}^{(1)} = \gamma \\
11. Enddo
12. Set \( d^{(b)} = \min \{ G^{(b)} : i=1, 2, ..., m_{l} \} \)
13. Enddo
14. Set \( l = l + 1, g_{l+1} = g_l + n \) and \( \gamma = g_{l+1} - 1 \)
15. For \( k=1 \) to \( m_{l-2} \) do
16. For \( i \in G_i^{(l-1)} \) set \( f_{i+\gamma} = k \)
17. Enddo

Figure 1. Algorithm ACRP (the aggregation algorithm based on coordinates partitioning recursively).
Combining all the issues above, the whole setup process can be described as figure 2, where \( \text{ibJ}_k \) and \( \text{kJ}_k \) are used to store the aggregations in CSR format. In figure 2, step 8 is used to compute a ILU(0) factorization, which is used as the preconditioner of the sparse linear system on level \( l \). In this paper, this system will be solved with preconditioned conjugate gradient iterations again.

1. Invoke algorithm ACRP with \( m_0, p \) and matrix \( A \), deriving \( l, f \) and \( g \)
2. For \( k = 1 \) to \( l - 1 \) do compute \( m_k = g_{k+2} - g_{k+1} \)
3. For \( k = 1 \) to \( l - 1 \) do
4. Compute \( \text{ibJ}_k \) and \( \text{kJ}_k \) with \( m_{k, k}, m_{k,k+1} \) and \( f(g_{k,k+1}; g_{k,k+1} + m_{k,k}) \)
5. Compute \( B_k = (P_{k,k+1})^T A_k \)
6. Compute \( A_{k+1} = B_k P_{k,k+1} \)
7. Endfor
8. Compute ILU(0) factorization of \( A_l \)

Figure 2. The setup process based on algorithm ACRP described as figure 1.

**Parallel Implementation of the Coordinates-Partitioning Based Multigrid Preconditioner**

Though there are several ways to parallelize a given algorithm, only the message passing version is considered here. The message passing interface (MPI) is used and the parallel algorithms are based on the domain decomposition method. Assume that the linear system (1) has been distributed to \( P \) processors, with approximately even number of rows of the coefficient matrix \( A \) assigned to each processor. For simplicity, we assume that the rows assigned to each processor have contiguous indexes. The components of all the vectors on each level are assigned to the processors according to the rows of the coefficient matrix. In the parallelization of PCG iteration and the multigrid algorithm, the results of the inner products are repeatedly stored on all processors. Therefore, the inner products are done with the MPI function MPI_Allreduce. In this way, to complete the whole parallel implementation, the remaining components includes the partitioning algorithm ACRP, the setup process described as figure 2, the smoothing steps in the application of multigrid, the multiplication of matrices to a vector.

To paralleling the algorithm ACRP, it can be done by replacing \( G_i^{(0)} \) in figure 1 with the local part. It should be noted that we only know the rows of \( A \) assigned to this processor, which can be regarded as a sub-matrix. It can be illustrated as figure 3. Without loss of generality, we assume that rows assigned to this processor have indexes from \( \text{iB} \) to \( \text{iB} + \text{iL} - 1 \), where \( \text{iB} \) is the index of the first row and \( \text{iL} \) is the number of rows. But the column indexes may ranging from 1 to \( n \). To derive the local version of \( G_i^{(0)} \), we can preserve the entries with column indexes ranging from \( \text{iB} \) to \( \text{iB} + \text{iL} - 1 \), and then subtracting \( \text{iB} - 1 \) from each index. Then we can apply the algorithm ACRP with the derived local version on all processors in parallel. The only operation requiring communication is the computation of \( \alpha^{(l)} \), that is, step 3 and 12 in figure 1, such that the number of levels \( l \) is the same for all processors. In addition, for the first several levels, communications may not be needed at all. Only if the number of nodes in each sub-graph is small enough, the communication step is required. In this paper, for simplicity, this saving in communication is not considered. The MPI function MPI_Allreduce is used for each level instead.

