Computational Optimization and Applications, 17, 85–107, 2000
© 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

Parallel Synchronous and Asynchronous Space-Decomposition Algorithms for Large-Scale Minimization Problems

CHIN-SUNG LIU
CHING-HUAN TSENG chtseng@cc.nctu.edu.tw
Applied Optimum Design Laboratory, Department of Mechanical Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan, ROC

Received September 22, 1997; November 9, 1998

Abstract. Three parallel space-decomposition minimization (PSDM) algorithms, based on the parallel variable transformation (PVT) and the parallel gradient distribution (PGD) algorithms (O.L. Mangasarian, SIMA Journal on Control and Optimization, vol. 33, no. 6, pp. 1916–1925.), are presented for solving convex or nonconvex unconstrained minimization problems. The PSDM algorithms decompose the variable space into subspaces and distribute these decomposed subproblems among parallel processors. It is shown that if all decomposed subproblems are uncoupled of each other, they can be solved independently. Otherwise, the parallel algorithms presented in this paper can be used. Numerical experiments show that these parallel algorithms can save processor time, particularly for medium and large-scale problems. Up to six parallel processors are connected by Ethernet networks to solve four large-scale minimization problems. The results are compared with those obtained by using sequential algorithms run on a single processor. An application of the PSDM algorithms to the training of multilayer Adaptive Linear Neurons (Madaline) and a new parallel architecture for such parallel training are also presented.

Keywords: unconstrained minimization, parallel algorithm, parallel training, synchronous algorithm, asynchronous algorithm, decomposition method

1. Introduction

The purpose of this paper is to discuss three parallel algorithms including synchronous, sequential, and asynchronous parallel space-decomposition minimization (PSDM) algorithms for solving the unconstrained minimization problem

$$\min_{x \in \Re^n} f(x), \tag{1}$$

where $f : \Re^n \to \Re$ is a lower bounded and continuously differentiable function. The PSDM algorithms are based on the parallel variable transformation (PVT) algorithm [10] and the parallel gradient distribution (PGD) algorithm [16]. The basic ideas behind these parallel algorithms are the decomposition of the variable space $S \in \Re^n$ into q decomposed subspaces. If all of these q decomposed subproblems are not coupled from each other, these subproblems can be distributed among q parallel processors and solved independently. The final minimum solution can be obtained by directly combining the q minimum solutions.

However, if any two subproblems are coupled to each other, the iterative parallel algorithms discussed in this paper can be used instead.

Several decomposition methods for decomposing the minimization problem (1) into subproblems have been proposed. Bouaricha and Moré [2] decomposed the minimization problem (1) into $f(x) = \sum_{i=1}^{N} f_i(x)$, where $x \in \Re^n$, $f(x)$ is a partially separable function, and $f_i(x)$ depends only on a few components of x. Kibardin [13] also decomposed the minimization problem (1) into $f(x) = \sum_{i=1}^{N} f_i(x)$, where $x \in \Re^n$, and $f_i(x)$ are convex functions. Mouallif, Nguyen and Strodiot [18], Fukushima et al. [9] decomposed the minimization problem (1) into $f(x) = f_0(x) + \sum_{i=1}^{N} f_i(x)$, where $x \in \Re^n$, f_0 is a differentiable, strongly convex function with modulus α, and $f_i(x) : \Re^n \to \Re \cup \{+\infty\}$, are closed proper convex functions. In these studies, the original minimization problem (1) could be decomposed into subproblems that can be minimized on parallel processors. However, if minimization problem (1) cannot be decomposed into uncoupled minimization subproblems, it cannot be solved independently among parallel processors and some other parallel algorithms should be used.

Recently, Ferris and Mangasarian [7] proposed a parallel variable distribution (PVD) algorithm, which distributes the q blocks $x_1, \ldots, x_q$ of variable x among q processors, where $x_i \in \Re^{n_i}$ and $\sum_{i=1}^{q} n_i = n$. Mangasarian [16] also introduced the parallel gradient distribution (PGD) algorithm for assigning a portion of gradient $\nabla f(x)$ to q processors, in which the minimization problem in n-dimensional real space S was also decomposed into q n_i-dimensional subproblems, where $\sum_{i=1}^{q} n_i = n$, and all subproblems can be minimized by using various convergent algorithms. Solodov [22] further extended the PVD algorithm to a more flexible inexact PVD algorithm, in which the exact global solutions of all subproblems were replaced by inexact solutions.

