ANZIAM J. 51(2009), 1–9
doi:10.1017/S1446181109000364

AN APPROXIMATE MATRIX INVERSION PROCEDURE BY PARALLELIZATION OF THE SHERMAN–MORRISON FORMULA

KENTARO MORIYA ✉[1], LINJIE ZHANG[2] and TAKASHI NODERA[3]

(Received 11 February, 2009; revised 5 November, 2009)

Abstract

The Sherman–Morrison formula is one scheme for computing the approximate inverse preconditioner of a large linear system of equations. However, parallelizing a preconditioning approach is not straightforward as it is necessary to include a sequential process in the matrix factorization. In this paper, we propose a formula that improves the performance of the Sherman–Morrison preconditioner by partially parallelizing the matrix factorization. This study shows that our parallel technique implemented on a PC cluster system of eight processing elements significantly reduces the computational time for the matrix factorization compared with the time taken by a single processor. Our study has also verified that the Sherman–Morrison preconditioner performs better than ILU or MR preconditioners.

1. Introduction

In this study we consider the solution of large and sparse linear systems of equations

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{1.1}$$

where the coefficient matrix A is of the order of $n \times n$ and the solution vector $\boldsymbol{x}$ and right-hand side $\boldsymbol{b}$ are $n \times 1$ vectors. Since iterative methods often require a large number of iterations to solve a linear system of equations (1.1), it is usually necessary to use some kind of a preconditioning technique. Recently, Bru *et al.* [1] proposed computing a preconditioner based on the Sherman–Morrison formula, and showed that it consistently performed well. However, it is not an easy task to parallelize the computation of the preconditioner in such a formula, since the kth column vector depends on the first $k-1$ column vectors in the matrix factorization.

[1]Head Office, Nikon System Inc., Japan; e-mail: kmoriya@nikon-sys.co.jp.
[2]College of Mathematical Sciences, Ocean University of China, Japan; e-mail: zhanglinjie@hotmail.com.
[3]Department of Mathematics, Faculty of Science and Technology, Keio University, Japan; e-mail: nodera@math.keio.ac.jp.

In this study, we propose a technique that partially parallelizes the computation of this preconditioner. Two test problems demonstrate that the parallel performance of our technique implemented on a PC cluster system is significantly faster than the solution of the problems by the use of a single processor. We also demonstrate that our technique can accelerate the convergence of the residual norm.

2. The Sherman–Morrison preconditioner

In the recurrence equation,

$$A_k = A_{k-1} + \boldsymbol{p}_k \boldsymbol{q}_k^{\mathrm{T}}, \tag{2.1}$$

where A_k and A_{k-1} are nonsingular matrices and $\boldsymbol{p}_k$, $\boldsymbol{q}_k$ are nonzero vectors. When the Sherman–Morrison formula is applied to Equation (2.1),

$$A_k^{-1} = A_{k-1}^{-1} - r_k^{-1} A_{k-1}^{-1} \boldsymbol{p}_k \boldsymbol{q}_k^{\mathrm{T}} A_{k-1}^{-1}, \tag{2.2}$$

where

$$r_k = 1 + \boldsymbol{q}_k^{\mathrm{T}} A_{k-1}^{-1} \boldsymbol{p}_k. \tag{2.3}$$

Summing Equation (2.2) from $k = 1$ to $k = n$ gives

$$A^{-1} = A_0^{-1} - \sum_{k=1}^{n} r_k^{-1} A_{k-1}^{-1} \boldsymbol{p}_k \boldsymbol{q}_k^{\mathrm{T}} A_{k-1}^{-1}, \tag{2.4}$$

where $A^{-1} = A_n^{-1}$. Equation (2.4) can be expressed as

$$A_0^{-1} - A^{-1} = \Psi \Omega^{-1} \Xi^{\mathrm{T}}, \tag{2.5}$$

where

$$\Psi = \left[A_0^{-1}\boldsymbol{p}_1,\ A_1^{-1}\boldsymbol{p}_2, \ldots, A_{n-1}^{-1}\boldsymbol{p}_n, \right], \quad \Omega^{-1} = \operatorname{diag}\left[r_1^{-1},\ r_2^{-1}, \ldots, r_n^{-1} \right]$$

