TWO-PHASE SEMI-ASYNCHRONOUS PARALLEL ITERATIVE METHODS FOR THE SYMMETRIC LINEAR COMPLEMENTARITY PROBLEMS

Jiann-Min Yang and Tai-Sheng Chang

楊　建　民*　　張　泰　生**

摘　要

在本文中，我們提出了一個在求解對稱線性互補問題上的二階段半非同步平行演算法。此法的求解程序可分成一連串交互的兩個演算階段。在第一階段中，我們進行完全非同步的演算，亦即，無論是各處理器在更新疊代解的次數上，或是各處理器之間的通訊次序上，都完全沒有任何的限制；而在第二階段的演算中，我們僅執行一個簡單的線性搜索步驟。此演算法大大地減少了因同步化及通訊上所產的冗額成本，因此增進了演算法的效率；而且由於此法在更新疊代解的演算中，擷取了大部份最新的資訊，也因而加快了收斂速度。至於在建立此法的收斂理論上，我們僅須對線性搜索範圍給予一微弱的限制，即可保證此半非同步演算過程，終將收斂到線性互補問題的解。我們將此法在IBM　3090的機器上模擬執行，並應用於求解一個由二次規劃所導出之對稱線性互補問題，其數值結果亦顯示出此法優於一些同步的平行疊代法。

ABSTRACT

In this paper, we propose a two-phase semi-asynchronous parallel iterative method for solving the symmetric linear complementarity problem. The solution process is divided into 1st-phase and 2nd-phase alternatingly. In the 1st-phase, we allow the process to be completely asynchronous. That is, the computation for new iterate on separate processors and communication in between them could be absolutely free and chaotic. We impose essentially no restrictions on the ordering and the number of times of such iterate updating and information exchanging among processors. In the 2nd-phase, we simply perform a line search. The method dramatically reduces the overhead arising from synchronization and communication. Moreover, the process has always used and taken advantage of the most recent information available, when updating the iterate. Convergence of the overall scheme is established under a mild restriction on the

* 國立政治大學資訊管理學系

**國立政治大學應用數學系

本研究接受國科會專題計畫補助（NSC 78-0408-E004-01），特此誌謝。

range of the line search. The method is simulated and implemented on the IBM 3090 machine for solving the symmetric linear complementarity problem arising from the constrained convex quadratic program. Numerical results are also reported.

1. Introduction.

In this paper, we propose a two-phase semi-asynchronous iterative method for the symmetric linear complementarity problem (LCP) which is to find an $x \in R^n$ such that:

$$w = q + Mx \geq 0, \quad x \geq 0, \quad \text{and} \quad w^t x = 0 \tag{1.1}$$

where q is a given n-vector and M is a given $n \times n$ real symmetric matrix.

The iterative solution for the above LCP(1.1) is a fundamental research area and has been studied for many years. Variant methods have been proposed. Motivated by the arrival for the new generation super and parallel computers which provide extremely powerful computing capability for the solution of very large-scale scientific problems. We focus our study on the parallel iterative methods for solving the LCP(1.1).

The idea of the parallel methods is to share work by using multiple processors and then to coordinate them via communication links (See Kung [1976]). One way to achieve the parallel iterative solutions for the LCP(1.1) is to decompose problem LCP(1.1) into several subproblems each of which can be assigned to a separate processor and then be solved independently. During the solution process, the processors need to be synchronized and then communicate to each other in each iteration. The problem LCP(1.1) will be solved by repeating the above process iteratively until some termination criteria have been reached (see Pang and Yang [1988a], [1988b]).

The above mentioned parallel iterative methods for the LCP(1.1) are synchronized algorithms. For synchronization, we mean that all the processors have to adjust their paces at some time spochs for exchanging information and then restart. Since the time needed for each processor to accomplish its assigned work might be different (see Kung [1976]), all the faster processors have to wait and sit idle there until the slowest processor having completed its assignment. Therefore, the overhead produced by the idling processors will typically downgrade the performance of such algorithms.

