Two level parallel preconditioning derived from an approximate inverse based on the Sherman–Morrison formula

L. Zhang[1] K. Moriya[2] T. Nodera[3]

(Received 23 May 2009; revised 21 September 2012)

Abstract

The AISM (Approximate Inverse based on the Sherman–Morrison Formula) method is one of the existing effective methods for computing an approximate inverse. This algorithm was proposed by Bru et al. [*SIAM J. Sci. Comput.*, **25**, pp.701–715 (2003)]. Although it has been showed that the AISM is generally a stable option for large linear systems of equations, its computation cost can be prohibitively high. Complications also arise when an attempt is made to parallelize the algorithm, since a sequential process is necessary. This article proposes a two level AISM in which the coefficient matrix is rearranged to a block form, which is more suitable for parallel computation. This technique can also significantly speed-up computations on a single processor. We implemented this technique on an Origin 2400 system with an MPI to illustrate its efficiency through numerical experiments.

Contents

1 Introduction **E2**

2 The AISM method **E4**

3 The two level parallel AISM method **E7**

4 Application to the Krylov subspace method **E10**

5 Numerical experiments **E12**
5.1 Example: elliptic PDE in the square E14
5.2 Example: irregular coefficient matrix E18

6 Concluding Remarks **E20**

References **E22**

1 Introduction

Assuming that A is a real $n \times n$ nonsingular and nonsymmetric matrix, our principal concern lies in finding the solution for the large, sparse linear systems of equations

$$Ax = b\,. \tag{1}$$

Scientific computations are fraught with these types of linear systems. For solving these types of linear systems of equations, Krylov subspace methods such as GMRES(m), BI-CGSTAB(ℓ) and IDR(s), are commonly used [4, 20, 22, 24, 25]. However, for more complex problems, that is, those with a large condition number or bad spectral distribution of the coefficient matrix A, the Krylov subspace methods may stagnate or converge very slowly [21, 24]. The use of certain preconditioning techniques is one of the options for improving the convergence of Krylov subspace methods [4, 20].

A preconditioning process includes computing a preconditioner $M \in \mathbb{R}^{n\times n}$ and solving the *right preconditioned* system

$$AM\mathbf{y} = \mathbf{b} \quad \text{and} \quad \mathbf{x} = M\mathbf{y}\,, \tag{2}$$

or alternatively solvin the *left preconditioned* system

$$MA\mathbf{x} = M\mathbf{b}. \tag{3}$$

There is also the option of two sided preconditioning. However, since right preconditioning (2) does not change the residual $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ of the original equation (1), this right preconditioning is more commonly used [20]. In this article, we chose to use right preconditioning.

For choosing the preconditioner M, the sparse approximate inverse of A is often used in the field of parallel computation [2, 5, 6, 11, 12, 16, 17]. For computing the sparse approximate inverse of coefficient matrix A, the AISM consistently performed well [3]. However, it was no easy task to parallelize the Sherman–Morrison formula since a sequential process needed to be included. The computation cost was relatively high. For these reasons, the AISM is not the favored method for computing large sparse linear systems of equations (1). Recently, in order to reduce computation time, Moriya et al. [15] proposed a partially parallelized AISM based on a technique developed by Naik [19], in which vectors were distributed in all processors that were making computations and communications were carried out in turns.

The two level parallel technique rearranges the coefficient matrix to a block form. This is a different approach from that of parallelizing the algorithm itself, and is more suitable for parallel computation. This parallel technique was originally proposed for the Cholesky decomposition [9], and Benzi et al. [2] proposed a two level parallel preconditioner based on an AINV (Approximate Inverse) with a two level parallel strategy.

In this article, we propose a two level parallel AISM where the AISM is applied to the two level parallel strategy. The two level parallel AISM was implemented on an Origin 2400 and our numerical experiments indicate that our proposed strategy is effective for parallel computation. Moreover, the two level AISM results in a significant speed-up for non-parallel computation. The numerical results of this non-parallel performance are tabulated in this study in detail.

2 The AISM method

The AISM (Approximate Inverse based on the Sherman–Morrison Formula) method computes the approximate inverse using the Sherman–Morrison formula [10, 18].

Sherman–Morrison Formula Given a nonsingular matrix $B \in \mathbb{R}^{n\times n}$ and two vectors $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$ such that $r = 1 + \mathbf{y}^T B^{-1}\mathbf{x} \neq 0$, the matrix $A = B + \mathbf{x}\mathbf{y}^T$ is nonsingular, and its inverse is

$$A^{-1} = B^{-1} - r^{-1}B^{-1}\mathbf{x}\mathbf{y}^T B^{-1}. \tag{4}$$

Let $\mathbf{x}_k \in \mathbb{R}^n$, $\mathbf{y}_k \in \mathbb{R}^n$ and $A_0 \in \mathbb{R}^{n\times n}$ satisfy $A_k = A_0 + \sum_{i=1}^{k} \mathbf{x}_i\mathbf{y}_i^T$, where A_0 is a nonsingular matrix, where k runs from 1 to n, and $A = A_n$. If A_k, $\mathbf{x}_k$ and $\mathbf{y}_k$ satisfy the Sherman–Morrison formula's conditions, then the inverse of A can be computed by applying the Sherman–Morrison formula (4) n times with

$$A^{-1} = A_0^{-1} - \sum_{k=1}^{n} r_k^{-1}A_{k-1}^{-1}\mathbf{x}_k\mathbf{y}_k^T A_{k-1}^{-1}\,, \quad r_k = 1 + \mathbf{y}_k^T A_{k-1}^{-1}\mathbf{x}_k\,, \quad k = 1, \ldots, n\,, \tag{5}$$