More recently, Fukushima [10] proposed a more general framework that was called parallel variable transformation (PVT) algorithm. In this algorithm, the variables are transformed into spaces of smaller dimension, which altogether span the space of the original variables. At each iteration of the PVT algorithm, the multiple candidate solutions can be computed by solving the subproblems on the transformed spaces in the parallelization phase. That is, the approximate solution $y_i^{(k)} \in \Re^{m_i}$ to the minimization subproblem defined below can be obtained among parallel processors.

$$\min_{y_i \in \Re^{m_i}} \varphi_i^{(k)}(y_i) \equiv f\left(A_i^{(k)} y_i + x^{(k)}\right), \tag{2}$$

where $\sum_{i=1}^{q} m_i \geq n$ and $A_i^{(k)}$ is an $n \times m_i$ matrix. Then, the the synchronization phase will generate the next iterate from the candidate solutions obtained in the parallelization phase. That is, find an approximate solution $z^{(k)} \in \Re^{q+1}$ to the minimization problem

$$\min_{z \in \Re^{q+1}} \Psi^{(k)}(z) \equiv f\left(B^{(k)} z\right), \tag{3}$$

where $x^{(k+1)} = B^{(k)} z^{(k)}$ and $B^{(k)}$ is an $n \times (q+1)$ matrix with its columns being $x^{(k)}$ and $A_i^{(k)} y_i^{(k)}$, $i = 1, \ldots, q$. Fukushima [10] also showed that the PVD and PGD algorithms can be a special case of the general PVT framework. Furthermore, the PVT framework can

also be a natural extension of the block Jacobi method with a possible overlap of variables among subproblems.

In this paper, three parallel space-decomposition minimization (PSDM) algorithms, which are based on the PGD algorithm and the decomposition methods, are discussed. The PSDM algorithms are also special cases of the PVT algorithm by setting $m_i = n_i$ and $A_i = [0, \ldots, 0, I_{n_i}, 0, \ldots, 0]^T \in \Re^{n \times n_i}$, where the $n_i \times n_i$ identity matrix I_{n_i} appears in the position corresponding to the decomposed subvector. In these PSDM algorithms, the variable space S is decomposed into subspaces and the minimization problem (1) is also decomposed into subproblems for the corresponding subspace. It was shown that any convergent algorithm that satisfies some specified descent conditions can be applied to solve the decomposed subproblems among parallel processors [10, 16]. These parallel algorithms are experimented on four large scale problems. The speedups and the efficiency of the three parallel algorithms are compared with variant numbers of processors. In addition, the parallel training of multilayer Adaptive Linear Neurons (Madaline) [23] and its application to recognition problems are also presented to demonstrate the application of the PSDM algorithms.

Asynchronous algorithms that can run efficiently on parallel processors without stopping and waiting for data communication have been proposed by Fischer and Ritter [8], and Conforti and Musmanno [5, 6]. It has been shown that the parallel asynchronous algorithms can significantly speed up the sequential algorithms run on a single processor. Therefore, the asynchronous communication mechanism for the PSDM algorithm will be extended discussed in this paper.

This paper is organized as follows. In Section 2, a brief introduction to the SDM algorithm run on a single processor is given. The three parallel algorithm including synchronous, sequential, and asynchronous algorithms are given in Sections 3, 4, and 5, respectively. Numerical experiment results and discussion are given in Section 6, and an application of the PSDM algorithms to the parallel training of Madaline networks is given in Section 7. All experimented problems are listed in the appendix. The notation and terminology used in this paper are described below.

$S \in \Re^n$ denotes n-dimensional Euclidean variable space with ordinary inner product and associated two-norm $\|\cdot\|$. Italic characters denoting variables and vectors. For a real-value matrix A of any dimension, A^T denotes its transpose. For a differentiable function $f : \Re^n \to \Re$, ∇f denotes the n-dimensional vector of partial derivatives with respect to x, and $\nabla f_{S_i}(x_{S_i})$ denotes the n_i-dimensional vector of partial derivatives with respective to $x_{S_i} \in \Re^{n_i}$. For simplicity of notation, changes in the ordering of variables are allowed throughout this paper. Therefore, the variable vector $x \in S$ can be decomposed into subvectors. That is, $x = [x_{S_1}, \ldots, x_{S_q}]$, where $x_{S_i}, i = 1, \ldots, q$ are subvector or sub-component of x.