One way to reduce such overhead is to develop the asynchronous algorithms (see Pang and Yang [1988b]). In most asynchronous algorithms, there exist a set of global variables which hold the most recent iterates. Whenever a processor has completed its work, it can exchange some information by reading and updating some global variables, and then, enters into the next iteration without waiting for other processors. Since no synchronization is needed, there exists no idling cost arising from waiting for communication.

However, we should remark here that as far as we are concerned, asynchronous methods were applied only for very restrictive problem domain. In fact, the only known asynchronous algorithms available for solving complementarity problems need the problem having some "nice" properties on the structure of input matrix M, e.g., M is a diagonally dominant or a Minkowski matrix (see Pang and Yang [1988b]).

The issue of how to exploit the benefits of asynchronous computing without reducing the domain of applications is very important. Therefore, we develop a semi-asynchronous algorithm for solving the LCP(1.1). We take the advantage of both the synchronized and asynchronous computing. In one hand, we eliminate most of the synchronized overhead by using the asynchronous computing through out almost the whole solution process. On the other hand, we introduce a very little synchronized step of performing a line search in order to preserve the domain of applications. Nevertheless, the line search step itself could improve the overall performance of the algorithm.

The rest of this paper is organized as the follows. In next section, we describe the semi-asynchronous iterative methods in detail. In section 3, we establish the convergence of the iterative scheme. In section 4, we report the numerical results and computational experiences. And in last section, we give some conclusion remarks and discuss some related future studies.

2. A Two-Phase Semi-Asynchronous Algorithm.

A great variety of iterative methods have been proposed to solve the LCP(1.1). Many of them can be described as the matrix splitting scheme addressed in Pang [1982]. We begin the description by letting

$$M = B + C,$$

that is we split the matrix M into two matrices B and C with the same order as M. Then we generate a sequence $\{x^k\}$ by the following way. Given x^k, let x^{k+1} be a solution to the following LCP:

$$w^k = q + Cx^k + Bx \geq 0, \quad x \geq 0, \quad \text{and} \quad x^t w^k = 0 \tag{2.1}$$

The general iterative scheme described above can be easily specialized to achieve the parallel computation. For instance, choosing B to be a block diagonal matrix will reach the decomposition of the LCP(2.1) into several smaller independant subproblems. Indeed, let

$$B = (B_{IJ})$$

be a partition of the matrix B into submatrices B_{IJ} with each diagonal submatrix B_{II} being square. Then each subproblem decomposed from LCP(2.1) can be specified as the following:

$$w_I^k = q_I + \sum_J C_{IJ} x_J^k + B_{II} x_I \geq 0, \quad x_I \geq 0 \quad \text{and} \quad x_I^t w_I^k = 0 \tag{2.2}$$

Therefore, the computation for each x_I can be assigned to separate processors. However, we need to synchronize the above processes in each iteration in order to get the full iterate x^{k+1}. The extra overhead resulting from synchronization will typically downgrade the performance of the algorithms.

Now, we begin to describe our two-phase semi-asynchronous method and show how to reduce such synchronization overhead by using asynchronous computing on parallel processors.

The solution process of the two-phase semi-asynchronous method is divided into 1st- and 2nd-phase alternatingly. In each 1st-phase, the computation and communication for the components of x-variable are allowed to be completely free and choatic. While in the 2nd-phase, one line search step and one simple LCP iteration are performed. We call the process an outer iteration when it has completed one 1st-phase and one 2nd-phase.

In each outer iteration k, the 1st-phase is divided into many time points, $t = 1, 2, \ldots, r, \ldots$, each of which represents the time when one or more processors having just finished their computation and being ready for further updating. The updating process of x-components is achieved by the following formula:

$$x_i = \left[- \frac{\left[\sum_j c_{ij}^k u_j^{k,r}\right] + q_i}{b_{ii}^k} \right]_+ \qquad i \in J^{k,r} \qquad (2.3)$$

where (1) $a = (b)_+$ if and only if $a = \max\{0, b\}$.

