Journal of Computational Mathematics, Vol.20, No.6, 2002, 561–574.

EXPERIMENTAL STUDY OF THE ASYNCHRONOUS MULTISPLITTING RELAXATION METHODS FOR THE LINEAR COMPLEMENTARITY PROBLEMS*1)

Zhong-zhi Bai
(*State Key Laboratory of Scientific/Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing 100080, China*)

Abstract

We study the numerical behaviours of the relaxed asynchronous multisplitting methods for the linear complementarity problems by solving some typical problems from practical applications on a real multiprocessor system. Numerical results show that the parallel multisplitting relaxation methods always perform much better than the corresponding sequential alternatives, and that the asynchronous multisplitting relaxation methods often outperform their corresponding synchronous counterparts. Moreover, the two-sweep relaxed multisplitting methods have better convergence properties than their corresponding one-sweep relaxed ones in the sense that they have larger convergence domains and faster convergence speeds. Hence, the asynchronous multisplitting unsymmetric relaxation iterations should be the methods of choice for solving the large sparse linear complementarity problems in the parallel computing environments.

Key words: Linear complementarity problem, Matrix multisplitting, Asynchronous iterative methods, Numerical experiment.

1. Introduction

Given a matrix $M = (m_{kj}) \in \mathbb{R}^{n\times n}$ and a vector $q = (q_k) \in \mathbb{R}^n$, the linear complementarity problem LCP(M, q) is to find a vector $z \in R^n$ such that

$$Mz + q \geq 0, \quad z \geq 0 \quad \text{and} \quad z^T(Mz + q) = 0.$$

Recently, many practical and efficient parallel iterative methods in the sense of matrix multisplitting were proposed for solving the LCP(M, q) on the high-speed multiprocessor systems, and the convergence properties of these methods were studied in depth for some standard matrix classes. For details we refer to [4] and references therein. In original, these methods were developed from the matrix multisplitting iterative methods for the system of linear equations (see Bai [1] and Bai, Sun and Wang [7]), as well as from the sequential iterative methods for solving the linear complementarity problems (see Cottle, Pang and Stone [9]).

Based on several splittings of the system matrix $M \in \mathbb{R}^{n\times n}$, the LCP$(M, q)$ can be decomposed into independent linear complementarity problems of smaller sizes. Through solving these sub-problems in parallel on the multiprocessor system without any communication barrier, Bai and Huang [6] presented an asynchronous multisplitting iterative method. This asynchronous multisplitting iterative method was established in accordance with the principle of using sufficiently and communicating flexibly the known information. Hence, it has the potential to achieve high parallel computing efficiency in actual computations. For the convenience

* Received May 5, 1999.
1) Subsidized by the Special Funds for Major State Basic Research Projects G1999032803 and Supported by the National Natural Science Foundation of China (19601036).

of real applications, Bai and Huang [6] further presented an explicit alternative, called the asynchronous multisplitting unsymmetric AOR method, of the above mentioned asynchronous multisplitting iterative method, by making use of the overrelaxation and acceleration techniques. This asynchronous multisplitting unsymmetric AOR method includes two relaxation sweeps within each of its iterations, and each sweep possibly includes its own pair of relaxation parameters. Therefore, it can cover a series of relaxed asynchronous multisplitting methods for solving the LCP(M, q). Moreover, a numerical example was given in [6] to show that these relaxed asynchronous multisplitting methods are quite efficient for solving the large sparse linear complementarity problems on the high-speed multiprocessor systems.

In this paper, we further study the numerical behaviours of the relaxed asynchronous multisplitting methods by solving some typical problems from practical applications. For various choices of the relaxation parameters and in both of sequential and parallel settings, a variety of experiments were implemented for the asynchronous multisplitting unsymmetric AOR method and its synchronous and sequential alternatives, as well as some of their typical cases from special choices of the relaxation parameters. Numerical results show that the parallel multisplitting relaxation methods always perform much better than their corresponding sequential alternatives, and that the asynchronous multisplitting relaxation methods often outperform their synchronous counterparts. Moreover, the two-sweep relaxed multisplitting methods have better convergence properties than the corresponding one-sweep relaxed ones in the sense that they have larger convergence domains and faster convergence speeds. Hence, the asynchronous multisplitting unsymmetric relaxation iterations should be the methods of choice for solving the large sparse linear complementarity problems in the parallel computing environments.

2. The Relaxed Asynchronous Multisplitting Methods

We assume that the considered multiprocessor system consists of α processors, and the host processor may be chosen to be any one of them. For a matrix $M \in \mathbb{R}^{n\times n}$, let $M = B_i + C_i$ $(i = 1, 2, \ldots, \alpha)$ be α Q-splittings (see [2, 3, 9, 11]) and $E_i \in \mathbb{R}^{n\times n}$ $(i = 1, 2, \ldots, \alpha)$ be α nonnegative diagonal matrices satisfying $\sum_{i=1}^{\alpha} E_i = I$ (the $n \times n$ identity matrix). Then the collection of triples $(B_i, C_i, E_i)(i = 1, 2, \ldots, \alpha)$ is called a multisplitting of the matrix M, and the matrices $E_i(i = 1, 2, \ldots, \alpha)$ are called weighting matrices. We introduce the following necessary notations for describing an asynchronous multisplitting iteration: $N_0 = \{0, 1, 2, \ldots\}$; for $\forall p \in N_0$, $J(p)$ is a nonempty subset of the number set $\Lambda = \{1, 2, \ldots, \alpha\}$; and for $\forall i \in \Lambda$ and $\forall p \in N_0$, $s_i(p)$ is an infinite sequence of nonnegative integers, such that: (1) for $\forall i \in \Lambda$, the set $\{p \in N_0 | i \in J(p)\}$ is infinite; (2) for $\forall i \in \Lambda$ and $\forall p \in N_0$, it holds that $s_i(p) \le p$; and (3) for $\forall i \in \Lambda$, it holds that $\lim_{p\to\infty} s_i(p) = \infty$. If we denote $s(p) = \min_{1\le i\le\alpha} s_i(p)$, then it holds that $s(p) \le p$ and $\lim_{p\to\infty} s(p) = \infty$. Assumption (1) demands that all processors of the multiprocessor system must proceed their local iterations without dead breakdown, Assumption (2) demands that the currently unavailable information should not be used in the current computations, and Assumption (3) demands that every processor of the multiprocessor system must adopt new information to update its local variables continually.