2. Space-decomposition minimization algorithm

The space-decomposition minimization (SDM) algorithm [14] is a sequential algorithm that can solve minimization problems (1). Although this algorithm was developed for a single processor, it is very suitable for parallel processing with minor modifications that

will be discussed in next few sections. Prior to describing these parallel algorithms, the sequential SDM algorithm, which is based on the non-overlapping subspace set and subspace minimization function, is interpreted below.

Definition 2.1 (non-overlapping subspace set). The original variable space S is spanned by $\{x \mid x \in \Re^n\}$. If the variable x is decomposed into $x = [x_{S_1}, \ldots, x_{S_q}]$, then the subspace S_i, spanned by the subvector $\{x_{S_i} \mid x_{S_i} \in \Re^{n_i}$, and $\sum_{i=1}^{q} n_i = n\}$, forms a non-overlapping subspace set $\{S_1, \ldots, S_q\}$. That is, $\bigcup_{i=1}^{q} S_i = S$, and $S_i \cap S_j = \emptyset$ if $i \neq j$.

From the Definition 2.1, the minimization function (1) can be decomposed as

$$f(x) = f_{S_i}\left(x_{S_i}, x_{\overline{S_i}}\right) + f_{\overline{S_i}}\left(x_{\overline{S_i}}\right), \tag{4}$$

where $x_{\overline{S_i}}$ is the complement vector of x_{S_i}, $f_{S_i}(x_{S_i}, x_{\overline{S_i}})$ is the subspace minimization function in the subspace S_i and $f_{\overline{S_i}}(x_{\overline{S_i}})$ is the complement subspace minimization function.

Corollary 2.2. *From* (4), $f_{\overline{S_i}}(x_{\overline{S_i}})$ *is only a function of* $x_{\overline{S_i}}$. *Therefore, it can be removed from the minimization subproblem if* $x_{\overline{S_i}}$ *is invariant in the subspace* S_i.

Corollary 2.2 can be efficiently applied to some simple problems that can be decomposed into

$$f(x) = \sum_{i=1}^{q} f_{S_i}\left(x_{S_i}\right), \tag{5}$$

where $\{S_1, \ldots, S_q\}$ is a non-overlapping subspace set. Therefore, the minimization problem (1) can be decomposed into q uncoupled subproblems that can be solved independently and a lot of processor time and memory resources can be saved. In Section 7, it will be demonstrated that the training of multilayer Adaptive Linear Neurons (Madaline) can be decomposed into uncoupled subproblems. However, most minimization problems cannot be decomposed into uncoupled subproblems, so iterative algorithms must be used. The iterative space-decomposition minimization (SDM) algorithm is described below and will be extended to parallel algorithms in the following sections.

Algorithm 2.3. (space-decomposition minimization (SDM) algorithm)

Step 1. Decompose the variable space into a non-overlapping subspace set $\{S_1, \ldots, S_q\}$ and derive the subspace minimization functions $f_{S_i}(x_{S_i}, x_{\overline{S_i}})$, for $i = 1 \cdots q$.
Step 2. Choose the starting point x^1, where $x^1 = [x^1_{S_1}, \ldots, x^1_{S_q}]$ and set $k = 1$.
Step 3. For $i = 1$ to q, solve one or more steps of the minimization subproblems $f_{S_i}(x^{(k)}_{S_i}, x_{\overline{S_i}})$ using any convergent descent algorithm.
Step 4. Apply convergence criterion, such as $\|\nabla f_{S_i}(x_{S_i})\| \leq \varepsilon$, to all subspaces. If the convergence criterion is satisfied for all subspaces, the minimum solution has been found as $x^* = [x^*_{S_1}, \ldots, x^*_{S_q}]$; otherwise, set $k = k + 1$ and go to Step 3.