and

$$\Xi = \left[\boldsymbol{q}_1^{\mathrm{T}} A_0^{-1},\ \boldsymbol{q}_2^{\mathrm{T}} A_1^{-1}, \ldots, \boldsymbol{q}_n^{\mathrm{T}} A_{n-1}^{-1} \right].$$

Define

$$\boldsymbol{u}_k := \boldsymbol{p}_k - \sum_{i=1}^{k-1} \frac{\boldsymbol{v}_i^{\mathrm{T}} A_0^{-1} \boldsymbol{p}_k}{r_i} \boldsymbol{u}_i, \quad \boldsymbol{v}_k := \boldsymbol{q}_k - \sum_{i=1}^{k-1} \frac{\boldsymbol{q}_k^{\mathrm{T}} A_0^{-1} \boldsymbol{u}_i}{r_i} \boldsymbol{v}_i, \tag{2.6}$$

so that $A_{k-1}^{-1}\boldsymbol{p}_k = A_0^{-1}\boldsymbol{u}_k$ and $\boldsymbol{q}_k^{\mathrm{T}} A_{k-1}^{-1} = \boldsymbol{v}_k^{\mathrm{T}} A_0^{-1}$ are satisfied. Substituting these two relations into Equations (2.3) and (2.5),

$$r_k = 1 + \boldsymbol{q}_k^{\mathrm{T}} A_0^{-1} \boldsymbol{u}_k \tag{2.7}$$

and

$$A_0^{-1} - A^{-1} = A_0^{-1} U \Omega^{-1} V^{\mathrm{T}} A_0^{-1}, \tag{2.8}$$

respectively, where $U = (\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n)$ and $V = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$. Note that A_0, $\boldsymbol{p}_k$ and $\boldsymbol{q}_k$ are still included in the relations (2.6), and these values have to be determined. The matrix $A_0^{-1} - A^{-1}$ should approximate the inverse of A. According to Bru *et al.* [1], the simplest method for achieving this approximation is to take

$$A_0 = sI, \quad \boldsymbol{p}_k = \boldsymbol{e}_k, \quad \boldsymbol{q}_k = (\boldsymbol{a}^k - s\boldsymbol{e}_k)^{\mathrm{T}}, \tag{2.9}$$

where $\boldsymbol{a}^k$ and $\boldsymbol{e}_k$ are the kth column vectors of A and the kth entry of the identity vector is 1.0, respectively. Bru *et al.* [1] demonstrated that $s = 1.5\times \|A\|_\infty$ is a suitable choice for this scalar in (2.9). If we adopt this selection approach (2.9), Equations (2.6) and (2.7) are transformed to

$$\boldsymbol{u}_k = \boldsymbol{p}_k - \sum_{i=1}^{k-1} \frac{(v_i)_k}{sr_i} \boldsymbol{u}_i, \tag{2.10}$$

$$\boldsymbol{v}_k = \boldsymbol{q}_k - \sum_{i=1}^{k-1} \frac{\boldsymbol{q}_k^{\mathrm{T}} \boldsymbol{u}_i}{sr_i} \boldsymbol{v}_i, \tag{2.11}$$

$$r_k = 1 + \frac{(\boldsymbol{v}_k)_k}{s}, \tag{2.12}$$

where $(\boldsymbol{v}_k)_k$ is the kth entry of $\boldsymbol{v}_k$. If we set the left-hand side of Equation (2.8) to be equal to the preconditioner M^{-1}, the formula for computing the preconditioner is then

$$M^{-1} = s^{-1}I - A^{-1} = s^{-2}U\Omega^{-1}V^{\mathrm{T}}. \tag{2.13}$$

Since M^{-1} is the product of U, Ω^{-1} and V^{T}, it can be computed by using Equations (2.10)–(2.12). From Equation (2.13) it is clear that M^{-1} differs from A^{-1} only in its diagonal entries. To reduce computational costs, not all of the nonzero entries of U and V are computed; significantly small entries are usually neglected.