(2) b_{ii}^k is the (i, i) entry of the diagonal matrix B^k,
$b_{ii}^k \neq 0$ for all i.
c_{ij}^k is the (i, j) entry of the matrix C^k, and $B^k + C^k = M$.

(3) q_i is the i-th component of the vector q.

(4) $u^{k,r} = (x_1^{k,r},\ldots,x_n^{k,r})^t$ where $x_i^{k,r}$ is the i-th component of x at time point r.

(5) $J^{k,r}$ indicates the index set of x-components x_i which are ready to update at time point r, that is $i \in J^{k,r}$.

In each time point r, we catch a new iterate $x^{k,r}$.

The 1st-phase can be terminated at arbitrary time point t_k. Let $t_k = \alpha$ and $x^{k,\alpha}$ be the final iterate of the 1st-phase in outer iteration k.

Remark: Notice that the above updating process is under the assumption of having no restrictions on the number of processors. If the number of processors is predetermined, and when the number of processors available is less than the number of components ready for the updating at some time point r, then we can arbitrarily choose a subset of $J^{k,r}$ to work on.

After completing the 1st-phase, the 2nd-phase follows. The tasks in the 2nd-phase are to perform a line search step and to update each x-component once.

The line search step is described as follows: Given a start point x^k and a direction $d^k = (x^{k,\alpha} - x^k)$ in each iteration k, find x'^k such that:

$$f(x'^k) = f(x^k + \lambda_k^* d^k) = \min_{\lambda_k \leq \lambda_{max}} \{f(x^k + \lambda_k d^k) \mid x^k + \lambda_k d^k \geq 0\} \tag{2.4}$$

where $\lambda_k{}^*$ is the optimal stepsize.

Remark: The direction d^k may be arbitrarily chosen. Intuitively, we believe that $d^k = (x^{k,\alpha} - x^k)$ is a good direction since $x^{k,\alpha}$ is typically a "better" iterate compared with x^k. (See the numerical results in section 4.)

One way to update each x-component once is by performing the formula (2.3), with $u^{k,r}$ replaced by x'^k and (B, C) being a regular splitting of M (i.e., (B — C) is positive definite.). Let x^{k+1} be such an updated iterate. One can find that x^{k+1} actually satisfies the following LCP:

$$v = q + C'^k x'^k + B'^k x \geq 0, \quad x \geq 0 \quad \text{and} \quad x^t v = 0 \tag{2.5}$$

where (B'^k, C'^k) is a regular splitting of the matrix M.

Moreover, the components which have been updated may directly enter into the 1st-phase of next outer iteration. Thus, we can actually combine this updating process with asynchronous computation of next outer iteration and generate more degree of parallelism.

In addition, if there exists communication memory for each processor, then the communication can be proceed concurrently with computation. As a result, the process will reduce the communication cost and the overall processing time.

3. Convergence Theorem.

In this section, we establish the convergency of the two-phase semi-asynchronous iterative algorithm mentioned in the previous section.

Lemma 3.1

Let M is a symmetric $n \times n$ matrix, B'^k and C'^k are the same defined as in (2.5), i.e. $(B'^k + C'^k) = M$ and $(B'^k - C'^k)$ is positive definite (p.d.)