when equation (5) is rewritten into matrix form, and

$$A_0^{-1} - A^{-1} = \Phi\Omega^{-1}\Psi^T, \tag{6}$$

where $\Phi = [A_0^{-1}\mathbf{x}_1, A_1^{-1}\mathbf{x}_2, \ldots, A_{n-1}^{-1}\mathbf{x}_n]$, $\Omega^{-1} = \operatorname{diag}[r_1^{-1}, r_2^{-1}, \ldots, r_n^{-1}]$ and $\Psi = [\mathbf{y}_1^T A_0^{-1}, \mathbf{y}_2^T A_1^{-1}, \ldots, \mathbf{y}_n^T A_{n-1}^{-1}]$. Note that when matrix Φ and Ψ are computed, $A_0^{-1}, \dots, A_{n-1}^{-1}$ needs to be computed. In order to avoid computing $A_0^{-1}, \dots, A_{n-1}^{-1}$ explicitly, the vectors $\mathbf{u}_k$ and $\mathbf{v}_k$ are defined as

$$\mathbf{u}_k := \mathbf{x}_k - \sum_{i=0}^{k-1} \frac{\mathbf{v}_i^T A_0^{-1} \mathbf{x}_k}{r_i} \mathbf{u}_i \,, \quad \mathbf{v}_k := \mathbf{y}_k - \sum_{i=0}^{k-1} \frac{\mathbf{y}_k^T A_0^{-1} \mathbf{u}_i}{r_i} \mathbf{v}_i \,, \quad k = 1, \dots, n\,.$$

Then,

$$\begin{aligned} A_{k-1}^{-1} \mathbf{x}_k &= A_0^{-1} \mathbf{u}_k \,, & (7)\\ \mathbf{y}_k^T A_{k-1}^{-1} &= \mathbf{v}_k^T A_0^{-1}, & (8)\\ r_k = 1 + \mathbf{y}_k^T A_0^{-1} \mathbf{u}_k &= 1 + \mathbf{v}_k^T A_0^{-1} \mathbf{x}_k \,. & (9) \end{aligned}$$

Equation (6) is rewritten with equations (7) and (8):

$$A_0^{-1} - A^{-1} = A_0^{-1} U \Omega^{-1} V^T A_0^{-1}, \tag{10}$$

where matrices $U = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n]$ and $V = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n]$.

In the selection of A_0, $\mathbf{x}_k$ and $\mathbf{y}_k$, Bru et al. [3] proposed

$$A_0 = s I_n \,, \quad (s > 0), \quad \mathbf{x}_k = \mathbf{e}_k \,, \quad \mathbf{y}_k = (\mathbf{a}^k - \mathbf{a}_0^k)^T, \quad k = 1, \dots, n\,,$$

where $I_n \in \mathbb{R}^{n \times n}$ is an identity matrix, $\mathbf{e}_k \in \mathbb{R}^n$ is the kth column of I_n. Vectors $\mathbf{a}^k$ and $\mathbf{a}_0^k$ are the kth row of matrices A and A_0. Substitute A_0, $\mathbf{x}_k$ and $\mathbf{y}_k$ in equation (10), and the result is

$$A^{-1} = s I_n - s^{-2} U \Omega^{-1} V^T, \tag{11}$$

and

$$\mathbf{u}_k = \mathbf{x}_k - \sum_{i=1}^{k-1} \frac{(\mathbf{v}_i)_k}{s r_i} \mathbf{u}_i \,, \quad \mathbf{v}_k = \mathbf{y}_k - \sum_{i=1}^{k-1} \frac{\mathbf{y}_k^T \mathbf{u}_i}{s r_i} \mathbf{v}_i \,, \quad r_k = 1 + (\mathbf{u}_k)_k / s \,, \tag{12}$$

where $(\mathbf{v}_i)_k$ is the kth element of $\mathbf{v}_i$, and $(\mathbf{u}_k)_k$ is the kth element of $\mathbf{u}_k$.

Algorithm 1: The AISM method.

```
for $k = 1, \dots, n$ do
    $\mathbf{x}_k = \mathbf{e}_k$, $\mathbf{y}_k = (\mathbf{a}^k - s\mathbf{e}_k)^T$;
    $\mathbf{u}_k = \mathbf{x}_k$, $\mathbf{v}_k = \mathbf{y}_k$;
    for $i = 1, \dots, k-1$ do
        $\mathbf{u}_k = \mathbf{u}_k - \frac{(\mathbf{v}_i)_k}{sr_i}\mathbf{u}_i$;
        $\mathbf{v}_k = \mathbf{v}_k - \frac{\mathbf{y}_k^T\mathbf{u}_i}{sr_i}\mathbf{v}_i$;
    end
    for $i = 1, \dots, n$ do
        if $|(\mathbf{u}_k)_i| < \mathsf{tolU}$ then drop-off $(\mathbf{u}_k)_i$;
        if $|(\mathbf{v}_k)_i| < \mathsf{tolV}$ then drop-off $(\mathbf{v}_k)_i$;
    end
    $r_k = 1 + (\mathbf{u}_k)_k/s$;
end
```

Matrices U and V are likely to be dense and if this is the case, then it will be necessary to drop the small elements to achieve an approximate sparse inverse of A:

$$A^{-1} \approx sI_n - s^{-2}U\Omega^{-1}V^T. \tag{13}$$

Putting the above derivations all together, the algorithm of the AISM is shown in Algorithm 1.

Since the AISM performs consistently well [3], it is often used for computing the approximate inverse preconditioner for solving large sparse linear systems.