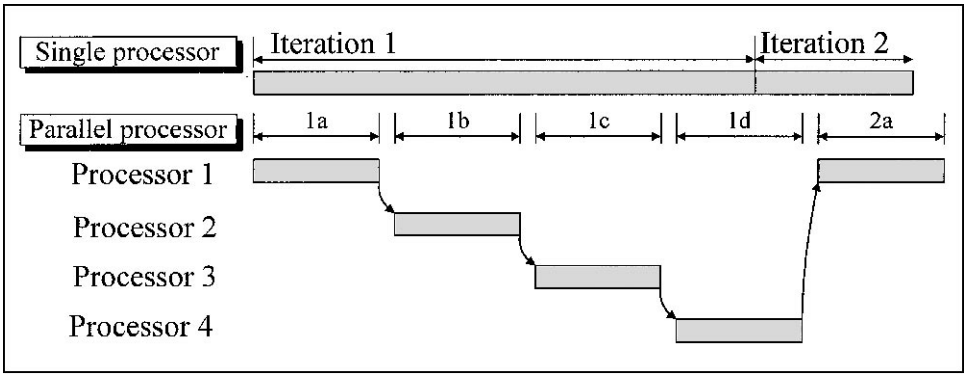

Figure 1. Decomposing the iterations of SDM algorithm into sub-iterations and distributing them among parallel processors.

3. Synchronous PSDM algorithm

Minimization problem (5) presented in the preceding section shows that if a minimization problem can be decomposed into q uncoupled subproblems, they can be distributed among q processors and run independently without any data communication; otherwise, the iterative algorithms must be employed. As shown in figure 1, every iteration of the SDM algorithm can be divided into q sub-iterations that can be distributed among q processors. Only the coupled variables are communicated to other processors and the running sequence is exactly alike to that of SDM algorithm. Although this approach doesn't benefit from parallel computing, it does present the concept of running the SDM algorithm on parallel processors. Using this concept, the synchronous PSDM algorithm is presented below.

Algorithm 3.1. (synchronous PSDM algorithm)

Step 1. Decompose the variable space S into a non-overlapping subspace set $\{S_1, \ldots, S_q\}$ and derive the q subspace minimization functions $f_{S_i}(x_{S_i}, x_{\overline{S_i}})$ which are distributed among q parallel processors.

Step 2. Initialize the q processors' starting points $x_{S_i}^1$, $i = 1 \cdots q$, and set $k = 1$.

Step 3. Simultaneously update all variable $x_{S_i}^{(k)}$ on all q processors using the convergent algorithms, such as the conjugate gradient method, that satisfies

$$-\nabla f_{S_i}\left(x_{S_i}^{(k)}, x_{\overline{S_i}}\right)^T d_{S_i}^{(k)} \geq \sigma_1\left(\left\|\nabla f_{S_i}\left(x_{S_i}^{(k)}, x_{\overline{S_i}}\right)\right\|\right) \geq 0, \tag{6}$$

and

$$f_{S_i}\left(x_{S_i}^{(k)}, x_{\overline{S_i}}\right) - f_{S_i}\left(x_{S_i}^{(k+1)}, x_{\overline{S_i}}\right) \geq \sigma_2\left(-\nabla f_{S_i}\left(x_{S_i}^{(k)}, x_{\overline{S_i}}\right)^T d_{S_i}^{(k)}\right) \geq 0, \tag{7}$$

where $\sigma_1(\cdot)$, $\sigma_2(\cdot)$ are forcing functions [10, 16], and $d_{S_i}^{(k)}$ is the search direction in the subspace S_i.

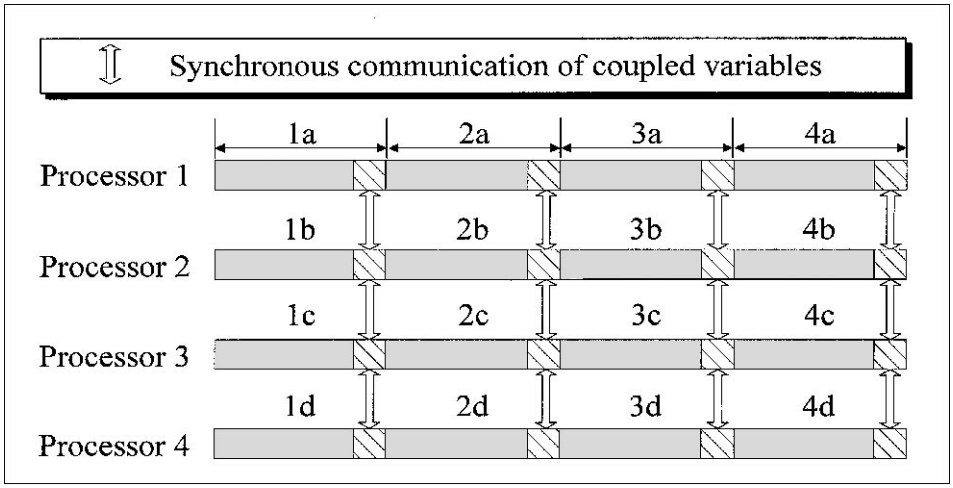

Figure 2. Synchronous communication of coupled variables among processors.