Let $(B_{p,i}, C_{p,i}, E_i)(i = 1, 2, \ldots, \alpha)$, $p \in N_0$, be a sequence of multisplittings of the matrix M. Then the following asynchronous multisplitting relaxation method for solving the LCP(M,q) was presented in Bai and Huang [6]:

Method 2.1. (Asynchronous Multisplitting Relaxation Method).
Given an initial vector $z^0 \in \mathbb{R}^n$. Suppose that we have obtained the approximate solutions $z^1, z^2, \ldots, z^p$ of the LCP(M, q). Then the next approximate solution z^{p+1} of the LCP(M,q) is

computed according to the formulas:

$$z^{p+1} = \sum_{i=1}^{\alpha} E_i z^{p,i} \quad \text{and} \quad z^{p,i} = \begin{cases} \omega \widehat{z}^{p,i} + (1-\omega) z^{s_i(p)}, & \text{if } i \in J(p), \\ z^p, & \text{otherwise}, \end{cases} \qquad i = 1,2,\dots,\alpha,$$

where $\widehat{z}^{p,i}$ is an arbitrary solution of the LCP$(B_{p,i}, q_{p,i})$:

$$z \geq 0, \quad B_{p,i} z + q_{p,i} \geq 0 \quad \text{and} \quad z^T(B_{p,i} z + q_{p,i}) = 0$$

with $q_{p,i} = C_{p,i} z^{s_i(p)} + q$, and $\omega (\neq 0)$ is a relaxation factor.

In Method 2.1, each processor is allowed to update or to retrieve the global approximate solution residing in the host processor at any time. Hence, new information can be used on time once it is available. Moreover, considerable savings in computational workloads are possible, since a component of $z^{p,i}$ does not need to be computed if the corresponding diagonal entry of the weighting matrix E_i is zero. The role of the weighting matrices $E_i (i = 1,2,\dots,\alpha)$ may be regarded as determining the distribution of the computational task to the individual processors.

However, at every iterate step p of Method 2.1, each processor needs to solve an implicit linear complementarity problem LCP$(B_{p,i}, q_{p,i})$. This makes Method 2.1 be less convenient in practical computations. To overcome this disadvantage, Bai and Huang [6] further defined a block and multi-parameter generalization of Method 2.1. It was called the asynchronous multisplitting unsymmetric AOR method. To formulate this method, we separate the number set $N = \{1,2,\dots,n\}$ into α nonempty subsets $J_i (i = 1,2,\dots,\alpha)$ such that $\cup_{i=1}^{\alpha} J_i = N$, and for $i \in \Lambda$, we introduce matrices

$$\begin{aligned}
L_{p,i} &= (\mathcal{L}_{kj}^{(p,i)}) \in \mathbb{R}^{n\times n}, & \mathcal{L}_{kj}^{(p,i)} &= \begin{cases} l_{kj}^{(p,i)}, & \text{for } k,j \in J_i \text{ and } k > j, \\ 0, & \text{otherwise}, \end{cases} \\
U_{p,i} &= (\mathcal{U}_{kj}^{(p,i)}) \in \mathbb{R}^{n\times n}, & \mathcal{U}_{kj}^{(p,i)} &= \begin{cases} u_{kj}^{(p,i)}, & \text{for } k,j \in J_i \text{ and } k < j, \\ 0, & \text{otherwise}, \end{cases} \\
W_{p,i} &= (\mathcal{W}_{kj}^{(p,i)}) \in \mathbb{R}^{n\times n}, & \mathcal{W}_{kj}^{(p,i)} &= \begin{cases} 0, & \text{for } k = j, \\ w_{kj}^{(p,i)}, & \text{otherwise}, \end{cases} \\
E_i &= diag(e_k^{(i)}) \in \mathbb{R}^{n\times n}, & e_k^{(i)} &= \begin{cases} e_k^{(i)} \geq 0, & \text{for } k \in J_i, \\ 0, & \text{otherwise}, \end{cases}
\end{aligned}$$

such that $M = D + L_{p,i} + U_{p,i} + W_{p,i}$, $i = 1,2,\dots,\alpha$, $p = 0,1,2,\dots$, where $M \in \mathbb{R}^{n\times n}$ is the system matrix of the LCP(M,q), and $D = diag(M)$ is a nonsingular matrix. Evidently, for $\forall p \in N_0$, $L_{p,i} (i = 1,2,\dots,\alpha)$ are strictly lower triangular matrices, $U_{p,i} (i = 1,2,\dots,\alpha)$ are strictly upper triangular matrices, $W_{p,i} (i = 1,2,\dots,\alpha)$ are zero-diagonal matrices, and $E_i (i = 1,2,\dots,\alpha)$ are nonnegative diagonal matrices. We call the matrix collections $(D + L_{p,i}, D + U_{p,i}, W_{p,i}, E_i)(i = 1,2,\dots,\alpha)$, $p \in N_0$, a sequence of block triangular multisplittings of the matrix M.

The asynchronous multisplitting unsymmetric AOR method can now be described as follows:

Method 2.2. (Asynchronous Multisplitting Unsymmetric AOR Method).
Given an initial vector $z^0 = ([z^0]_1, \dots, [z^0]_n)^T \in \mathbb{R}^n$. Suppose that we have obtained the approximate solutions $z^1, z^2, \dots, z^p$ of the LCP(M,q). Then the next approximate solution $z^{p+1} = ([z^{p+1}]_1, \dots, [z^{p+1}]_n)^T$ of the LCP(M,q) is computed according to the formulas:

$$[z^{p+1}]_k = \sum_{i=1}^{\alpha} e_k^{(i)} [z^{p,i}]_k \quad \text{and} \quad [z^{p,i}]_k = \begin{cases} [\widehat{z}^{p,i}]_k, & \text{if } i \in J(p), \\ [z^p]_k, & \text{otherwise}, \end{cases} \qquad k = 1,2,\dots,n,$$

where for $k \in J_i$,

$$[\widehat{z}^{p,i}]_k = \begin{cases} 0, & \text{if} \quad \gamma_2 \sum_{j=1}^{k-1} u_{kj}^{(p,i)}([\widehat{z}^{p,i}]_j - [\widetilde{z}^{p,i}]_j) \\ & \qquad +\omega_2[M\widetilde{z}^{p,i} + q]_k > m_{kk}[\widetilde{z}^{p,i}]_k, \\ [\widetilde{z}^{p,i}]_k + \frac{\gamma_2}{m_{kk}} \sum_{j=1}^{k-1} u_{kj}^{(p,i)}([\widehat{z}^{p,i}]_j - [\widetilde{z}^{p,i}]_j) - \frac{\omega_2}{m_{kk}}[M\widetilde{z}^{p,i} + q]_k, & \\ \qquad \text{otherwise}, & \end{cases}$$

and

$$[\widetilde{z}^{p,i}]_k = \begin{cases} 0, & \text{if} \quad \gamma_1 \sum_{j=1}^{k-1} l_{kj}^{(p,i)}([\widetilde{z}^{p,i}]_j - [z^{s_i(p)}]_j) \\ & \qquad +\omega_1[Mz^{s_i(p)} + q]_k > m_{kk}[z^{s_i(p)}]_k, \\ [z^{s_i(p)}]_k + \frac{\gamma_1}{m_{kk}} \sum_{j=1}^{k-1} l_{kj}^{(p,i)}([z^{s_i(p)}]_j - [\widetilde{z}^{p,i}]_j) - \frac{\omega_1}{m_{kk}}[Mz^{s_i(p)} + q]_k, & \\ \qquad \text{otherwise}. & \end{cases}$$

Here, $\gamma_j(j = 1, 2)$ are relaxation factors, and $\omega_j(\neq 0)(j = 1, 2)$ are acceleration factors.