For the setup process described as figure 2, step 5, 6 and 8 should be considered for parallel processing. Step 8 is to perform the ILU(0) factorization of \( A_l \). For the order of \( A_l \) will be by far smaller than \( n \), when PCG iteration is used to solve a linear system with \( A_l \) as the coefficient matrix,
the convergence rate can be expected to vary very fast even if much cheaper preconditioner is used. Thus, in parallel computation, a block diagonal analog can be used as the preconditioner, where $A_i$ is firstly approximated by a diagonal block matrix, with one block on each processor, and then ILU(0) is performed to each block locally. Step 5 means to perform a summation over all the rows related to each aggregation. The row on the coarser level related to an aggregation is on the same processor as those related to its entries. Therefore, there are no communications. Step 6 means to perform a summation over the columns, which requires no communication too. But the column indexes are global, ranging from 1 to $m_{i-1}$ for level $l$, therefore, a column derived from the summation over the columns related to the same aggregation will have a global column index too, which is given by the corresponding component of the vector $f$. Before summation, we should know that which aggregation is in for each column. Thus a global snapshot of the aggregations over all the processors should be derived on each processor.

To get the global snapshot of all the aggregations, three steps are required. First, we should derive the number of points on all processors in each level. If denote the number of points on processor $i$ as $n_{i,k}$ for level $k$ in multigrid, then we know that for level 1, $n_{i,1}$ is different from each other, so they should be passed to all the processors, which can be implemented with the MPI function MPI_Allgather. For level $k>1$, the number of local points must be $m_{i,k}$ on every processor. Therefore no communication is required. Second, the number of points in each aggregation should be known on each processor. By the construction, for the first level, it is variable for different aggregations. But for level $k>1$, it is equal to $p$ at all the time. Thus, communication is required for the first level only and it can be implemented with MPI_Allgatherv. Finally, the indexes of the points in all the aggregations should be known. For the same reason as for the number of points in each aggregation, communication is required for the first level only again and it can also be done with another MPI_Allgatherv. For level $k>1$, each processor has $m_{i,k}$ points, and each aggregation has $p$ points, thus for each point on this level, the aggregation index can be computed directly.

For the multiplication $v = Au$, if there a matrix entry with column index $j$ not in $iB$ to $iB+iL-1$, where $iB$ and $iL$ are the same as before, the $j$-th component of $u$ must be on other processors and should be received from the related processor. As for $v$, the components can be classified into two classes. Some components of $v$ will not depend on any components of $u$ belonging to other processors, which is called the interior components. The others are called the border components. Therefore, the computation of the interior component can be done in parallel on all the processors. At the same time, for the components of $u$ which the border components depend on, a non-blocked receive operation can be issued. And the corresponding send operations can be issued too. After the computation of the interior components, the MPI_Wait function can be used to wait the receive operations to complete. Once they are completed, the computation of the border components is followed. In this way, the computation and the communication can be overlapped. The multiplication of $A_i$ to a vector $u^{(i)}$ and the multiplication of $B_i$ to a vector $u^{(i)}$ is very similar. The only difference is that $B_i$ has less rows than $A_i$. It should be noted that the structure information used in the multiplication will be invariable to the PCG iterations, including the number of components received from each processor, the number of components sent to each processor, the component indexes received and sent, and so on. This invariable information can be computed in the setup process and be used in the iterations directly.