If x'^k is obtained by (2.4) and x^{k+1} is obtained by (2.5)

then

(a) $f(x^{k+1}) \leq f(x'^k)$ for all k and

(b) $f(x^{k+1}) \leq f(x^k)$ for all k,

where $f(x) = q^t x + x^t M x/2$

pf:

(a) $f(x'^k) - f(x^{k+1})$

$$= (q + Mx'^k)(x'^k - x^{k+1}) - (x'^k - x^{k+1})^t M (x'^k - x^{k+1})/2$$

$$= (q + C'^k x'^k + B'^k x^{k+1})^t (x'^k - x^{k+1})$$

$$+ (B'^k x'^k - B'^k x^{k+1})^t (x'^k - x^{k+1})$$

$$- (x'^k - x^{k+1})^t M (x'^k - x^{k+1})/2$$

$$= (q + C'^k x'^k + B'^k x^{k+1})^t (x'^k) \tag{3.1}$$

$$+ (x'^k - x^{k+1})^t (2B'^k - M)(x'^k - x^{k+1})/2$$

$$\geq (x'^k - x^{k+1})^t (2B'^k - M)(x'^k - x^{k+1})/2 \tag{3.2}$$

$$= (x'^k - x^{k+1})^t (B'^k - C'^k)(x'^k - x^{k+1})/2$$

$$= (x'^k - x^{k+1})^t (B'^k - C'^k)(x'^k - x^{k+1})/2$$

$$\geq 0 \tag{3.3}$$

where (3.1) holds for x^{k-1} being the solution of (2.5),
(3.2) holds for both $(q + C'^k x'^k + B'^k x^{k+1})$ and x'^k are nonnegative.
(3.3) holds because we suppose $B'^k - C'^k$ is p.d.

(b) It is trivial since $f(x^{k+1}) \le f(x'^k)$ and $f(x'^k) \le f(x^k)$ by (2.4)

Lemma 3.2

Let f, x'^k and x^{k+1} be defined as in the lemma 3.1. If the sequence $\{f(x^k)\}$ converges then

$$\| x'^k - x^{k+1} \| \to 0 \text{ as } k \to \infty . \tag{3.4}$$

Pf: Since $\{f(x^k)\}$ converges and by lemma 3.1, we have

$$f(x^k) - f(x^{k+1}) \to 0 \quad \text{as } k \to \infty .$$

By lemma 3.1 (a) and (2.4) we have

$$f(x^{k+1}) \le f(x'^k) \le f(x^k), \text{ for all } k.$$

So,

$$f(x'^k) - f(x^{k+1}) \to 0 \quad \text{as } k \to \infty .$$

Moreover, the proof of lemma 3.1 shows

$$f(x'^k) - f(x^{k+1}) \ge (x'^k - x^{k+1})^t(B'^k - C'^k)(x'^k - x^{k+1})/2 \ge 0.$$

Thus,

$$\| x'^k - x^{k+1} \|_2 \to 0 \quad \text{as } k \to \infty .$$

Lemma 3.3

Let x^{k+1} be the same defined as in lemma 3.1 and x'^k be obtained by (2.4) with stepsize $|\lambda_k|$ bounded by $\varepsilon_k / \| x^{k,\alpha} - x^k \|$, where $\varepsilon_k \ge 0$ for all k and lim $\varepsilon_k = 0$

then

$$\|x'^k - x^k\| \to 0 \text{ as } k \to \infty .$$

pf.: $x'^k = x^k + \lambda_k * (x^{k,\alpha} - x^k)$

$$\|x'^k - x^k\| = |\lambda_k| \, \|x^{k,\alpha} - x^k\|$$

$$\leq \|x^{k,\alpha} - x^k\| \, \varepsilon_k / \|x^{k,\alpha} - x^k\|$$

$$= \varepsilon_k \to 0 \qquad \text{as } k \to \infty$$

and $\|x'^k - x^k\| \geq 0$ for all k

so

$$\|x'^k - x^k\| \to 0 \qquad \text{as } k \to \infty$$

Theorem 3.4

Let M be a symmetric and copositive matrix (that is, $x^tMx \geq 0$ for all $x \geq 0$) and each (B'^k, C'^k), $k = 1, 2, \ldots$, be a regular splitting of M. Let x'^k and x^{k+1} be as the same defined in lemma 3.3.