3 The two level parallel AISM method

The two level technique rearranges the coefficient matrix into

$$P^TAP = \begin{bmatrix} A_1 & & & B_1 \\ & \ddots & & \vdots \\ & & A_p & B_p \\ C_1 & \cdots & C_p & A_s \end{bmatrix}, \tag{14}$$

where P is a permutation matrix. Since blocks A_i are independent of each other, their approximate inverses can be computed in parallel. Here the subscript s is different from the scalar in the AISM.

In order to transform the coefficient matrix into the form given in equation (14), graph partitioning and domain decomposition are used. In this article, graph partitioning is used, since it is already available for general problems.

Let graph $G = \langle V, E\rangle$ be the graph of matrix $A \equiv (a_{ij}) \in \mathbb{R}^{n\times n}$, where $V = \{1, \ldots, n\}$ is the set of vertices and E is the set of edges $\{\langle i, j\rangle \mid i, j \in V, a_{ij} \neq 0\}$. Graph partitioning algorithms partition graph G into p subgraphs G_i $(i = 1, \ldots, p)$ that are roughly of equal size and have a smaller number of edges that are cut by the partitioning. The nodes in subgraphs are then divided into two groups. Nodes that are not connected by the edges cut by the partitioning are called inner nodes, and the others are called separator nodes. In this, the inner nodes in G_i are denoted as g_i^I, and the separator nodes in G_i are denoted as g_i^B. The next step is to rearrange the nodes in the order of g_1^I, g_2^I, …, g_p^I, g_1^B, g_2^B, …, g_p^B. Matrix A is also permuted according to the new order, resulting in the block angular form given in equation (14). The dimension of submatrix A_i consists of the number of inner nodes belonging to subgraph G_i, and the dimension of A_s consists of the total number of separator nodes.

For a given graph, the number of inner nodes in the sub graphs decrease and the number of separator nodes increase when the number p increases.

The dimension of A_i decreases and the dimension of the Schur complement increases with p.

The next step is to obtain the inverse of P^TAP. Its inverse is computed with

$$(P^TAP)^{-1} = \begin{bmatrix} A_1^{-1} & & & E_1 \\ & \ddots & & \vdots \\ & & A_p^{-1} & E_p \\ & & & S^{-1} \end{bmatrix} \begin{bmatrix} I_1 & & & \\ & \ddots & & \\ & & I_p & \\ F_1 & \cdots & F_p & I_s \end{bmatrix}, \qquad (15)$$

where $E_i = -A_i^{-1}B_iS^{-1}$ and $F_i = -C_iA_i^{-1}$. $S = A_s - \sum_{i=1}^p C_iA_i^{-1}B_i$ is the Schur complement. I_i and I_s are identity matrices that have the same dimension as A_i and A_s. When we replace A_i^{-1} and S^{-1} with their approximate inverses computed with the AISM, the approximate inverse of P^TAP is

$$(P^TAP)^{-1} \approx \begin{bmatrix} M_1 & & & \bar{E}_1 \\ & \ddots & & \vdots \\ & & M_p & \bar{E}_p \\ & & & M_s \end{bmatrix} \begin{bmatrix} I_1 & & & \\ & \ddots & & \\ & & I_p & \\ \bar{F}_1 & \cdots & \bar{F}_p & I_s \end{bmatrix}, \qquad (16)$$

where $M_i \approx A_i^{-1}$, $M_s \approx \bar{S}^{-1}$, $\bar{E}_i = -M_iB_iM_s$, $\bar{F}_i = -C_iM_i$. $\bar{S} = A_s - \sum_{i=1}^p C_iM_iB_i$ is the approximate Schur complement. In this article, $C_iM_iB_i$ is referred to as the local part of the approximate Schur complement. Usually, $\bar{E}_i$ and $\bar{F}_i$ are not computed explicitly.

The two level AISM computes the approximate inverse of P^TAP with equation (16).

Level 1: Compute $M_i \approx A_i^{-1}$ with the AISM.

Level 2: Compute $M_s \approx \bar{S}^{-1}$ with the AISM.

The approximate inverse of P^TAP is obtained by computing the approximate inverses of A_i and $\bar{S}$, which are much smaller than the original matrix A. Instead of computing the original approximate inverse problem, $(p+1)$ smaller approximate inverse problems are computed.

It is known that the computation cost for the original approximate inverse problem is $O(\dim(A)^3)$ [1, 3]. This indicates that the cost for a $(p+1)$ smaller approximate inverse problems is

$$O\left(\sum_{i=1}^{p}\dim(A_i)^3+\dim(\bar{S})^3\right),$$

where $\dim(D)$ is the dimension of matrix D. If p is selected properly, then the computation cost of the $(p+1)$ smaller approximate inverses are likely to be more cost effective than the original calculations with a proper p. It is likely that this will result in a significant speed-up on a single processor.

The optimum p on a single processor, denoted by p_{optS} is

$$p_{\text{optS}}=\min_p\sum_{i=1}^{p}\left\{\dim(A_i)^3+\dim(\bar{S})^3\right\}. \qquad (17)$$

This evaluation is relative, since the cost for an approximate inverse depends heavily on the number of nonzero elements, but it is easy to implement and effective to select the value of p.

After the aforementioned evaluation (17) is made, a two level parallel AISM parallel computation is implemented.

The computations of M_i in level 1 are independent of each other, and is computed separately. In addition, the local part of the approximate Schur complement $C_iM_iB_i$ is computed separately.

We set the value of p in equation (16) to coincide with the number of processors. It is possible to have a different number of processors from the value of p, but it will make the application far more complicated.