For the smoothing process in the application of multigrid, we can approximate $L_i$ with a diagonal block matrix in a similar way to the parallelization of ILU(0). Then each processor can compute a local part of $u^{(i)}$ independently. Correspondingly, the post smoothing should be used on the condition that $L_i$ is replaced by this block version, to ensure the symmetric property of the preconditioner, which is vital to the convergence of the PCG iterations. Apparently, the computation in the brackets can be done in a similar way to the operation $Au^{(i)}$. And the multiplication of $L_i^{-T}$ to the result can be done independently on all the processors again with the diagonal block approximation to $L_i$.
Numerical Experiments

In this section, all the experiments are performed on a cluster of workstation, which has 32 nodes, with two processors on each node. The processor is Intel(R) Xeon(R) CPU E5-4640 0 @ 2.40GHz (cache 20480 KB). The operating system is Linux version 2.6.32-431.el6.x86_64 and the compiler is Intel FORTRAN Version 15.0.0.090. The MPICH version 3.1.3 is used for communication between processors. For the initial linear systems, they are distributed to processors according to the way mentioned in section 3 and the partitioning scheme is the same as step 1 and 9 in figure 1. After partitioning, the rows assigned to a processor may have discontinuous indexes. Therefore, they are relabeled such that the indexes are continuous on every processor and the indexes on processor $k$ are all less than those on processor $k+1$ for $k=0$ to $P-2$. The column indexes are relabeled correspondingly. When solving a linear system with multigrid preconditioned conjugate gradient method, for the outer PCG iteration, the initial approximation is selected as zero all the time and the stop criterion is selected as the 2-norm of the residual vector is reduced by 1E-10. For the inner PCG iteration on the coarsest level, the linear system is solved with preconditioned conjugate gradient too, and the initial approximation is selected similarly, while the stop criterion is selected as the 2-norm of the residual vector is reduced by 1/10. During the coarsening process, the number of nodes on the coarsest level for each processor is selected as $m_0=1024/P$, where $P$ is the number of processors. This selection can ensure that the order of the linear system on the coarsest level is $m_0$, invariable to the number of processors. Thus, the number of levels are almost the same too for different number of processors.

Two numerical experiments are done for discrete linear systems derived from a model partial differential equation with finite difference schemes, denoted as Lin1 and Lin2 respectively. They are derived from the following PDE problem with Dirichlet boundaries

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f,$$

where $x, y \in (0,1)$ and the function $f$ and the boundary values are given from a true solution $u=1$. In the discretization, $n+2$ discrete points are selected in each dimension and $u(x_i, y_j)$ is denoted as $u_{ij}$ for any function $u$, where

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

and $h=1/(n+1)$. For Lin1, The discrete form is

$$-u_{i-1,j} - u_{i+1,j} + 4u_{i,j} - u_{i,j-1} - u_{i,j+1} = h^2 f_{ij}, \quad (3)$$

For Lin2, the following discrete form is used

$$-u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} + 8u_{i,j} - u_{i+1,j-1} - u_{i-1,j+1} - u_{i,j+1} - u_{i,j-1} = h^2 f_{ij}. \quad (4)$$

In the experiments, $n$ is selected as 2048 for both linear systems. The results are given in table 1. In the table, Vcyc and Wcyc mean the V- and W- cycle multigrid respectively, and K001, K025, K035 denote the K- cycle multigrid with $t=0.01$, $t=0.25$, and $t=0.35$ respectively. The sign setup and iteration mean the setup and the iteration process respectively. The speedup is derived from the time on one processor to the parallel execution time for the same cycle.

From table 1, it is clear that for the linear system Lin1, when the number of points aggregated each time is 2, the speedup for the setup is increased step by step until 32 processors used for each cycle. When 64 processors is used, the speedup is worse than that for 32 processors. The speedup is increased step by step for V-cycle until 64 processors. For K-cycle with $t=0.35$, the maximal speedup is reached when 32 processors is used. For other cycles, the maximal speedup is reached when 16 processors is used instead. When 32 processors is used, approximately 20x speedup can be achieved for the setup process. As for iteration, the maximum speedup is achieved for V-cycle when 64 processors used, which is approximately 11.5.

It can be seen from table 1 that when solving Lin1 with 4 points aggregated each time, the speedup for the setup reaches the maximal when 32 processors used again. In addition, the speedup for the iteration reaches the maximal when 64 processors is used for every cycle. The setup process can
achieve approximately 22x speedup and the iteration process can achieve 11.4x speedup for V-cycle, which is maximal over all the cycles. When 8 points are aggregated each time, the setup process can achieve the maximal speedup with 32 processors again, and the maximum is 21.9. The iteration process can achieve 11.9x speedup for V-cycle again, which is the maximum over all cycles.