Then

(1) the sequence $\{f(x^k)\}$ converges. Where $f(x^k) = q^tx + x^tMx/2$.

(2) Any accumulation point x^* of $\{x^k\}$ is a solution to the LCP (1.1).

Pf: (1) By lemma 3.1 (b), we have

$f(x^{k+1}) \leq f(x^k)$

Since M is symmetric and copositive,

$f(x)$ is bounded below for $x \geq 0$ (see Eaves [1971]).

Thus, $\{f(x^k)\}$ converges.

(2) By lemma 3.2 and lemma 3.3, we have the results

$\|x'^k - x^{k+1}\| \rightarrow 0.$ as $k \rightarrow \infty$ and

$\|x'^k - x^k\| \rightarrow 0$ as $k \rightarrow \infty$

further,

$$0 \leq \|x^k - x^{k+1}\| \leq \|x^k - x'^k\| + \|x'^k - x^{k+1}\| \rightarrow 0 \text{ as } k \rightarrow \infty$$

then

$$\|x^k - x^{k+1}\| \rightarrow 0 \qquad \text{as } k \rightarrow \infty.$$

Let $\{x^{k_i+1}\}$ be a subsequence converging to x^* for each k_i,

we have:

$$u = q + C'^{k_i}x'^{k_i} + B'^{k_i}x^{k_i+1} \geq 0,\ x^{k_i+1} \geq 0 \text{ and } u^t x^{k_i+1} = 0 \tag{3.4}$$

passing the limit $k_i \rightarrow \infty$ and using (3.4) as well as the fact that $x^{k_i+1} \rightarrow x^*$ then we conclude that x^* solves the LCP(1.1).

Notice that the theorem 3.4 above does not provide the existence of the accumulation point of the sequence $\{x^k\}$. We give two approaches to quarantee that $\{x^k\}$ is bounded and thus has at least one accumulation point. This is referred to the proposition 2 in Pang and Yang [1988b]. We modify and restate the proposition as following.

Proposition 3.5

Let M be as given in theorem 3.4, if either

(a) the homogeneous linear complementarity problem

$$x \geq 0,\ Mx \geq 0 \text{ and } x^t Mx = 0$$

has zero as the unique solution;

or

(b) the level set $\{x \geq 0: f(x) \leq f(x^0)\}$ is bounded

then the sequence $\{x^k\}$ generated by the two-phase semi-asynchronous algorithm decribed in the previous section is bounded for $x \geq 0$.

Corollary 3.6

If M is positive definite the condition (a) in the proposition (3.5) holds.

The next result shows that the existence of the regular splittings of any real symmetric matrix M is quaranteed.

Proposition 3.7

Let M be a real symmetric matrix partitioned into blocks as $M = (M_{IJ})$ with each diagonal block M_{II} being square.

If we let $B = \lambda \times \text{Diag}\,(M_{II}) + k \times I$ and $C = M - B$ where k and λ are scalars, $\text{Diag}(M_{II})$ is the block diagonal matrix with each diagonal block being M_{II}, and I is an $n \times n$ identity matrix.

Then

the (B,C) is a regular splitting of M provided

$\lambda \times \beta + k > \gamma/2$

where γ is the spectral radius of M and
β is the smallest among the eigenvalues of the diagonal blocks M.

The proposition 3.6 is easy to be verified. We note that the condition $\lambda \times \beta + k > \gamma/2$ holds for every real symmetric matrix M, since k can be arbitrarily chosen.

Combining theorem 3.4 with proposition 3.5, we may state the following convergence result for the semi-asynchronous method.

Corollary 3.8

Let M satisfy the assumption in theorem 3.4 and either condition (a) or (b) in proposition 3.5 holds. Then the sequence $\{x^k\}$ generated by the semi-asynchronous method converges to the solution of the LCP(1.1).