The first step is to let the ith processor compute $M_i \approx A_i$ and $\bar{S}_i = C_iM_iB_i$, then send the computed $\bar{S}_i = C_iM_iB_i$ to a certain processor, for example, let the first processor compute $\bar{S}$ with $\bar{S}_i = C_iM_iB_i$ and its approximate inverse M_s. After M_s is computed, the first processor sends it over to the

Table 1: Steps of the two level parallel AISM method

step	first PE	...	pth PE
1	Compute $M_1 \approx A_1^{-1}$	...	Compute $M_p \approx A_p^{-1}$
2	Compute $\bar{S}_1 = C_1 M_1 B_1$	...	Compute $\bar{S}_p = C_p M_p B_p$
3	Send $\bar{S}_1$ to other PEs	...	Send $\bar{S}_p$ to other PEs
4	Compute $\bar{S} = A_s - \sum_{j=1}^{p} \bar{S}_j$	...	Compute $\bar{S} = A_s - \sum_{j=1}^{p} \bar{S}_j$
5	Compute $M_s \approx \bar{S}^{-1}$	...	Compute $M_s \approx \bar{S}^{-1}$

other processors. In level 2, all processors with the exception of the first processor are standing by. In order to avoid sending M_s to the other processors, we let each processor compute $\bar{S}$ and M_s simultaneously. Table 1 gives more details of our two level parallel AISM. Henceforth, a processor will be referred to as a PE.

Similar to when a single processor is used, the optimum value of p of the parallel computation is

$$p_{\text{optP}} = \min_{p} \max_{i=1,\dots,p} \left\{ (\dim(A_i)^3) + \dim(\bar{S})^3 \right\}. \tag{18}$$

4 Application to the Krylov subspace method

This section explains how to apply the computed approximate inverse to the Krylov subspace method. We use GMRES(m).

After the approximate inverse is computed with equation (16), the ith PE will contain data for A_i, B_i, C_i, A_s, M_i, $\bar{S}$ and M_s. Based on this, we partition any vector, for $\mathbf{x} \in \mathbb{R}^n$, in the GMRES$(m)$ as $(x_1, x_2, \dots, x_p, x_s)^T$, and let the ith PE store the data of x_i and x_s. The dimensions of $x_1, x_2, \dots, x_p, x_s$ will coincide with the dimensions of $A_1, A_2, \dots, A_p, A_s$.

The multiplication of the permutated coefficient matrix $P^{T}AP$ with vector $\boldsymbol{x}$ can be computed in the following manner:

$$\begin{bmatrix} A_1 & & & B_1 \\ & \ddots & & \vdots \\ & & A_p & B_p \\ C_1 & \cdots & C_p & A_s \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_s \end{bmatrix} = \begin{bmatrix} A_1x_1 + B_1x_s \\ \vdots \\ A_px_p + B_px_s \\ \sum_{i=1}^{p} C_ix_i + A_sx_s \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_p \\ y_s \end{bmatrix} . \qquad (19)$$

If parallel computation is necessary, first let the ith PE compute y_i and C_ix_i, then send the computed C_ix_i to the other PEs. Each PE will then compute y_s with the computed C_ix_i from the other PEs. When the computation is finished, the ith PE will contain the data for subvectors y_i and y_s of $\boldsymbol{y}$, where $\boldsymbol{y} = P^{T}AP\boldsymbol{x}$.

The multiplication of the preconditioner with vector $\boldsymbol{x}$ is computed with equation (20), where $z_s = \sum_{i=1}^{p} \bar{F}_ix_i + x_s$. If parallel computation is necessary, the ith PE should be allowed to compute $\bar{F}_ix_i$ separately, first. When this process is finished, the computed $\bar{F}_ix_i$ are sent to the other PEs and each PE will proceed to compute $z_s = \sum_{i=1}^{p} \bar{F}_ix_i$ with the data from $\bar{F}_ix_i$ they have received from the other PEs. After this, the ith PE will compute $M_ix_i + \bar{E}_iz_s$ and M_sz_s:

$$\begin{bmatrix} M_1 & & & \bar{E}_1 \\ & \ddots & & \vdots \\ & & M_p & \bar{E}_p \\ & & & M_s \end{bmatrix} \begin{bmatrix} I_1 & & & \\ & \ddots & & \\ & & I_p & \\ \bar{F}_1 & \cdots & \bar{F}_p & I_s \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_s \end{bmatrix} = \begin{bmatrix} M_1x_1 + \bar{E}_1z_s \\ \vdots \\ M_px_1 + \bar{E}_pz_s \\ M_sz_s \end{bmatrix} \qquad (20)$$

The inner products of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are computed in the following manner: $\alpha = \sum_{i=1}^{p} \langle x_i, y_i \rangle + \langle x_s, y_s \rangle$. If parallel computation is necessary, first let the ith PE compute $\langle x_i, y_i \rangle$. The next step would be to compute $\langle x_s, y_s \rangle$ on one of the other PEs, for example the first one. The final step would be to gather the computed results and to sum them up to get α.

```
Algorithm 2: Procedure of numerical experiment.
  1. Compute the approximate inverse of coefficient matrix $A$ using the AISM on a single PE;
  2. Apply graph partitioning;
  3. Compute the approximate inverse of $P^TAP$ with the two level AISM on a single PE with $p = 2, 4, \dots, 16$;
  4. Compute the approximate inverse of $P^TAP$ with the two level parallel AISM on $p$ PEs ($p = 2, 4, \dots, 16$);
  5. Compare the performance of the two level parallel AISM versus the parallel MR (Minimal Residual) method [8] and the parallel ILU (Incomplete LU) decomposition;
```

```
Algorithm 3: Graph partitioning
  1. Construct a graph of the coefficient matrix. The output should then be transferred into a file according to the pmetis [13] format;
  2. Run pmetis to divide the graph into $p$ parts ($p = 2, 4, \dots, 16$);
```

5 Numerical experiments

We implemented the aforementioned algorithm on an Origin 2400 with MPI (Message Passing Interface) to illustrate its efficacy. The numerical experiments were carried out by Algorithm 2. The graph partitioning was carried out according to Algorithm 3.