3. Parallel implementation

3.1. Parallel matrix decomposition The drawback of computing M^{-1} using Equation (2.13) is that the computation of $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ where $k = 2, \ldots, n$ is dependent on the vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{k-1}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1}$, and the scalars $r_1, \ldots, r_{k-1}$. The parallelization of these computations is therefore not straightforward and so we propose a parallel technique that computes Equations (2.10) and (2.11).

3.2. Allocation of $\boldsymbol{u}_k$, $\boldsymbol{v}_k$ and r_k to PEs Assuming that the orders of matrices U, Ω and V are n and the number of processing elements (PEs) is d, the $(ml+1)$th to $(l+1)m$th column vectors of U, V and diagonal entries of Ω^{-1} are allocated to the lth PE, where m is the number of column vectors and diagonal entries are covered by one PE. In the case of the matrix U, for instance, the allocation is given by

$$U = \{\underbrace{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m}_{\mathrm{PE}_0}, \underbrace{\boldsymbol{u}_{m+1}, \ldots, \boldsymbol{u}_{2m}}_{\mathrm{PE}_1}, \ldots\ldots, \underbrace{\boldsymbol{u}_{n-m+1}, \ldots, \boldsymbol{u}_n}_{\mathrm{PE}_{d-1}}\},$$

where PE_l is the lth PE for $l = 0, \ldots, d-1$.

An additional assumption we have made is that n can be divided by d, with $m = n/d$. Note, however, that we can also proceed when n is not divisible by d. This is solved through an allocation where the zeroth to $(\tilde{d}-1)$th PEs cover $\lfloor n/d \rfloor + 1$, and the $\tilde{d}$th to $(d-1)$th PEs cover $\lfloor n/d \rfloor$ column vectors and diagonal entries, where $\tilde{d} = n \bmod d$ and $\lfloor n/d \rfloor$ is the integer part of n/d.

3.3. Parallel implementation of the preconditioner From Equations (2.10) and (2.11) we know that $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ depend on $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{k-1}$, $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1}$ and $r_1, \ldots, r_{k-1}$. Owing to this dependency, under normal circumstances the lth PE is unable to start computing these vectors until the zeroth to $(l-1)$th PEs have finished computing. However, the computation of Equations (2.10) and (2.11) can be partially parallelized. These two equations can be separated into three terms:

$$\boldsymbol{u}_k = \boldsymbol{p}_k - \sum_{i=1}^{lm} \frac{(v_i)_k}{sr_i}\boldsymbol{u}_i - \sum_{i=lm+1}^{k-1} \frac{(v_i)_k}{sr_i}\boldsymbol{u}_i, \quad \forall k = lm+1, \ldots, (l+1)m, \tag{3.1}$$

$$\boldsymbol{v}_k = \boldsymbol{q}_k - \sum_{i=1}^{lm} \frac{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{u}_i}{sr_i}\boldsymbol{v}_i - \sum_{i=lm+1}^{k-1} \frac{\boldsymbol{q}_k^{\mathrm{T}}\boldsymbol{u}_i}{sr_i}\boldsymbol{v}_i, \quad \forall k = lm+1, \ldots, (l+1)m. \tag{3.2}$$

The second term of the right-hand sides of Equations (3.1) and (3.2) cannot be computed until values are received from the other PEs, but the third term can be computed without values from any other PE. The problem is that the computation of the third term cannot be started until all of the vectors included in the second term have been received. If PE_0 transfers $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ to the other PEs after computing all of these vectors, it cannot work with the other PEs simultaneously. However, if PE_0 computes only $\tilde{m}$ $(<m)$ column vectors, it can work with the other PEs in parallel after transferring these vectors, since PE_0 still has the remaining column vectors to compute. During the first step, while PE_0 is computing the first to $\tilde{m}$th column vectors, the other PEs are idle.