Table 1. Speedup for the setup and iteration process when solving Lin1 and Lin2 using different schemes, where P is the different number of processors and p is the different number of points aggregated each time.

<table>
<thead>
<tr>
<th></th>
<th>Lin1</th>
<th>Lin2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P=2</td>
<td>P=4</td>
</tr>
<tr>
<td>setup</td>
<td>2.30</td>
<td>5.43</td>
</tr>
<tr>
<td>iter</td>
<td>Vcyc</td>
<td>1.90</td>
</tr>
<tr>
<td></td>
<td>Wcyc</td>
<td>1.72</td>
</tr>
<tr>
<td></td>
<td>K001</td>
<td>1.72</td>
</tr>
<tr>
<td></td>
<td>K025</td>
<td>1.80</td>
</tr>
<tr>
<td></td>
<td>K035</td>
<td>1.85</td>
</tr>
<tr>
<td>setup</td>
<td>2.45</td>
<td>5.84</td>
</tr>
<tr>
<td>iter</td>
<td>Vcyc</td>
<td>1.87</td>
</tr>
<tr>
<td></td>
<td>Wcyc</td>
<td>1.83</td>
</tr>
<tr>
<td></td>
<td>K001</td>
<td>1.90</td>
</tr>
<tr>
<td></td>
<td>K025</td>
<td>1.88</td>
</tr>
<tr>
<td></td>
<td>K035</td>
<td>1.82</td>
</tr>
<tr>
<td>setup</td>
<td>2.34</td>
<td>5.94</td>
</tr>
<tr>
<td>iter</td>
<td>Vcyc</td>
<td>1.85</td>
</tr>
<tr>
<td></td>
<td>Wcyc</td>
<td>1.63</td>
</tr>
<tr>
<td></td>
<td>K001</td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td>K025</td>
<td>1.79</td>
</tr>
<tr>
<td></td>
<td>K035</td>
<td>1.32</td>
</tr>
</tbody>
</table>

For linear system Lin1, there are only 5 non-zero entries in each row of the coefficient matrix. Let us to investigate the cases when Lin2 is solved. For these cases, there are 9 non-zero elements in each row. From table 1, we can see that, for the setup, the speedup is always attained when 32 processors are used. For p=2, 4 and 8, the speedup attains 18.8, 19.8 and 21.1. As for the iteration time, the speedup is always higher when the V-cycle is used. The maximum is approximately 14.

From the above analyses, it can be concluded that the parallel implementation provided in this paper is satisfied. The setup process can efficiently scale to 32 processors and the iteration process can scale to 64 processors for most of the cases, especially for the V-cycle multigrid preconditioner. For the cases with more non-zeros in each row of the coefficient matrix such as Lin2, the speedup is significantly higher. The maximal speedup for Lin2 can attain 14 or more, while for Lin1, it is about 12. In addition, it should be noted that, as for the iteration time, the K-cycle with proper parameter t will be better than the other cycles.

Summary

In this paper, a parallel implementation for the setup and for the iteration process is provided. In the implementation, the original sparse linear system is assumed to be distributed to processors beforehand. For the setup process, the parallel implementation is based on the serial coordinates-based recursive partitioning, with only a few communication operations inserted, and with some additional operations related to the matrix-vector multiplications added. And communication reduction schemes based on this special partitioning is carefully considered and applied. For the iteration process, the scheme to overlap computations and communications is applied for the matrix-vector multiplication and the simplified Gauss-Seidel smoother, where the block diagonal approximation is used. Finally, the parallel implementation is validated for some kinds of popular multigrid cycles in solving sparse linear systems from some two-dimensional model partial
equations with preconditioned conjugate gradient iterations. The results show that the parallel efficiencies of both the setup and the iteration processes are satisfied.

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

References