Evidently, when $\alpha = 1$ and $s_i(p) = p(i \in \Lambda, p \in N_0)$, Method 2.1 and Method 2.2 respectively reduce to sequential relaxation methods, and when $J(p) = \Lambda$ and $s_i(p) = p(i \in \Lambda, p \in N_0)$, they respectively reduce to synchronous multisplitting relaxation methods. We remark that Method 2.1 and Method 2.2 are quite suitable for both the tightly coupled multiprocessor and the multicomputer having a shared global memory.

Method 2.2, its synchronous and sequential counterparts, together with some of their typical cases resulted from special choices of the relaxation parameters, will be the experiment methods of this paper.

3. The Experimental Objects

In this section, we present detailed descriptions about the tested problems, the computing environment, the perfomed methods, and the starting and the stopping criterions, which were used in our actual numerical computations.

3.1 The Experimental Problems

Denote by BTD(S_k, T_k, R_k) a block tridiagonal matrix with S_k, T_k and R_k being submatrices of suitable sizes located at its $(k, k-1)$, (k, k) and $(k, k+1)$ positions, respectively, and denote by tri(s_k, t_k, r_k) a tridiagonal matrix with s_k, t_k and r_k being entries located at its $(k, k-1)$, (k, k) and $(k, k+1)$ positions, respectively. We consider the following three practical examples of the LCP(M, q).

Example 3.1. *The linear complementarity problem with the system matrix $M \in \mathbb{R}^{n\times n}$ corresponding to the Laplacian 5-point finite difference operator:*

$$M = BTD(-I, T, -I) \in \mathbb{R}^{n\times n}, \quad T = tri(-1, 4, -1) \in \mathbb{R}^{\tilde{n}\times\tilde{n}},$$

and the known vector $q \in \mathbb{R}^n$ suitably chosen, e.g., $q = (1, -1, \ldots, (-1)^{n-1}, (-1)^n)^T \in \mathbb{R}^n$, where $n = \tilde{n}^2$. Note that $M \in \mathbb{R}^{n\times n}$ is an H_+-matrix. Therefore, the LCP(M, q) has a unique solution [5].

Example 3.2. *The linear complementarity problem with the system matrix $M \in \mathbb{R}^{n\times n}$ corresponding to the Journal Bearing finite difference operator:*

$$M = BTD(-\beta_{k-1}I, T_k, -\beta_k I) \in \mathbb{R}^{n\times n}, \quad T_k = tri(-\beta_{j-1}, \eta_k + \eta_j, -\beta_j) \in \mathbb{R}^{\tilde{n}\times\tilde{n}},$$

and the known vector $q \in \mathbb{R}^n$ suitably chosen, e.g., $q = (50\pi, 50\pi, \ldots, 50\pi)^T \in \mathbb{R}^n$, where π denotes the ratio of the circumference of a circle to its diameter,

$$\beta_k = (k+\frac{1}{2})^3, \quad \eta_k = (k+\frac{1}{2})^3 + (k-\frac{1}{2})^3, \qquad k = 1, 2, \ldots, \tilde{n}.$$

Note that $M \in \mathbb{R}^{n\times n}$ is a K-matrix. Therefore, the LCP(M,q) has a unique solution (cf. [9]).

Example 3.3. *The linear complementarity problem with the system matrix $M \in \mathbb{R}^{n\times n}$ corresponding to the Laplacian 9-point finite difference operator:*

$$M = BTD(S,T,S) \in \mathbb{R}^{n\times n}, \quad T = tri(-4, 20, -4) \in \mathbb{R}^{\tilde{n}\times\tilde{n}}, \quad S = tri(-1, 4, -1) \in \mathbb{R}^{\tilde{n}\times\tilde{n}},$$

and the known vector $q \in \mathbb{R}^n$ suitably chosen, e.g., $q = (1, -1, \ldots, (-1)^{n-1}, (-1)^n)^T \in \mathbb{R}^n$. Note that $M \in \mathbb{R}^{n\times n}$ is an H_+-matrix. Therefore, the LCP(M,q) has a unique solution.

We remark that the finite difference discretizations at equidistant grids of a free boundary value problem about the flow of water through a porous dam (see [10]) may result in linear complementarity problems of the types of Example 3.1 and Example 3.3. For the practical background, the mathematical description and the numerical treatment of the journal bearing problem, we refer to [9] for details.

3.2 The Experimental Environment

We run our programs on an SGI Power Challenge multiprocessor computer as PVM applications. This parallel machine consists of four 75 MHz TFP 64-bit RISC processors. These CMOS processors each delivers a peak theoretical performance of 0.3 GFLOPS. The data cache size is 16 Kbytes.

3.3 The Experimental Methods

The tested methods in our numerical experiments are the asynchronous multisplitting unsymmetric AOR method and its synchronous and sequential counterparts, together with some of their typical cases from special choices of the relaxation parameters. They are listed in the following three tables:

(a) the sequential relaxation methods in [8] and [12]:

Method	γ_1	ω_1	γ_2	ω_2	Description
SOR	ω	ω	0	0	the successive overrelaxation method
SSOR	ω	ω	ω	ω	the symmetric SOR method
USOR	γ	γ	ω	ω	the unsymmetric SOR method
AOR	γ	ω	0	0	the accelerated overrelaxation method
SAOR	γ	ω	γ	ω	the symmetric AOR method

(b) the synchronous multisplitting relaxation methods in [3] and [11]:

Method	γ_1	ω_1	γ_2	ω_2	Description
MSOR	ω	ω	0	0	the multisplitting SOR method
MSSOR	ω	ω	ω	ω	the multisplitting SSOR method
MUSOR	γ	γ	ω	ω	the multisplitting USOR method
MAOR	γ	ω	0	0	the multisplitting AOR method
MSAOR	γ	ω	γ	ω	the multisplitting SAOR method

(c) the asynchronous multisplitting relaxation methods:

Method	γ_1	ω_1	γ_2	ω_2	Description
AMSOR	ω	ω	0	0	the asynchronous MSOR method
AMSSOR	ω	ω	ω	ω	the asynchronous MSSOR method
AMUSOR	γ	γ	ω	ω	the asynchronous MUSOR method
AMAOR	γ	ω	0	0	the asynchronous MAOR method
AMSAOR	γ	ω	γ	ω	the asynchronous MSAOR method