During the second step, however, PE_0 computes the $(\tilde{m}+1)$th to $2\tilde{m}$th column vectors and transfers them while the other PEs update their covering column vectors $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ using the column vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{\tilde{m}}$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{\tilde{m}}$, received from PE_0 during the first step. During step three, while PE_0 is computing the $(2\tilde{m}+1)$th to $3\tilde{m}$th columns, the other PEs are updating their covering column vectors using the column vectors $\boldsymbol{u}_{\tilde{m}+1}, \ldots, \boldsymbol{u}_{2\tilde{m}}$ and $\boldsymbol{v}_{\tilde{m}+1}, \ldots, \boldsymbol{v}_{2\tilde{m}}$ received from PE_0 during step two. Subsequent steps follow a similar pattern. Defining $G_k = \{\boldsymbol{u}_k, \boldsymbol{v}_k, r_k\}$, $G_1, \ldots, G_{k-1}$ are required for the computation of G_k. Based on this, G_k $(k = 1, \ldots, n)$ can be computed in parallel as shown in Figure 1.

When PE_0 has finished computing its covering G_k, PE_1 then works in the same way as PE_0, and transfers its covering G_k to the PEs from PE_2 to PE_{d-1}. During this stage, although PE_0 is idle, the other PEs can work in parallel. Here, $\tilde{m}$ is defined as the block number, and the appropriate block number will be dependent on the specifications of the system. The scheme of parallelization is shown in Figure 2, where tolU and tolV are the thresholds. Entries of $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ lower than these thresholds are neglected.

PE	Step 1	Step 2	Step 3	Step 4
0	$G_1, \ldots, G_{\tilde m}$ are computed and transferred.	$G_{\tilde m+1}, \ldots, G_{2\tilde m}$ are computed and transferred.	$G_{2\tilde m+1}, \ldots, G_{3\tilde m}$ are computed and transferred.	$G_{3\tilde m+1}, \ldots, G_{4\tilde m}$ are computed and transferred.
$1, \ldots, d-1$	Idle	All the allocated G_k are updated using $G_1, \ldots, G_{\tilde m}$.	All the allocated G_k are updated using $G_{\tilde m+1}, \ldots, G_{2\tilde m}$.	All the allocated G_k are updated using $G_{2\tilde m+1}, \ldots, G_{3\tilde m}$.

FIGURE 1. A parallel scheme for computing G_k.