It suffices to note that if M is positive definite, then the conditions required in theorem 3.4 and the condition (a) in proposition 3.5 are satisfied.

In the case of M being positive semi-definite, we give another way to quarantee the boundness of the sequence $\{x^k\}$. We note that B is positive definite if (B, C) is a regular splitting of M. Therefore, each subproblem (2.1) has a solution, according to the condition (a) in proposition 3.5. A matrix B such that each subproblem (2.1) has a solution is referred to as a Q-matrix in the terminology of linear complementarity theory (see Pang [1979]). A splitting (B, C) with B being a Q-matrix is called a Q-splitting. We then begin to restate the theorem, addressed in Lin and Pang [1987], as following:

Theorem 3.9

Let M be a symmetric positive semi-definite matrix and (B, C) be a regular Q-splitting of M. If there exists a vector x such that

$$q + Mx > 0. \tag{3.5}$$

Then for any $x^0 \geq 0$, the sequence $\{x^k\}$ generated by the semi-asynchronous method is bounded, and thus has an accumulation point.

We note that if M and (B, C) satisfy the theorem 3.9 and the constraint (3.5) holds for some x then the sequence $\{x^k\}$ generated by the semi-asynchronous method converges to the solution of LCP(1.1).

4. Implementation.

In this section, we report the numerical results simulated and implemented on the IBM 3090 vector computer system.

We assume the problem to be solved is following:

$$\textbf{Minimize} \quad p^t x + x^t x/2 \qquad \textbf{subject to} \quad Ax \geq b \text{ and } Cx = d \tag{4.1}$$

where p is a given n-vector,
b and d are given m-vectors and
A and C are given $m \times n$ matrices.

By the Karush-Kuhn-Tucker optimality conditions, we can easily transform problem (4.1) to the following equivalent LCP:

$$w = -b - Ap + AA^t u + AC^t v \geq 0, \; u \geq 0, \; u^t w = 0 \tag{4.2a}$$

$$-d - Cp + CA^t u + CC^t v = 0 \tag{4.2b}$$

If there exist u and v satisfying (4.2), then the solution to (4.1) is

$$x = -(p - A^t u - C^t v).$$

We note that the problem (4.2) is indeed the "mixed" LCP (see Lin and Pang [1987]), which is a special case of LCP, if we take the matrix M and the vector q to be

$$M = \begin{bmatrix} AA^t & AC^t \\ CA^t & CC^t \end{bmatrix} \text{ and } q = \begin{bmatrix} -b - Ap \\ -d - Cp \end{bmatrix} \text{ respectively.}$$

Notice that M herein is symmetric and positive semi-definite (i.e., $x^t M x \geq 0$ for all x). We can apply the semi-asynchronous method on this problem if M is chosen to satisfy some slater constraints. (see Theorem 3.9)

Since the structure of M and q provides a natural decomposition, it becomes obvious for one to divide the problem (4.2) into two groups, (4.2a) and (4.2b). Nevertheless, the process of further decomposition of the problem can be easily accomplished by just partitioning the Matrix M into many smaller submatrices.

In order to understand the performance of synchronized and asynchronous computing, we simulate and implement the algorithms proposed by Mangasarian and De Leone [1986b], Pang and Yang [1987a] and our two-phase semi-asynchronous algorithm from the viewpoint of two-stage (see Pang and Yang [1987a]). The first method (namely, the M-D method) performs one inner iteration in each outer iteration. The second method (namely, P-Y method) performs a number of inner iterations in each outer iteration (see Pang and Yang [1987a]). Both the two methods are synchronized and communicate at the end of each outer iteration. However, our semi-asynchronous method (namely, the Y-C method), performs asynchronous computing and exchanges information during each outer iteration. Since both of the M-D method and the semi-asynchronous method contain a line search step at the end of each outer iteration, the line search step is added in the P-Y method for the sake of comparison.