For the convenience of application and without loss of generality, the block triangular multisplittings $(D+L_{p,i}, D+U_{p,i}, W_{p,i}, E_i)(i=1,2,\ldots,\alpha)$, $p=0,1,2,\ldots$, are now chosen to be stationary ones (i.e., they are independent of the iterate index p), and they have the following structures:

$$\begin{aligned}
L_{p,i} &= (\mathcal{L}_{kj}^{(p,i)}) \in \mathbb{R}^{n\times n}, \quad \mathcal{L}_{kj}^{(p,i)} &&= \begin{cases} m_{kj}, & \text{for } k,j\in J_i \text{ and } k>j, \\ 0, & \text{otherwise}, \end{cases} \\
U_{p,i} &= (\mathcal{U}_{kj}^{(p,i)}) \in \mathbb{R}^{n\times n}, \quad \mathcal{U}_{kj}^{(p,i)} &&= \begin{cases} m_{kj}, & \text{for } k,j\in J_i \text{ and } k<j, \\ 0, & \text{otherwise}, \end{cases} \\
W_{p,i} &= (\mathcal{W}_{kj}^{(p,i)}) \in \mathbb{R}^{n\times n}, \quad \mathcal{W}_{kj}^{(p,i)} &&= \begin{cases} 0, & \text{for } k=j, \\ 0, & \text{for } k,j\in J_i, \\ m_{kj}, & \text{otherwise}, \end{cases} \\
E_i &= diag(e_k^{(i)}) \in \mathbb{R}^{n\times n}, \quad e_k^{(i)} &&= \begin{cases} 1, & \text{for } 1\le k\le \tilde{n}_1\tilde{n}, \quad i=1, \\ 0.5, & \text{for } \tilde{n}_{i-1}\tilde{n}+1\le k\le \tilde{n}_i\tilde{n}, \quad 2\le i\le\alpha, \\ 0.5, & \text{for } \tilde{n}_i\tilde{n}+1\le k\le\tilde{n}_{i+1}\tilde{n}, \quad 1\le i\le\alpha-1, \\ 1, & \text{for } \tilde{n}_\alpha\tilde{n}+1\le k\le n, \quad i=\alpha, \end{cases}
\end{aligned}$$

where for $i=1,2,\ldots,\alpha$, $\tilde{n}_i = \text{Int}\left(\frac{i\tilde{n}}{\alpha+1}\right)$, and $J_i=\{\tilde{n}_{i-1}\tilde{n}+1, \tilde{n}_{i-1}\tilde{n}+2, \ldots, \tilde{n}_{i+1}\tilde{n}\}$. Here, $\text{Int}(\bullet)$ denotes the integer part of the corresponding real number.

3.4 The Starting and the Stopping Criterions

All the computations are started from an initial vector having all components equal to 40.0, and terminated once the current iterations z^p obey $\frac{|(z^p)^T(Mz^p+q)|}{|(z^0)^T(Mz^0+q)|} \le 10^{-7}$.

4. Numerical Results

The three examples of the LCP(M,q) of various sizes are tested by the sequential, the synchronous and the asynchronous relaxation methods in Tables (a)-(c) in Section 3.3 when the processor number α ranges from $\{1,2,3,4\}$, respectively. For $n=4900$ and $\alpha=3$, some of the representative numerical results are listed in the following tables and depicted by the following figures. We use CPU to denote the CPU time (in seconds) required for an iteration to reach the above stopping criterion, ∞ to denote that an iteration does not satisfy the stopping criterion after 5000 iterations, and SP to denote the speed-up of a parallel execution, which is defined to be the ratio of the CPU times of the sequential to the corresponding parallel runnings.

4.1 Performance of Example 3.1

The computing results of this example were reported in Bai and Huang [6]. However, for the sake of conveniently comparing them with those of the other two examples so that the numerical behaviours of the parallel multisplitting relaxation methods can be revealed in depth, we still copy them here.

Tabel (A_1)**:** CPUs for the sequential SOR-like methods

ω	0.6	0.7	0.9	1.1	1.3	1.5
SOR	91.92	73.14	48.09	32.51	21.38	13.23
SSOR	45.98	37.03	24.27	16.26	10.72	∞

Tabel (A_2)**:** CPUs and SPs for the multisplitting SOR-like methods

ω		0.6	0.7	0.9	1.1	1.3	1.5
MSOR	CPU	48.12	37.70	24.92	16.82	11.10	6.97
	SP	1.91	1.94	1.93	1.93	1.93	1.90
MSSOR	CPU	23.23	19.06	12.55	8.47	5.62	∞
	SP	1.98	1.94	1.93	1.92	1.91	-

Table (A_1) presents the CPUs for the sequential SOR and SSOR methods at some values of the relaxation parameter ω. Table (A_2) presents the CPUs and the SPs for the synchronous multisplitting SOR and SSOR methods at the same values of the relaxation parameter ω as in Table (A_1). We note that for the MSOR method, the optimal CPU 6.97 is attained at $\omega = 1.5$, and the optimal SP 1.94 is attained at $\omega = 0.7$, however, for the MSSOR method, the optimal CPU 5.62 is attained at $\omega = 1.3$, and the optimal SP 1.98 is attained at $\omega = 0.6$. Therefore, the synchronous multisplitting SOR and SSOR methods cost smaller CPUs at a larger parameter, and achieve higher SPs at a smaller parameter. The reason is that the MSOR and MSSOR methods converge much faster than their corresponding sequential ones when the parameter is far away from the optimum, and they have comparable convergence speeds with the corresponding sequential relaxation methods when the parameter is much close to the optimum. Roughly speaking, the CPUs of the MSSOR method are approximately halves of those of the MSOR method, and the SPs of the MSSOR method are majorly higher than those of the MSOR method, correspondingly. The average of the SPs for the MSOR method is 1.923, while that of the SPs for the MSSOR method is 1.936.

Tabel (A_3)**:** CPUs and SPs for the asynchronous multisplitting SOR-like methods

ω		0.6	0.7	0.9	1.1	1.3	1.5
AMSOR	CPU	42.41	33.59	22.09	14.99	10.04	6.34
	SP	2.17	2.18	2.18	2.17	2.13	2.09
AMSSOR	CPU	21.30	16.80	11.26	7.70	5.12	∞
	SP	2.16	2.20	2.16	2.11	2.09	-

Table (A_3) presents the CPUs and the SPs for the asynchronous multisplitting SOR and SSOR methods at the same values of the relaxation parameter ω as in Table (A_1) and Table (A_2). We note that for the AMSOR method, the optimal CPU 6.34 is attained at $\omega = 1.5$, and the optimal SP 2.18 is attained at both $\omega = 0.7$ and 0.9, however, for the AMSSOR method, the optimal CPU 5.12 is attained at $\omega = 1.3$, and the optimal SP 2.2 is attained at $\omega = 0.7$. Therefore, the asynchronous multisplitting SOR and SSOR methods cost smaller CPUs at a larger parameter, and achieve higher SPs at a smaller parameter. The reason is that the AMSOR and AMSSOR methods converge much faster than their corresponding sequential ones when the parameter is far away from the optimum, and they have comparable convergence speeds with the corresponding sequential relaxation methods when the parameter is much close to the optimum. Roughly speaking, the CPUs of the AMSSOR method are approximately halves of those of the AMSOR method, and the SPs of the AMSSOR method are majorly lower than those of the AMSOR method, correspondingly. The average of the SPs for the AMSOR method is 2.15, while that of the SPs for the AMSSOR method is 2.14.