The parallel implementation

```
1-1:  "<<1: computing the first term of Equations (3.1) and (3.2)>>"
1-2:  for $k = lm + 1$ to $(l + 1)m$ do
1-3:      $\boldsymbol{p}_k = \boldsymbol{e}_k$, $\boldsymbol{q}_k = (\boldsymbol{a}^k - s\boldsymbol{e}_k)^T$, $\boldsymbol{u}_k = \boldsymbol{p}_k$, $\boldsymbol{v}_k = \boldsymbol{q}_k$
1-4:  endfor
2-1:  "<<2: computing the second term of Equations (3.1) and (3.2)>>"
2-2:  for $i = 0$ to $l - 1$ do
2-3:      for $k = mi + 1$ to $m(i + 1)$ do
2-4:          if mod $(k,\ \tilde m) = 1$ then
2-5:              Receive $\boldsymbol{u}_j,\ \boldsymbol{v}_j,\ r_j,\ (j = k, \ldots, k + \tilde m - 1)$ from PE$_i$
2-6:          endif
2-7:          $\boldsymbol{u}_k = \boldsymbol{u}_k - \{(v_i)_k/(sr_i)\}\boldsymbol{u}_i, \quad \boldsymbol{v}_k = \boldsymbol{v}_k - \{\boldsymbol{q}_k^T \boldsymbol{u}_i/(sr_i)\}\boldsymbol{v}_i$
2-8:      endfor
2-9:  endfor
3-1:  "<< 3: computing the third term of Equations (3.1) and (3.2)>>"
3-2:  for $k = lm + 1$ to $(l + 1)m$ do
3-3:      for $i = lm + 1$ to $k - 1$ do
3-4:          $\boldsymbol{u}_k = \boldsymbol{u}_k - \{(v_i)_k/(sr_i)\}\boldsymbol{u}_i, \quad \boldsymbol{v}_k = \boldsymbol{v}_k - \{\boldsymbol{q}_k^T \boldsymbol{u}_i/(sr_i)\}\boldsymbol{v}_i$
3-5:      endfor
3-6:      for $i = 1$ to $i = n$ do
3-7:          if $|(u_k)_i| <$ tolU   dropoff $|(u_k)_i|$
3-8:          if $|(v_k)_i| <$ tolV   dropoff $|(v_k)_i|$
3-9:      endfor
3-10:     $r_k = 1 + (v_k)_k/s$
3-11:     if mod $(k,\ \tilde m) = 0$ then
3-12:         Send $\boldsymbol{u}_i,\ \boldsymbol{v}_i,\ r_i (i = k - \tilde m + 1, \ldots, k)$ to PE$_{l+1}, \ldots,$ PE$_{d-1}$
3-13:     endif
3-14: endfor
```

FIGURE 2. The parallel implementation of the Sherman–Morrison preconditioner for the lth PE.

4. Numerical experiments

We used a PC cluster system with four nodes configured with the Linux Fedora Core 4 operating system. Each node was configured with two Pentium Xeon 3.6 GHz processors and 1 GB memory. The total number of PEs was eight. The MPI communication library was used. The main experiments were used to analyse the parallel performance of the computation of the Sherman–Morrison preconditioner

TABLE 1. The relationship between the number of blocks and the computation time of the Sherman–Morrison preconditioner for Example 1. The column containing underlined computation times shows the optimum value of $\tilde{m}$.

	$\tilde{m}$	4608	2304	1152	576	288	144	72	36
time (s)	(a) tolU, tolV = 0.1	67	51	49	48	48	48	48	$\underline{48}$
	(b) tolU, tolV = 0.01	471	365	325	318	317	317	318	$\underline{312}$

based on the proposed technique, and test the convergence of the preconditioned GMRES(m) algorithm [3]. Regarding the convergence test, as well as the present preconditioner, we also applied the ILU factorization and the MR method [2] to the GMRES(m) algorithm in order to compare the performance of the present preconditioner with these alternative preconditioners. The stopping criterion was

$$\frac{\|\boldsymbol{r}_i\|_2}{\|\boldsymbol{b}\|_2} < 10 \times 10^{-12}, \tag{4.1}$$

where $\boldsymbol{r}_i$ was the ith step of the GMRES(m) algorithm. We parallelized the implementations of all of the preconditioners as well as the GMRES(m) algorithm.

EXAMPLE 1. Consider the boundary value problem described by Simoncini [4] given by the solution of a partial differential equation

$$-\frac{d}{dx}\left[\{\exp(-xy)\}u_x\right] - \frac{d}{dy}\left[\{\exp(xy)\}u_y\right] + 10.0(u_x + u_y) - 60.0u = f(x, y),$$
$$u(x, y)|_{\partial\Theta} = 1 + xy,$$

in the region $\Theta = [0, 1]^2$. We discretized this problem using a five-point difference scheme with 192^2 grid points to obtain a linear system of order 36,864. In the initial experiments we determined the computation time of the Sherman–Morrison preconditioner for this system for a selection of values of $\tilde{m}$.

The data in Table 1 shows that the computation of the Sherman–Morrison preconditioner is most efficient when $\tilde{m} = 36$. Based on this, we set $\tilde{m}$ to 36 in the following experiments. We also used a different number of PEs to analyse the parallel performance of this preconditioner which was computed using this technique.