Parameter s in the AISM was set to $1.5 \times \|A\|_\infty$ or $15 \times \|A\|_\infty$. The default value was $1.5 \times \|A\|_\infty$ [3, 15]. The tolerance of U and V was tolU = tolV = tol, where tol was set to 0.1 or 0.01. The default value was 0.1. When computing the approximate Schur complement, little elements were not dropped off since the computed matrices were still sparse enough in our experiments.

The initial matrix in the MR (Minimal Residual) method was set to zero and the inner iteration number was set to $n_i = 2$ or 5. Tolerance in the MR method was set to 0.1 or 0.01.

For the parallel application of the ILU decomposition, we used the parallel techniques proposed by Moriya et al. [14]. Let p be the number of PEs, and m be the number of row vectors of L and U covered by one PE. The lth PE holds the $((l-1)m+1)$th to (lm)th row vectors of L and U. The multiplication of preconditioner $(LU)^{-1}$ with any vector $\boldsymbol{z}$ is computed by solving the following two equations

$$L\tilde{\boldsymbol{z}} = \boldsymbol{z} \quad \text{and} \quad U\boldsymbol{w} = \tilde{\boldsymbol{z}}\,. \tag{21}$$

The lth PE only compute the $((l-1)m+1)$th to (lm)th elements of $\tilde{\boldsymbol{z}}$ and $\boldsymbol{w}$. After this step, $\tilde{\boldsymbol{z}}$ needs to be solved. The ith elements of $\tilde{\boldsymbol{z}}$ on the lth PE is computed with

$$\tilde{z}_i = \frac{1}{L_{i,i}} \left(z_i - \sum_{j=1}^{(l-1)m} L_{i,j}\tilde{z}_j - \sum_{j=(l-1)m+1}^{i-1} L_{i,j}\tilde{z}_j \right). \tag{22}$$

In order to compute the second part in between the parentheses (*) of the above equation (22), it was necessary to obtain data for $\tilde{z}_j$ $(j = 1, \ldots, (l-1)m)$ from the other PEs. Computations and communications were carried out in turns. If each PE sent their computed elements to the other PEs after the m elements were all computed, then the other PEs would have to be on standby mode for a long time. In order to reduce this waiting time, we let each PE send their computed $\tilde{m}\,(\leqslant m)$ elements to the other PEs. $\boldsymbol{w}$ in equation (21) was computed in the same way.

Further to this, we changed the value of $\tilde{m}$ and measured the computation time of equation (21) of the ILU(0) decomposition. The $\tilde{m}$ that achieved the shortest computation time was adopted.

We applied these three preconditioning techniques to the GMRES(m) [22, 23] to estimate the computed approximate inverse matrices. The initial approximate value of GMRES(m) was set to a zero vector and its convergence condition was set to

$$\|\boldsymbol{r}_k\|_2/\|\boldsymbol{b}\|_2 \leqslant 10^{-12}. \tag{23}$$

Table 2: Example 5.1: Results of graph partitioning for p.

p	$\dim(A_i)$	$\dim(A_s)$
2	18239/18239	386
4	9033/9024/9014/9013	780
6	5986/5892/5986/5992/5819/5975	1214
8	4384/4457/4365/4465/4439/4380/4448/4360	1566
10	3486/3557/3505/3446/3513/3456/3562/3478/ 3411/3536	1914
12	2864/2960/2898/2941/2852/2910/2846/2908/ 2926/2948/2843/2904	2064
14	2431/2522/2473/2489/2470/2467/2406/2514/ 2412/2466/2444/2516/2415/2477	2362
16	2173/2110/2198/2157/2155/2110/2202/2159/ 2156/2182/2083/2095/2168/2199/2169/2088	2460

The maximum iteration number was set to 20 000. The execution of each method was interrupted if the residual norm did not converge after 20 000 iterations. The restart cycle of the GMRES(m) was set to 30, 40 and 50.

All experiments were carried out on an Origin 2400 configured with 16 MIPS R12000 processors with a clock speed of 300 MHz and 8 GB of main memory, using double precision C and MPI-1.2. CPU times were all measured in seconds.

5.1 Example: elliptic PDE in the square

We studied the following boundary value problem of the elliptic partial differential equation in the unit square region $\Omega = [0,1]^2$:

$$-u_{xx} - u_{yy} + D\left\{(y-\tfrac{1}{2})u_x + (x-\tfrac{1}{3})(x-\tfrac{2}{3})u_y\right\} - 43\pi^2 u = G(x,y),$$
$$u(x,y)|_{\partial\Omega} = 1 + xy,$$

where $G(x, y)$ was chosen so that the exact solution was $u = 1 + xy$ on Ω. A five point central difference was applied, with uniform mesh spacing in each direction. Dh was set to 2^{-7}, where $h = 1/193$ was the mesh size. The dimension of the coefficient matrix was $36\,864$.

The first step was to compute the approximate inverse of coefficient matrix A with the AISM on a single PE. The computation time was 3799 seconds.