Table (A_2) and Table (A_3) show that the asynchronous multisplitting SOR and SSOR methods outperform the synchronous multisplitting SOR and SSOR methods, and the multisplitting SSOR-like methods outperform the corresponding multisplitting SOR-like methods, respectively, in terms of both elapsed time and parallel speed-up.

Tabel (A_4)**:** CPUs for the sequential AOR-like methods

γ	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.7	1.8	1.9
ω	1.0	1.0	1.0	1.1	1.1	1.1	1.2	1.2	1.2	1.2
AOR	63.03	54.90	47.08	35.65	28.53	21.42	13.06	9.79	6.50	3.03
SAOR	29.69	26.03	22.97	17.79	14.87	11.42	7.21	5.59	3.91	2.04
USOR	39.31	31.74	25.37	18.41	13.94	9.93	∞	∞	∞	∞

Tabel (A_5)**:** CPUs and SPs for the multisplitting AOR-like methods

γ		0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.7	1.8	1.9
ω		1.0	1.0	1.0	1.1	1.1	1.1	1.2	1.2	1.2	1.2
MAOR	CPU	33.17	28.74	24.52	18.59	15.00	11.35	7.04	5.43	3.68	2.00
	SP	1.90	1.91	1.92	1.92	1.90	1.89	1.86	1.80	1.77	1.52
MSAOR	CPU	15.14	13.59	11.97	9.48	7.31	6.16	4.01	3.12	2.21	1.29
	SP	1.96	1.92	1.92	1.88	2.03	1.85	1.80	1.79	1.77	1.58
MUSOR	CPU	20.45	16.53	13.19	9.76	7.46	5.25	∞	∞	∞	∞
	SP	1.92	1.92	1.92	1.89	1.87	1.89	-	-	-	-

Tabel (A_6)**:** CPUs and SPs for the asynchronous multisplitting AOR-like methods

γ		0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.7	1.8	1.9
ω		1.0	1.0	1.0	1.1	1.1	1.1	1.2	1.2	1.2	1.2
AMAOR	CPU	28.80	25.16	21.91	16.53	13.35	10.25	6.30	4.84	3.30	1.82
	SP	2.19	2.18	2.15	2.16	2.14	2.09	2.07	2.02	1.97	1.66
AMSAOR	CPU	13.64	12.14	10.72	8.40	6.86	5.47	3.52	2.72	1.96	1.19
	SP	2.18	2.14	2.14	2.12	2.17	2.09	2.08	2.06	1.99	1.71
AMUSOR	CPU	18.15	14.67	11.78	8.73	6.63	4.76	∞	∞	∞	∞
	SP	2.17	2.16	2.15	2.11	2.10	2.09	-	-	-	-

Table (A_4) presents the CPUs for the sequential AOR, SAOR and USOR methods at some values of the relaxation parameter pair (γ, ω), which includes the optimal points in Tabels (A_1)–(A_3). Table (A_5) presents the CPUs and the SPs for the synchronous multisplitting AOR, SAOR and USOR methods at the same values of the relaxation parameter pair (γ, ω) as in Table (A_4). We note that for the MAOR method the optimal CPU 2.0 is attained at $(\gamma, \omega) = (1.9, 1.2)$, and the optimal SP 1.92 is attained at both $(\gamma, \omega) = (0.8, 1.0)$ and $(1.0, 1.1)$; for the MSAOR method the optimal CPU 1.29 is attained again at $(\gamma, \omega) = (1.9, 1.2)$, and the optimal SP 2.03

is attained at $(\gamma, \omega) = (1.2, 1.1)$; and for the MUSOR method the optimal CPU 5.25 is attained at $(\gamma, \omega) = (1.4, 1.1)$, and the optimal SP 1.92 is attained at $(\gamma, \omega) = (0.4, 1.0)$, $(0.6, 1.0)$ and $(0.8, 1.0)$. Therefore, the synchronous multisplitting AOR, SAOR and USOR methods cost smaller CPUs at a larger parameter pair, and achieve higher SPs at a smaller parameter pair. The reason is that the MAOR, MSAOR and MUSOR methods converge much faster than their corresponding sequential ones when the parameter pair is far away from the optimum, and they have comparable convergence speeds with the corresponding sequential relaxation methods when the parameter pair is much close to the optimum. Roughly speaking, the CPUs of the MSAOR and MUSOR methods are approximately halves of those of the MAOR method, and the CPUs of the MUSOR method are a little bit more than those of the MSAOR method, correspondingly. The average of the SPs for the MAOR method is 1.85, the average of the SPs for the MSAOR method is also 1.85, while that of the SPs for the MUSOR method is 1.9. Table (A_6) presents the CPUs and the SPs for the asynchronous multisplitting AOR, SAOR and USOR methods at the same values of the relaxation parameter pair (γ, ω) as in Table (A_4) and Table (A_5). We note that for the AMAOR method the optimal CPU 1.82 is attained at $(\gamma, \omega) = (1.9, 1.2)$, and the optimal SP 2.19 is attained at $(\gamma, \omega) = (0.4, 1.0)$; for the AMSAOR method the optimal CPU 1.19 is attained at $(\gamma, \omega) = (1.9, 1.2)$, and the optimal SP 2.18 is attained at $(\gamma, \omega) = (0.4, 1.0)$; and for the AMUSOR method the optimal CPU 4.26 is attained at $(\gamma, \omega) = (1.4, 1.1)$, and the optimal SP 2.17 is attained at $(\gamma, \omega) = (0.4, 1.0)$. Therefore, the asynchronous multisplitting AOR, SAOR and USOR methods cost smaller CPUs at a larger parameter pair, and achieve higher SPs at a smaller parameter pair. The reason is that the AMAOR, AMSAOR and AMUSOR methods converge much faster than their corresponding sequential ones when the parameter pair is far away from the optimum, and they have comparable convergence speeds with the corresponding sequential relaxation methods when the parameter pair is much close to the optimum. Roughly speaking, the CPUs of the AMSAOR and MUSOR methods are approximately halves of those of the AMAOR method, and the CPUs of the AMUSOR method are a little bit more than those of the AMSAOR method, correspondingly. The average of the SPs for the AMAOR method is 2.06, the average of the SPs for the AMSAOR method is 2.07, while that of the SPs for the AMUSOR method is 2.13.

Tables (A_4)-(A_6) show that the asynchronous multisplitting AOR, SAOR and USOR methods outperform the synchronous multisplitting AOR, SAOR and USOR methods, respectively, and the two-sweep relaxed multisplitting methods outperform the one-sweep relaxed multisplitting methods, correspondingly, in terms of both elapsed time and parallel speed-up. Moreover, the two-parameter relaxed multisplitting AOR-like methods have larger convergence domains than the corresponding two-parameter relaxed multisplitting SOR-like methods.

In Figures (A_1)-(A_3), we give the behaviours of the asynchronous multisplitting AOR, SAOR and USOR methods, respectively. The r and w axes in each figure correspond to the γ and ω axes, respectively. It is clearly demonstrated that all these methods have good convergence properties over a wide range of the relaxation parameters.