Table 2 shows that a speedup of around a factor of six or seven is possible when eight PEs are used, where $\tilde{s} = s/(1.5\,\|A\|_\infty)$ and $s = 1.5\,\|A\|_\infty$ or $s = 1.5\,\|A\|_\infty \times 10$. See Table 3 for the convergence test of the preconditioned GMRES(m) algorithm. Note that the computation time of the GMRES(m) algorithm includes the preconditioning time.

In Table 3, the parameters "tol" and "imax" are the thresholds of the dropoff and iteration of the MR method, respectively. When using the ILU or MR preconditioners,

TABLE 2. The parallel performance of the computation time of the Sherman–Morrison preconditioner for Example 1.

	Number of PEs			
$\tilde{s}$	1	2	4	8
10^0	1.00	1.95	3.83	7.01
10^1	1.00	1.85	3.45	6.11

TABLE 3. The computation time of the iterations required to satisfy criterion (4.1) for Example 1. SM represents the Sherman–Morrison preconditioner and imax the number of iterations of the MR method. The computation time of GMRES(m) algorithm includes the preconditioning time and "–" denotes cases where the residual norm could not converge within two hours.

Preconditioner		Iterative solver					
		GMRES(20)		GMRES(30)		GMRES(40)	
Name	Time	Time	Iter	Time	Iter	Time	Iter
SM(tolU, tolV $= 0.1$, $\tilde{s} = 10^0$)	48	–	–	200	12514	111	4453
SM(tolU, tolV $= 0.1$, $\tilde{s} = 10^1$)	87	92	341	94	465	95	477
SM(tolU, tolV $= 0.01$, $\tilde{s} = 10^0$)	315	346	1160	331	565	327	382
SM(tolU, tolV $= 0.01$, $\tilde{s} = 10^1$)	560	605	1075	578	425	578	396
ILU(0)	0.24	–	–	–	–	–	–
ILU(1)	0.35	–	–	–	–	–	–
ILU(2)	0.48	–	–	–	–	28	934
MR(tol $= 0.1$, imax $= 2$)	60	–	–	–	–	–	–
MR(tol $= 0.1$, imax $= 1$)	25	–	–	–	–	–	–
MR(tol $= 0.01$, imax $= 2$)	65	–	–	159	10616	–	–
MR(tol $= 0.01$, imax $= 1$)	29	–	–	–	–	87	6021

the GMRES(m) algorithm sometimes converged much faster than was the case when using the Sherman–Morrison preconditioner. However, since in most cases these two preconditioners were ineffective, their use did not lead to a converged solution. On the other hand, with only one exception, the GMRES(m) algorithm with the Sherman–Morrison preconditioner converged. Therefore, it seems that this preconditioner performs more robustly than the other two preconditioners.

EXAMPLE 2. We studied two kinds of linear systems, coefficient matrices of which are described in Table 4 (see also [5]).

In both of these linear systems, all of the entries of the right-hand side $\boldsymbol{b}$ were 1.0. As in Example 1, we measured the appropriate $\tilde{m}$, analysed the parallel performance and compared the performance of the three kinds of preconditioners by using the GMRES(m) algorithm. The results are presented in Tables 5–7.

TABLE 4. The properties of the coefficient matrices in Example 2.

Name	Nonzeros	Order	Feature
ecl32	347,097	51,993	Circuit simulation
af23560	460,456	23,560	Transient stability analysis of a Navier–Stokes solver

TABLE 5. The relationship between the number of blocks and the computation time of the Sherman–Morrison preconditioner in Example 2. Columns containing an underlined computation time show the optimum value of $\tilde{m}$ for a given case.