Step 4. Simultaneously communicate coupled variables among the$q$ parallel processors as shown in figure 2. Then set

$$x_{S_i}^{(k+1)} = x_{S_i}^{(k)} + v_i^{(k)} \alpha_i^{(k)} d_{S_i}^{(k)} \tag{8}$$

for all q processors, where $\alpha_i^{(k)}$ is the step size found in subspace S_i, and

$$\sum_{i=1}^{q} v_i^{(k)} = 1, \quad v_i^{(k)} \geq \delta > 0. \tag{9}$$

Step 5. Check the convergence criterion $\|\nabla f_{S_i}(x_{S_i}^{(k)})\| \leq \varepsilon$ for all subspaces. If the convergence criterion is satisfied for all subspaces, the minimum solution has been found. That is, $x^* = [x_{S_1}^*, \ldots, x_{S_q}^*]$; otherwise, set $k = k + 1$ and go to Step 3.

The processes of communicating coupled variables among parallel processors for the synchronous PSDM algorithm are shown in figure 2. In the final stage of every iterations, all coupled variables are communicated among parallel processors, and the next iteration starts immediately after any one processor has obtained the required data. Although the Algorithm 3.1 was derived for convex minimization problems, it can be extended to non-convex minimization problems by taking the best solution or some better solution from the solutions of the q decomposed subproblems [10, 16].

In the PSDM algorithm, the dimension of any decomposed subspace can be varied dynamically during running. That is, if the jth processor occasionally becomes busy, some variables in S_j can be redistributed among the other processors during running, thus preventing suspension of the entire algorithm due to some processors occasionally become

busy on other tasks. In addition, any local variable on any processor can be updated as many times as possible before all q processors are ready to run Step 4 of Algorithm 3.1.

Although the synchronous PSDM algorithm benefits from the parallel processing, numerical experiments show that some processors may be idle due to the synchronization overhead for the communication of coupled variables among processors. In addition, the strong convex combinations (9) are too conservative [1], and some calculation results may be made obsolete for nonconvex problems, which can decrease the overall efficiency of the parallel algorithm. To eliminate these disadvantages and extend it to more general minimization problems, the sequential PSDM algorithm was devised.

4. Sequential PSDM algorithm

As shown in figure 1, any iteration of the SDM algorithm run on a single processor can be divided into q sub-iterations that can be distributed among q parallel processors. This approach presents the concept of parallel processing. If only part of variables are coupled to other subproblems, the coupled variables can be updated first. Then, the other variables are updated. Meanwhile, the coupled variables are communicated to the next processor. Therefore, any iteration of the SDM algorithm can be divided into q sub-iterations, which can then be distributed to q parallel processors and run concurrently. As shown in figure 3(A), the original iteration of SDM algorithm is divided into 4 sub-iterations and distributed to 4 processors. It also shows that the next processor is triggered to run immediately after the coupled variables have been updated in the previous processor, and the processor is continuing to update uncoupled variables without waiting. This approach can get more benefits of parallel processing and is presented in the following algorithm.

Algorithm 4.1. (sequential PSDM algorithm)

Step 1-2. are the same as those in the parallel synchronous Algorithm 3.1.

Step 3. The first processor starts updating all coupled variables, then triggers Step 4 to run. Meanwhile, the coupled variables are communicated to the next processor and triggering that processor to update the coupled variables, and so on, until the last processor has been triggered to run.

Step 4. Update uncoupled variables immediately after coupled variables have been updated in Step 3 for all parallel processors.

Step 5. After the last processor has updated all variables, check the convergence criterion $\|\nabla f_{S_i}(x_{S_i}^{(k)})\| \leq \varepsilon$ for all decomposed subspaces. If convergent criterion has been satisfied for all decomposed subspaces, the minimum solution has been found. That is, $x^* = [x_{S_1}^*, \ldots, x_{S_q}^*]$; otherwise, set $k = k + 1$, communicating coupled variables to the first processor and go to Step 3.