4.2 Performance of Example 3.2

The computing results of this example are depicted in Figures (B_1)-(B_7). In these figures, the x-axis corresponds to the relaxation parameter, and the y-axis corresponds to the CPU in logarithmic scale. Note that here we only investigate a representative curve from the two-dimensional surface of the CPU vs. the relaxation parameters γ and ω for the two-parameter relaxed methods, which was cut off by the hyperplane $\omega = 1.3$.

In Figures $(B_1) - (B_3)$, the classes of USOR, AOR and SAOR methods were compared in both sequential and parallel settings. Figure (B_1) depicts the performance of the USOR,

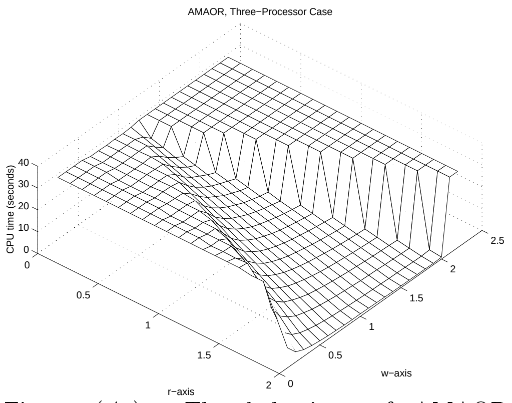

Figure (A_1): The behaviour of AMAOR method for the problem with $n = 4900$. The divergent points are represented in the graph by CPU time being 30.

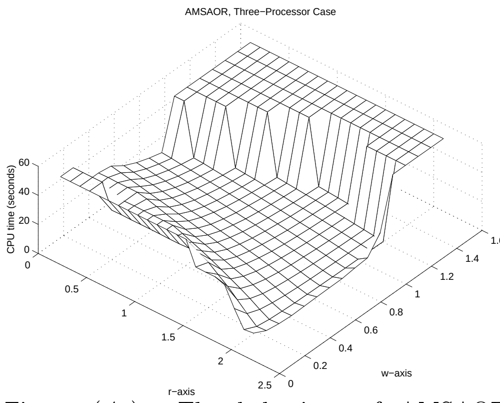

Figure (A_2): The behaviour of AMSAOR method for the problem with $n = 4900$. The divergent points are represented in the graph by CPU time being 40.

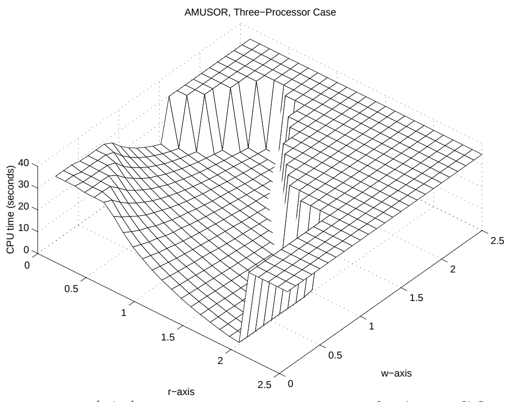

Figure (A_3): The behaviour of AMUSOR method for the problem with $n = 4900$. The divergent points are represented in the graph by CPU time being 30.

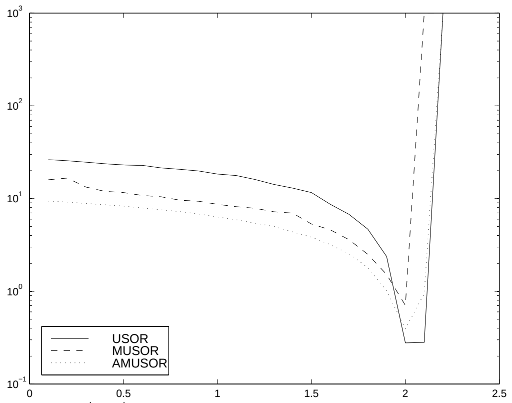

Figure (B_1): CPU vs. parameter curves for the USOR-like methods.

MUSOR and AMUSOR methods. Evidently, the AMUSOR method always outperforms the MUSOR method, and the MUSOR method almost always outperforms the USOR method except for the points nearby the optimum. Figure (B_2) depicts the performance of the AOR, MAOR and AMAOR methods. Obviously, the AMAOR method always outperforms the MAOR method, and the MAOR method almost always outperforms the AOR method except for the points nearby the optimum. Note that the AMAOR method has larger convergence domain than the MAOR method. Figure (B_3) depicts the performance of the SAOR, MSAOR and AMSAOR methods. Clearly, the AMSAOR method always outperforms the MSAOR method, and the MSAOR method almost always outperforms the SAOR method except for the points nearby the optimum. That both synchronous and asynchronous multisplitting relaxation methods perform worse than the corresponding sequential relaxation method in the nearby of the optimal points of the relaxation parameter is probably because the convergence properties of the parallel methods are not better than the sequential method when the relaxation parameters are much close to the optimal values.

In Figures $(B_4) - (B_7)$, the classes of SOR, SSOR and AOR methods were compared in both

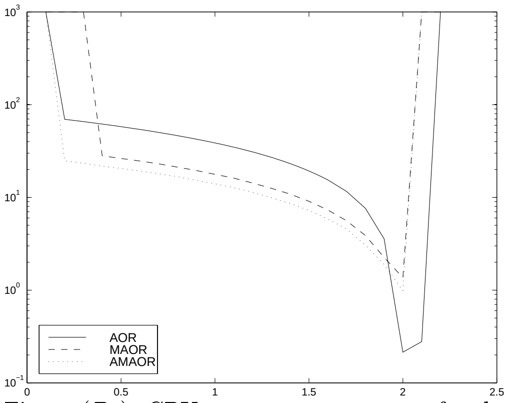

Figure (B_2): CPU vs. parameter curves for the AOR-like methods.

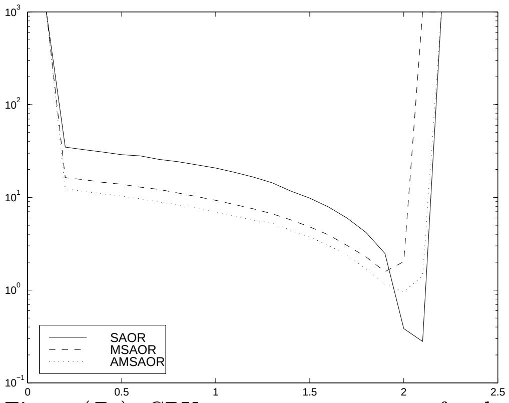

Figure (B_3): CPU vs. parameter curves for the SAOR-like methods.