We describe the simulation for the synchronized parallel computation of the M-D method and the P-Y method by illustrating the computation in the outer iteration k. We solve the mixed LCP(4.2a) and (4.2b) respectively to obtain new iterates u^k and v^k. Therefore, the time for parallelly updating u and v is estimated to MAX(Tu, Tv), where Tu and Tv are the wall clock time for updating u and v respectively.

We then describe the simulation of the asynchronous computation in the 1st-phase of Y-C method. In the beginning, we perform both one update-u and update-v processes. We record Tu and Tv, the time needed for updating u and v respectively. If Tu > Tv, we will perform the update-v process next. Instead of using u^k, we use time amount Tv to regenerate a partially updated u^k in the next update-v process. The simulation process in the case of (Tu < Tv) is similar. Repeat the process until some termination criteria have been reached.

In the cases of our implementation, the values of vectors p and d and the entries of the matrices A and C are generated by a random number generating subroutine. The problem size is determined by the row number m, the column number n and the density of the matrices A and C (the density of a matrix is the percentage of nonzero entries in the matrix). We run examples of four classes ((m,n) = (40,500), (80,500), (80,750) and (90,750)). Each class has four different matrix densities (d =5%, 8%, 11% and 14%). The results are summarized in TABLE I and TABLE II. TABLE I lists the results of the above mentioned three parallel algorithms. One can find out that the semi-asynchronous method has better performance and is at least 15% faster than the other two synchronized methods. TABLE II lists the speed-up values of the P-Y method and semi-asynchronous method. The speed-up is defined as follow:

TABLE I: The results of three parallel algorithms

problem size	matrix density	M-D Method		P-Y Method		Y-C Method	
		iter.	CPU time	iter.	CPU time	iter.	CPU time
m=40 n=500	5%	114	1104	21	353	15	271
	8%	104	1492	23	564	18	472
	11%	85	1666	22	733	17	590
	14%	110	2734	20	847	16	704
m=80 n=500	5%	151*	2876	36	1175	27	930
	8%	151*	4406	33	1646	26	1365
	11%	151*	5943	35	2344	27	1898
	14%	151*	7374	38	3147	29	2517
m=80 n=750	5%	151*	4195	25	1212	17	899
	8%	151*	6491	28	2086	18	1477
	11%	151*	8634	28	2777	18	1951
	14%	151*	10834	26	3241	20	2677
m=90 n=750	5%	151*	4686	28	1497	21	1201
	8%	151*	7191	29	2373	19	1697
	11%	151*	9720	29	3196	20	2377
	14%	151*	12170	28	3885	20	2992

Remark: (1) The CPU times reported are in 1/1000 seconds.
(2) The Column "iter" reports the number of the outer iteration.
(3) "*" indicates the test problem reaches the maximum number of outer iteration.

TABLE II: The speed-up values

problem size	matrix density	sequential		P-Y Method			Y-C Method		
		iter.	time	iter.	time	speed-up	iter.	time	speed-up
m=40 n=500	5%	21	676	21	353	1.915	15	271	2.494
	8%	23	1079	23	564	1.913	18	472	2.286
	11%	22	1421	22	733	1.939	17	590	2.408
	14%	20	1643	20	847	1.940	16	704	2.301
m=80 n=500	5%	36	2237	36	1175	1.904	27	930	2.405
	8%	33	3207	33	1646	1.948	26	1365	2.349
	11%	35	4595	35	2344	1.960	27	1898	2.421
	14%	38	6183	38	3147	1.965	29	2517	2.456
m=80 n=750	5%	25	2348	25	1212	1.937	17	899	2.612
	8%	28	4103	28	2086	1.967	18	1477	2.778
	11%	28	5455	28	2777	1.964	18	1951	2.796
	14%	26	6415	26	3241	1.979	20	2677	2.396
m=90 n=750	5%	28	2915	28	1497	1.947	21	1201	2.420
	8%	29	4686	29	2373	1.975	19	1697	2.761
	11%	29	6271	29	3196	1.962	20	2377	2.638
	14%	28	7676	28	3885	1.976	20	2992	2.566