The second step was to construct a graph of the coefficient matrix and to partition it into p parts ($p = 2, 4, \ldots, 16$). The time required for graph partitioning was no more than one second. The results of the graph partitioning are tabulated in Table 2. The dimension of A_s is the same as that of the Schur complement. Table 2 shows that the subgraphs are approximately the same size. The dimension of A_i decreases as p increases, and the dimension of Schur complement increases as p increases.

Based on the data given in Table 2, we computed the value of p_{optS} and p_{optP} with equations (17) and (18) separately. Note that p_{OPTS} is the optimum value of p on a single processor, and p_{OPTP} is the optimum value of p in parallel computation. The results were: $p_{\text{OPTS}} = 16$ and $p_{\text{OPTP}} = 16$.

A two level AISM was run on a single PE to illustrate its efficiency. The data for the computation time and speed-up are shown in Table 3. The data indicates that a 14 optimal speed-up was achieved only on a single PE when $p = 16$. This coincided with the value of $p_{\text{optS}} = 16$ computed with equation (17).

The proposed two level parallel AISM was then applied on p PEs. The data for computation times and speed-ups are shown in Table 3. An optimal speed-up was achieved when 16 PEs were used. This, too, coincided with the value of $p_{\text{optP}} = 16$ computed with equation (18).

Finally, we compared the two level parallel AISM with the MR method and the ILU decomposition. Since the computation time of the two level parallel AISM with 16 PEs rendered the best results, we compared this with the performance of the MR method and the ILU decomposition of 16 PEs. The numerical results are shown in Table 4, where T_{pre} is the computation time for preconditioning.

Table 3: Example 5.1: Numerical results of the two level AISM on a single PE, and of the two level parallel AISM on p PEs.

	single PE		p PEs	
p	time	speed-up	time	speed-up
1	3800	1	3800	–
2	1703	2	850	4
4	818	5	207	18
6	551	7	95	40
8	427	9	58	65
10	361	11	43	89
12	316	12	34	113
14	295	13	31	124
16	277	14	28	137

For the MR and AISM, T_{pre} is the time required for computing their approximate inverses. For the ILU decomposition, T_{pre} is the time required for an incomplete decomposition. IT is the iteration number for the GMRES(m). T_{total} is the total computation time, including the preconditioning time, in seconds.

Table 4 indicates that the MR method converges only when $\text{tol} = 0.01$ and $\text{imax} = 3$. Although the computation time of the preconditioning for the AISM was longer than that of the ILU decomposition, there was no striking difference in their total computation times. When we applied the ILU decomposition using 16 PEs, it was necessary to determine the proper value of $\tilde{m}$ by changing it with various values and measuring the computation time of equation (21). This was clearly a more complicated process than the two level parallel AISM. In this example, the proper value of $\tilde{m}$ was $2\,304$.

Table 4: Example 5.1: Numerical results of GMRES(m) with 16 PEs; times are in seconds; IT denotes the number of iterations.

Preconditioner		T_{pre}	Iterative solver					
			GMRES(30)		GMRES(40)		GMRES(50)	
			IT	T_{total}	IT	T_{total}	IT	T_{total}
None		0.0	—	—	—	—	—	—
AISM								
tol	$s/\|A\|_\infty$							
0.1	1.5	27	12950	1255	10632	1129	8395	973
0.1	15	38	12237	1594	10727	1505	8748	1314
0.01	1.5	114	1637	801	1625	819	1222	650
0.01	15	166	1446	1100	1458	1137	1280	1017
ILU(0)		1	18258	1979	12164	1375	9536	1120
ILU(1)		1	5650	873	5114	816	4188	689
ILU(2)		2	2603	524	2990	616	2558	537
MR								
tol	imax							
0.1	1	200	—	—	—	—	—	—
0.1	2	379	—	—	—	—	—	—
0.1	3	560	—	—	—	—	—	—
0.01	1	200	—	—	—	—	—	—
0.01	2	381	—	—	—	—	—	—
0.01	3	558	18188	1292	17058	1331	10561	1086

- —, the residual norm could not converge within the maximum iterations.
- imax, inner iterations of the MR method.

Table 5: Example 5.2: Results of graph partitioning for p

p	$\dim(A_i)$	$\dim(A_s)$
2	24884/24869	2240
4	12288/11164/11702/11598	5241
6	6899/7065/7929/7567/7123/7530	7880
8	5570/5629/4984/5165/5497/5320/5580/5084	9164
10	3787/3745/4416/4050/4245/4240/4187/4375 4117/3864	10967
12	3520/3586/3243/3233/2973/3138/3349/3251 3489/3287/3194/3510	12220
14	2802/2386/2623/2627/3046/3051/2847/3012 2556/2931/2804/2917/2523/2822	13046
16	2599/2331/2590/2409/2278/2353/2214/2215 2322/2461/2387/2258/2307/2489/2460/2289	14031

5.2 Example: irregular coefficient matrix

Our second example was a study of a linear system of equations where the coefficient matrix was far more irregular than the coefficients in the previous one. We used matrix "ecl32" from the Florida Sparse Matrix Collection [7]. Its dimension is $51\,993$ with $380\,415$ nonzero elements. The right vector b was decided by letting the exact solution be $(1.0, 1.0, \ldots, 1.0)$.

The first step was to compute the approximate inverse of coefficient matrix A with the AISM on a single PE. The computation time was 4577 seconds.

Next, we carried out the graph partitioning in the same manner as Example 5.1. The time required for graph partitioning was no more than three seconds. The results of the graph partitioning are shown in Table 5. Table 5 suggests that the dimension of A_i decreases as p increases and the dimension of the Schur complement increases as p increases, which is similar to Example 5.1.