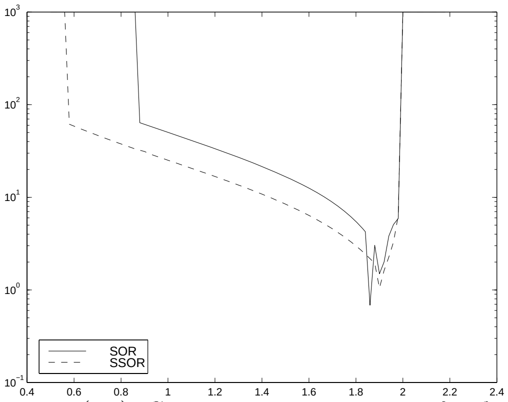

Figure (B_4): CPU vs. parameter curves for the SOR and SSOR methods.

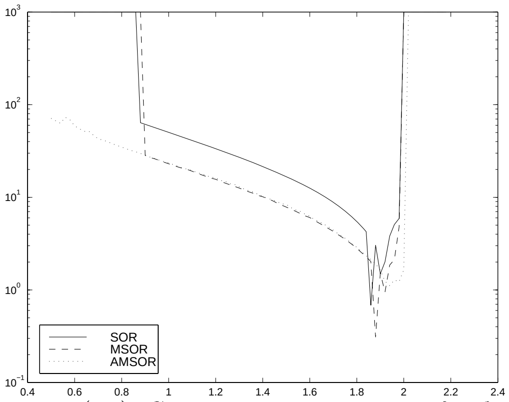

Figure (B_5): CPU vs. parameter curves for the SOR-like methods.

sequential and parallel settings. Figure (B_4) depicts the performance of the SOR and SSOR methods. Figure (B_5) depicts the performance of the SOR, MSOR and AMSOR methods. Evidently, the MSOR method always outperforms the SOR method, and its convergence prop-

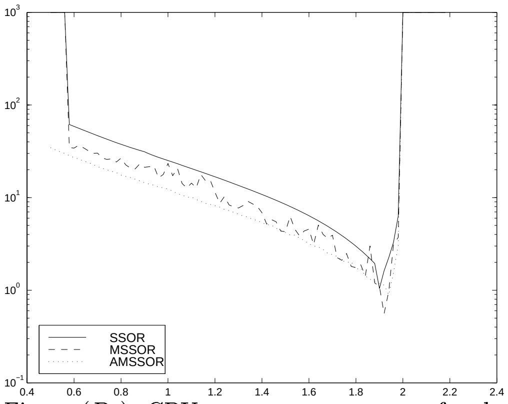

Figure (B_6): CPU vs. parameter curves for the SSOR-like methods.

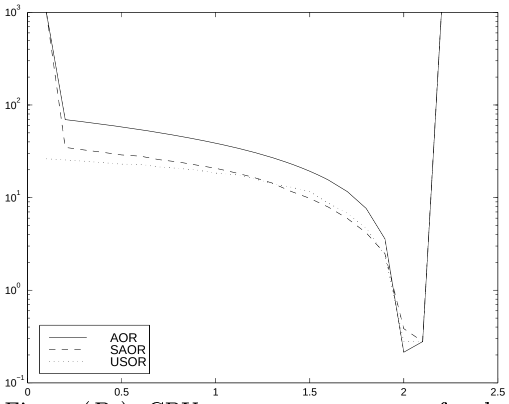

Figure (B_7): CPU vs. parameter curves for the AOR, SAOR and USOR methods.

erty is quite comparable with the AMSOR method, except for the points nearby the optimum. Around the optimal point, the MSOR method has the best numerical behaviour, while the AMSOR method has the worst one. However, the AMSOR method has larger convergence domain than both MSOR and SOR methods. Figure (B_6) depicts the performance of the SSOR, MSSOR and AMSSOR methods. Obviously, the MSSOR method always outperforms the SSOR method, and its convergence property is quite comparable with the AMSSOR method, except for the points nearby the optimum. Around the optimal point, the MSSOR method has the best numerical behaviour, while the SSOR method has the worst one. However, the AMSSOR method has larger convergence domain than both MSSOR and SSOR methods. Figure (B_7) depicts the performance of the AOR, SAOR and USOR methods.

Figures $(B_4) - (B_7)$ show that the asynchronous multisplitting relaxation methods almost always outperform the synchronous multisplitting relaxation methods and the sequential relaxation methods. That the convergence speeds of these three classes of methods are quite different in the nearby of the optimums of the relaxation parameters is probably because of the drastic difference of the convergence properties of these methods when the relaxation parameters are close to the optimums.

4.3 Performance of Example 3.3

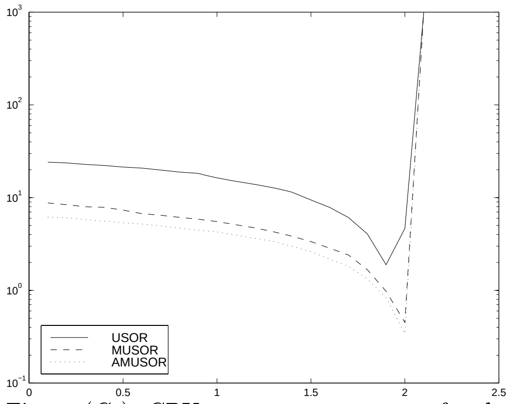

Figure (C_1): CPU vs. parameter curves for the USOR-like methods.

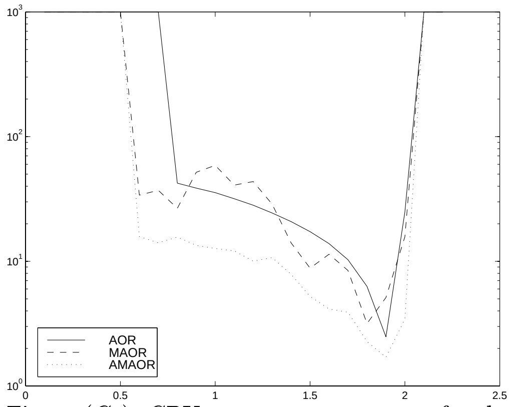

Figure (C_2): CPU vs. parameter curves for the AOR-like methods.

The computing results of this example are depicted in Figures (C_1)-(C_7). In these figures, the x-axis corresponds to the relaxation parameter, and the y-axis corresponds to the CPU in logarithmic scale. Again, note that here we only investigate a representative curve from the two-dimensional surface of the CPU vs. the relaxation parameters γ and ω for the two-parameter relaxed methods, which was cut off by the hyperplane $\omega = 1.3$.

In Figures $(C_1) - (C_3)$, the classes of USOR, AOR and SAOR methods were compared in both sequential and parallel settings, and in Figures $(C_4) - (C_7)$, the classes of SOR, SSOR and AOR methods were done in these situations, too. The numerical behaviours of these methods for this example are quite analogous to those for Example 3.2, correspondingly.