Remark: (1) The CPU times reported are in 1/1000 seconds.
(2) The Column "iter" reports the number of the outer iteration.
(3) The sequential method is the sequential version of the P-Y method.

$$\text{speed-up} = \frac{\text{TCPU(S)}}{\text{TCPU(P)}}$$

where TCPU(P) is the execution time of the parallel algorithm.
TCPU(S) is the execution time of the sequential algorithm.

Note that the speed-up of semi-asynchronous method are greater than 2 in most cases. We believe that such superior performance is obtained not only by parallel computation but also by fast convergence through asynchronous computing. Note that the number of outer iteration is quite less than that of the synchronized methods. As a result, the semi-asynchronous method does less work and converges faster than the other two synchronized methods do. This is the reason why the speed-up of semi-asynchronous method is larger than 2, in most cases.

5. Extensions.

The arrival of new generation super and parallel computers motivated the development of many parallel iterative methods for solving the LCP. People intend to exploit the parallel and the high-speed computation capability of the new generation computers. In this paper, we develop the two-phase semi-asynchronous iterative method for the LCP(1.1). It is worthy to extend our method to solve the nonsymmetric LCP using the similar ideas.

In addition, it is quite meaningful for one to implement the semi-asynchronous algorithm on real super and parallel machines in order to know the exact behavior of this algorithm. Therefore, we are now engaged in establishing the parallel computing environment with Transputers (see May and Shepherd [1987]). The numerical results and performance of implementing semi-asynchromous method on Transputer would be soon reported.

References

[1] B. C. Eaves [1971], "On Quadratic Programming", *Management Science 17*, pp. 698-711.

[2] H. T. Kung [1976], "Synchronized and Asynchronous Parallel Algorithms for Multiprocessors", in J. F. Traub ed., *Algorithms and Complexity: New Directions and Recent Results* (Academic Press) pp. 153-200.

[3] Y. Y. Lin and J. S. Pang [1987], "Iterative Methods for Large Convex Quadratic Programs: A Survey", *SIAM Journal on Control and Optimization 25*, pp. 383-411.

[4] O. L. Mangasarian and R. De Leone [1986a], "Parallel Successive Overrelaxation Methods for Symmetric Linear Complementarity Problems and Linear Programs", Mathematics Research Center Report #2947, University of Wisconsin (Madison, Wisconsin).

[5] O. L. Mangasarian and R. De Leone [1986b], "Parallel Gradient Projection Successive Over-relaxation for Symmetric Linear Complementarity Problems and Linear Programs", Technical Report #659, Department of Computer Sciences, University of Wisconsin (Madison, Wisconsin).

[6] D. May and R. Shepherd [1987], "The Inmos Transputer", Invited Paper on *Parallel Processing, State of Art Report 15:4*, C. Jesshope, R.J. O'Gorman and J.M. Stewart eds., Pergarmon Infotech. 1987.

[7] J. S. Pang [1979], "On Q-matrices", *Mathematical Programming 17* (1979), pp. 243-247.

[8] J. S. Pang [1982], "On the Convergence of a Basic Iterative Method for the Implicit Complementarity Problem", *Journal of Optimization Theory and Applications 37*, pp. 149-162.

[9] J. S. Pang and J. M. Yang [1988a], "Two-stage Parallel Iterative Methods for the Symmetric Linear Complementarity Problem," *Annals of Operations Research: Parallel Optimization on Novel Computer Architectures 14*, pp. 61-75.

[10] J. S. Pang and J. M. Yang [1988b], "Parallel Newton Methods for the Nonlinear Complementarity Problem", *Mathematical Programming: Parallel Methods in Mathematical Programming 42 No. 2*, pp. 407-420.