Table 6: Example 5.2: Numerical results of the two level AISM on a single PE, and of the two level parallel AISM on p PEs

	single PE		p PEs	
p	time	speed-up	time	speed-up
1	4578	1	4578	–
2	2265	2	1107	4
4	1438	3	351	13
6	1422	3	281	16
8	1489	3	271	17
10	1692	3	307	15
12	2023	2	370	12
14	2241	2	396	12
16	2651	2	447	10

The dimension of A_i was quite different from that of Example 5.1. The dimension of A_s, which is same as the dimension of the Schur complement, becomes larger when p increases. When $p = 16$, the dimension of the Schur complement is about seven times that of the dimension of A_i. This can be explained by the irregular distribution of the nonzero elements of A. Based on the data tabulated in Table 5, the results were $p_{\text{optS}} = 8$ and $p_{\text{optP}} = 8$ with equations (17) and (18).

A two level AISM was run on a single PE to illustrate its efficiency. The data for the computation time and speed-up are shown in Table 6. From this data we know that a optimal 3.2 speed-up was achieved only on a single PE when $p = 6$. The real p_{optS} was six. This is different when the value is computed with equation (17). As mentioned in section 3, equations (17) and (18) do not provide an exact estimation, since the computation cost deeply depends on the number of nonzero elements. In this example, the nonzero elements were much more irregular than in Example 5.1, so the p_{optS} computed with equation (17) was different from the actual calculation cost.

However, equation (17) was still useful in terms of finding the proper p.

We applied the proposed two level parallel AISM on p PEs. The data for the computation times and speed-ups are shown in Table 6. An optimal speed-up was achieved when eight PEs were used, which coincides with the value computed with equation (18).

Finally, we made a comparison of the two level parallel AISM, the MR method and the ILU decomposition. Since the computation time of the two level parallel AISM with eight PEs was the shortest, we compared the different methods using eight PEs. For this example, the proper value of $\tilde{m}$ in the ILU decomposition was $6\,500$.

The numerical results are tabulated in Table 7. Table 7 shows that the MR method does not converge. The total computation time of the ILU decomposition was shorter than the two level parallel AISM, but the computation time of the ILU decomposition depended heavily on the value of $\tilde{m}$. If an unsuitable $\tilde{m}$ was used, then the computation time increased significantly. For example, when we set $\tilde{m} = 813$, the total computation time was three times longer than that of the two level parallel AISM.

6 Concluding Remarks

We proposed a strategy for the parallel implementation of the AISM. Numerical experiments of two different linear systems of equations demonstrated that the proposed parallel implementation can perform effectively and underscores the efficiency of our proposed strategy as an effective scheme for parallelizing the AISM. The numerical results presented in this article suggest that our AISM is potentially a useful tool for obtaining solutions for large sparse linear systems of equations using modern high performance architectures. In addition, instead of computing the approximate inverse problem of the original matrix, the two level AISM computes several small approximate inverse problems. When the computing cost of these small problems is lower than the original

Table 7: Example 5.2: Numerical Results of the GMRES(m) with eight PEs; times are in seconds; IT denotes the number of iterations.

Preconditioner		T_{pre}	Iterative solver					
			GMRES(30)		GMRES(40)		GMRES(50)	
			IT	T_{total}	IT	T_{total}	IT	T_{total}
None		0.0	—	—	—	—	—	—
AISM								
tol	$s/\|A\|_\infty$							
0.1	1.5	271	2863	1014	2754	1052	1857	848
0.1	15	290	2508	1003	3623	1408	2009	960
0.01	1.5	310	1580	790	1926	944	2435	1173
0.01	15	335	2790	1284	1962	1050	1854	1057
ILU(0)		1	1354	285	1307	289	874	203
ILU(1)		6	1165	443	854	337	985	398
ILU(2)		19	1170	620	1185	638	728	408
MR								
tol	imax							
0.1	1	1023	—	—	—	—	—	—
0.1	2	1971	—	—	—	—	—	—
0.1	3	2920	—	—	—	—	—	—
0.01	1	1023	—	—	—	—	—	—
0.01	2	1972	—	—	—	—	—	—
0.01	3	2917	—	—	—	—	—	—

- —, the residual norm could not converge within the maximum iterations.
- imax, inner iterations of the MR method.

problem, running the two level AISM on a single processor will result in a significant speed-up. Further research is needed to improve its performance when the Schur complement is large and more numerical experiments are necessary for determining whether the approximate inverse of the Schur complement should be computed in parallel to maximize performance levels.

Acknowledgements This research was supported by the Fundamental Research Funds for the Central Universities (No.201013043), NSFC (No.10971204) and SRF for ROCS, SEM.