(a-1). ecl32, tolU, tolV $= 0.1$								
$\tilde{m}$	6500	3250	1625	813	407	204	102	51
Time (s)	241	175	<u>157</u>	159	160	161	161	161
(a-2). ecl32, tolU, tolV $= 0.01$								
$\tilde{m}$	6500	3250	1625	813	407	204	102	51
Time (s)	1371	1090	942	901	<u>896</u>	902	911	913
(b-1). af23560, tolU, tolV $= 0.1$								
$\tilde{m}$	2945	1473	737	369	185	93	47	24
Time (s)	1477	1196	1059	1007	<u>989</u>	1003	1027	1027

TABLE 6. The parallel performance of the computation time of the Sherman–Morrison preconditioner in Example 2.

	Number of PEs			
$\tilde{s}$	1	2	4	8
10^0	1.00	1.67	2.84	4.45
10^1	1.00	1.60	2.72	4.37

Note that for the case tolU, tolV $= 0.01$ for the system af23560, the preconditioner did not converge within two hours, and thus its results are not shown in Table 5. Table 6 shows the parallel performance of the Sherman–Morrison preconditioner when using the af23560 system. As can be seen, computing this preconditioner using eight PEs is about 4.5 times faster than when using a single PE. Table 7 also shows the computation time and iterations for the preconditioned GMRES(m) algorithm. However, since this iterative solver with the ILU or MR preconditioner did not converge in every single case considered, no results are shown for these preconditioners. In contrast, the GMRES(m) algorithm using the Sherman–Morrison preconditioner converged when using the af23560 system where tolU, tolV $= 0.1$, and when using the ecl32 system

TABLE 7. The computation time of the iterations required to satisfy (4.1) for Example 2. SM denotes the Sherman–Morrison preconditioner and "–" denotes cases where the residual norm could not converge within two hours. The computation time of the GMRES(m) algorithm includes the preconditioning time.

	Preconditioner		Iterative solver					
			GMRES(20)		GMRES(30)		GMRES(40)	
	Name	Time	Time	Iter	Time	Iter	Time	Iter
(a) ecl32	SM(tolU, tolV $= 0.1, \tilde{s} = 10^0$)	157	–	–	–	–	–	–
	SM(tolU, tolV $= 0.1, \tilde{s} = 10^1$)	234	–	–	–	–	–	–
	SM(tolU, tolV $= 0.01, \tilde{s} = 10^0$)	956	1016	797	988	408	971	231
	SM(tolU, tolV $= 0.01, \tilde{s} = 10^1$)	1343	1427	819	1387	415	1365	221
(b) af23560	SM(tolU, tolV $= 0.1, \tilde{s} = 10^0$)	1068	1161	759	1147	621	1138	549
	SM(tolU, tolV $= 0.1, \tilde{s} = 10^1$)	2175	2311	740	2285	649	2266	544

where tolU, tolV $= 0.01$. Thus, it can be seen that only this particular preconditioner is effective for the convergence of the residual norm.

5. Conclusions

A technique that allows the parallel implementation of the Sherman–Morrison preconditioner has been explored in this study. Application of this technique to two numerical test problems demonstrated that the computation of this preconditioner by this technique using eight PEs was 4.5–7 times faster than when using a single PE. The numerical results of the convergence test also showed that the Sherman–Morrison preconditioner often performed more effectively than either the ILU or MR preconditioners.

References

[1] R. Bru, J. Credán, J. Marín and J. Mas, "Preconditioning sparse nonsymmetric linear systems with the Sherman–Morrison formula", *SIAM J. Sci. Comput.* **25** (2003) 701–715.

[2] M. Grote and T. Huckel, "Parallel preconditioning with sparse approximate inverses", *SIAM J. Sci. Comput.* **18** (1997) 838–853.

[3] Y. Saad and M. K. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM J. Sci. Stat. Comput.* **7** (1986) 856–869.

[4] Y. Simoncini, "On the convergence of restarted Krylov subspace method", *SIAM J. Matrix Anal. Appl.* **22** (2000) 430–452.

[5] "University of Florida sparse matrix collection", online http://www.cise.ufl.edu/research/sparse/matrices.