5. Conclusions

The numerical computations show that: in terms of CPU time and parallel efficiency, the asynchronous multisplitting relaxation methods are superior to the corresponding synchronous

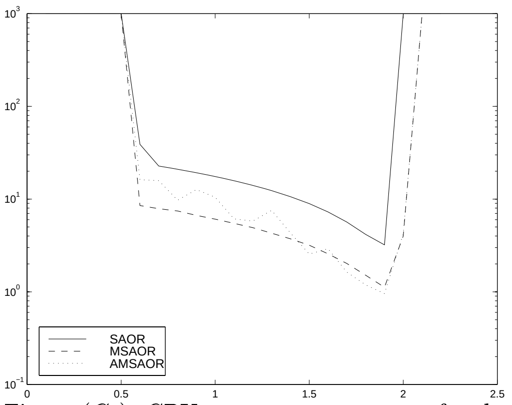

Figure (C_3): CPU vs. parameter curves for the SAOR-like methods.

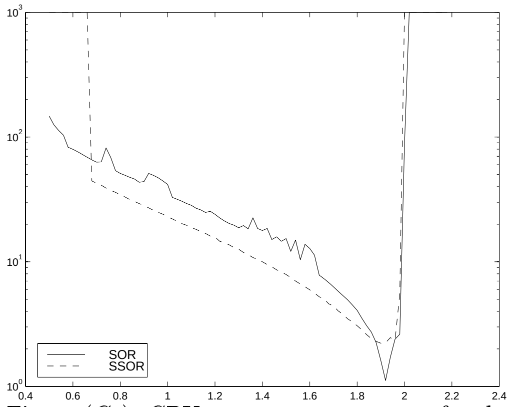

Figure (C_4): CPU vs. parameter curves for the SOR and SSOR methods.

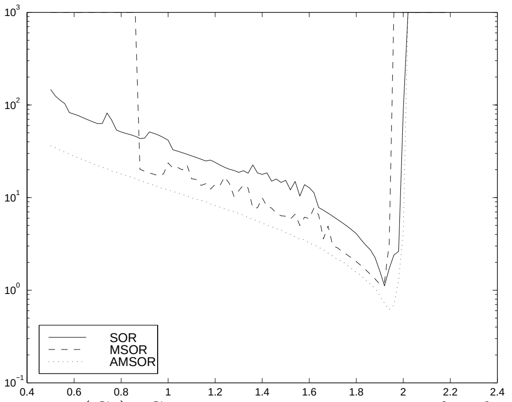

Figure (C_5): CPU vs. parameter curves for the SOR-like methods.

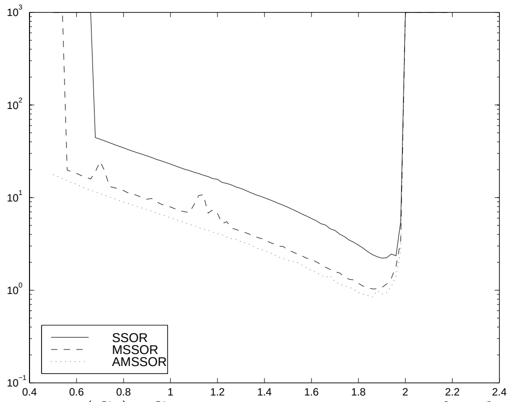

Figure (C_6): CPU vs. parameter curves for the SSOR-like methods.

multisplitting relaxation methods, the multisplitting accelerated overrelaxation methods are superior to the corresponding multisplitting successive overrelaxation methods, and the two-sweep relaxed multisplitting methods are superior to the corresponding one-sweep relaxed multisplit-

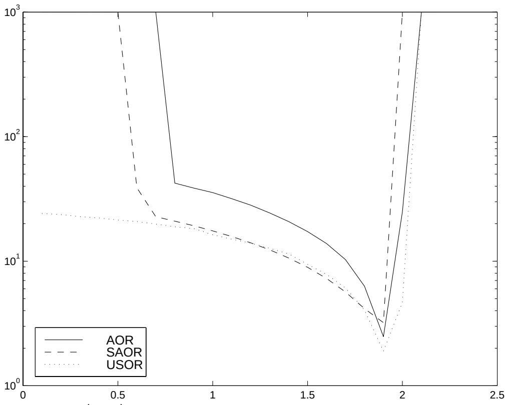

Figure (C_7): CPU vs. parameter curves for the AOR, SAOR and USOR methods.

ting methods. In particular, the advantages of the AMAOR and AMSAOR methods over the AMSOR, AMSSOR and AMUSOR methods, respectively, are, roughly speaking, that (i) when the latter ones diverge, the former ones can still converge; (ii) when the latter ones converge, the former ones converge faster with higher parallel efficiency; and (iii) the former ones are less sensitive to the relaxation parameters and they have larger convergence domains than the latter ones. Moreover, the numerical property of the AMUSOR method is almost comparable with that of the AMSAOR method. Therefore, we can conclude that the two-sweep multi-parameter relaxed asynchronous multisplitting methods have better numerical properties than their corresponding synchronous alternatives, and they should be our choice of methods in actual computations.

References

*setckptjcm*1393.*bbl*

References

[1] Z.Z. Bai, Parallel Iterative Methods for Large-sparse Systems of Algebraic Equations, Ph.D. Thesis, Shanghai University of Science and Technology, March, 1993.

[2] Z.Z. Bai, Parallel chaotic multisplitting iterative methods for the large sparse linear complementarity problem, *J. Comput. Math.,* **19** (2001), 281-292.

[3] Z.Z. Bai, On the convergence of the multisplitting methods for the linear complementarity problem, *SIAM J. Matrix Anal. Appl.,* **21** (1999), 67-78.

[4] Z.Z. Bai, D.J. Evans, Matrix multisplitting methods with applications to linear complementarity problems I: Parallel synchronous and chaotic methods, *Calculateurs Parallés,* **13:1** (2001), 125-154; II: Parallel asynchronous methods, *Intern. J. Computer Math.,* **79:2** (2002), 205-232.

[5] Z.Z. Bai, D.J. Evans, Matrix multisplitting relaxation methods for linear complementarity problems, *Intern. J. Computer Math.,* **63** (1997), 309-326.

[6] Z.Z. Bai, Y.G. Huang, A class of asynchronous parallel multisplitting relaxation methods for the large sparse linear complementarity problems, *J. Comput. Math.,* to appear, 2000.

[7] Z.Z. Bai, J.C. Sun, D.R. Wang, A unified framework for the construction of various matrix multisplitting iterative methods for large sparse system of linear equations, *Computers Math. Appl.,* **32** (1996), 51-76.

[8] R.W. Cottle, G.H. Golub, R.S. Sacher, On the solution of large structured linear complementarity problems: The block partitioned case, *Appl. Math. Optim.,* **4** (1978), 347-363.

[9] R.W. Cottle, J.S. Pang, R.E. Stone, The Linear Complementarity Problem, Academic Press, New York, 1992.

[10] C.M. Elliott, J.R. Ockenden, Weak Variational Methods for Moving Boundary Value Problems, Pitman, London, 1982.

[11] N. Machida, M. Fukushima, T. Ibaraki, A multisplitting method for symmetric linear complementarity problems, *J. Comp. Appl. Math.,* **62** (1995), 217-227.

[12] O.L. Mangasarian, Solution of symmetric linear complementarity problems by iterative methods, *J. Optim. Theory Appl.,* **22** (1977), 465-485.