References

[1] Benzi, M., and Tůma, M., A Sparse Approximate Inverse Preconditioner for Nonsymmetric Linear Systems, *SIAM J. Sci. Comput.*, **19** (1998) 968–994. doi:10.1137/S1064827595294691 E9

[2] Benzi, M., Marín, J., and Tůma, M., A Two-level Parallel Preconditioner Based on Sparse Approximate Inverse, *Iterative Methods in Scientific Computation IV, D.R. Kincaid, A. C. Elster, eds. IMACS Series in Computational and Applied Mathematics*, IMACS, New Brunswick, NJ (1999) 167–178 . E3

[3] Bru, R., Cerdán, J., Marín, J., and Mas, J., Preconditioning Sparse Nonsymmetric Linear Systems with the Sherman–Morrison Formula, *SIAM J. Sci. Comput.*, **25** (2003) 701–715 . doi:10.113/S1064827502407524 E3, E5, E6, E9, E12

[4] Bruaset, A. M., *A Survey of Preconditioned Iterative Methods*, *Pitman Research Notes in Mathematics Series*, No. 328, Longman Scientific & Technical, U. K (1995). E2

[5] Chow, E., and Saad, Y., Approximate Inverse Techniques for Block-partitioned Matrices, *SIAM J. Sci. Comput.*, **18** (1997) 1657–1675 . doi:10.1137/S1064827595281575 E3

[6] Chow, E., and Saad, Y., Approximate Inverse Preconditioners via Sparse-sparse Iterations, *SIAM J. Sci. Comput.*, **19** (1997) 995–1023(1997) doi:10.1137/S1064827594270415 E3

[7] Davis, T., University of Florida Sparse Matrix Collection. *NA Digest, 92, 1994*, `http://www.cise.ufl.edu/research/sparse/matrices` E18

[8] Grote, M., and Huckel, T.: Parallel Preconditioning with Sparse Approximate Inverses, *SIAM J. Sci. Comput.*, **18** (1997) 838–853 . doi:10.1137/S1064827594276552 E12

[9] Heath, M. T., Ng, E., and Peyton, B. W., Parallel Algorithms for Sparse Linear Systems, *SIAM Review.*, **33** (1990) 420–460. doi:10.1137/1033099 E3

[10] Hager, W. W., Updating the Inverse of a Matrix, *SIAM Rev.*, **31** (1989) 221–239, . doi:10.1137/1031049 E4

[11] Huckel, T., Efficient Computation of Sparse Approximate Inverses, *Numer. Lin. Alg. Appl.*, **5** (1998) 57–71. doi:10.1002/(SICI)1099-1506(199801/02)5:13.0.CO;2-C E3

[12] Huckel, T., Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning, *Appl. Numer. Math.*, **30** (1999) 291–303. doi:10.1016/S0168-9274(98)00117-2 E3

[13] Karypis, G., and Kumar, V., A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, *SIAM J. Sci. Comput.*, **20** (1998) 359–392. doi:10.1137/S1064827595287997 E12

[14] Moriya, K. and Nodera, T, The Parallelization of Preconditioner for Large Sparse Linear Systems of Equations, *Bulletin of the JSIAM* (in Japanese), **12** (2002) 14–28. E13

[15] Moriya, K, Zhang, L. and Nodera, T., An Approximate Matrix Inversion Procedure by Parallelization of the Sherman–Morrison formula, The *ANZIAM J.*, **51** (2009) 1–9. doi:10.1017/S1446181109000364 E3, E12

[16] Moriya, K, Zhang, L. and Nodera, T., Efficient Approximate Inverse Preconditioning Techniques for Reduced Systems on Parallel Computers, *Substracting Techniques and Domain Decomposition Method*, Edited by: F. Magoules, Saxe-Coburg Pub. (2010) 203–228. doi:10.4203/csets.24.8 E3

[17] Moriya, K. and Nodera, T., A New Scheme of Computing the Approximate Inverse Preconditioner for the Reduced Linear Systems, *J. of Comp. and Appl. Math.*, **199** (2007) 345–352. doi:10.1016/j.cam.2005.08033 E3

[18] Sherman, J. and Morrison, W., Adjustment of an Inverse Matrix, Corresponding to a Change in One Element of a Given Matrix, *Ann. Math. Statist.*, **21** (1950) 124–127. doi:10.1244/aoms/1177729893 E4

[19] Naik, V. K., A Scalable Implementation of the NAS Benchmark BT on Distributed Memory Systems, *IBM Systems Journal*, **34** (1995) 273–291. doi:10.114/sj.342.0273 E3

[20] Saad, Y., *Iterative Methods for Sparse Linear Systems*, 2nd Ed. *SIAM* (2003). doi:10.1137/1.9780898718003 E2, E3

[21] Saad, Y., Preconditioning Techniques for Nonsymmetric and Indefinite Linear Systems, *J. Comput. Appl. Math.*, **24** (1988) 89–105. doi:10.1016/0377-0427(88)90345-7 E2

[22] Saad, Y., and Schultz, M. H., GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Statist. Comput.*, **7**(1986) 856–869 . doi:10.1137/0907058 E2, E13

[23] Shimoncini, V., On the Convergence of Restarted Krylov Subspace Method, *SIAM J. Matrix Anal. Appl.*, **22** (2000) 430–452. doi:10.1137/S0895479898348507 E13

[24] Simoncini, V. and Szyld, D. B., Recent Computational Developments in Krylov Subspace Methods for Linear Systems, *Numer. Lin. Alg. Appl.*, **14** (2007) 1–59. doi:10.1002/nla.499 E2

[25] Van der Vorst, H., *Iterative Krylov Methods for Large Linear Systems*, Cambridge University Press (2003). doi:10.1017/CBO9780511615115 E2

Author addresses

1. **L. Zhang**, College of Mathematical Sciences, Ocean University of China, 23 XiangGangDongLu Road, Qingdao, ShanDong, 266071, China.
   `mailto:zhanglinjie@hotmail.com`
2. **K. Moriya**, Ohi-Branch, Nikon System Inc., Japan.
   `mailto:kmoriya@nikon-sys.co.jp.`
3. **T. Nodera**, Department of Mathematics, Faculty of Science and Technology, Keio University, 3-14-1 Hiyoshi, Kohoku, Yokohama 223-8522, Japan.
   `mailto:nodera@math.keio.ac.jp`