The running sequences of the sequential PSDM algorithm are alike to that of SDM algorithm except that they are distributed among q parallel processors. Therefore, all convergence conditions are the same as those for the SDM algorithm. As shown in figure 3(A) and (B), the sequential PSDM algorithm can get more benefits of parallel processing if the

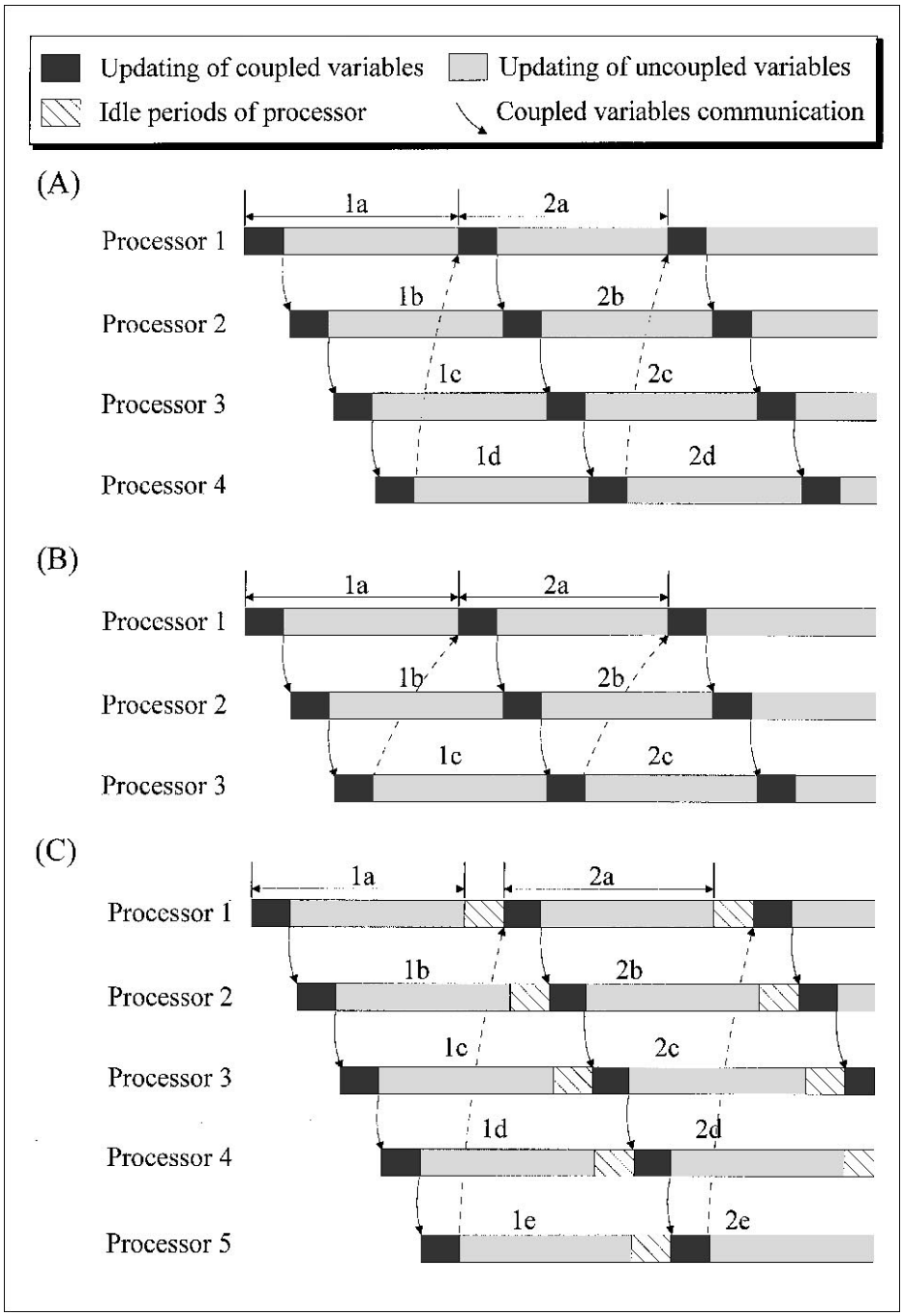

Figure 3. Sequential communication of coupled variables among processors with (A) maximum number of processors without resulting idle time, (B) less processors, (C) too many processors that will result in idle time of processors.

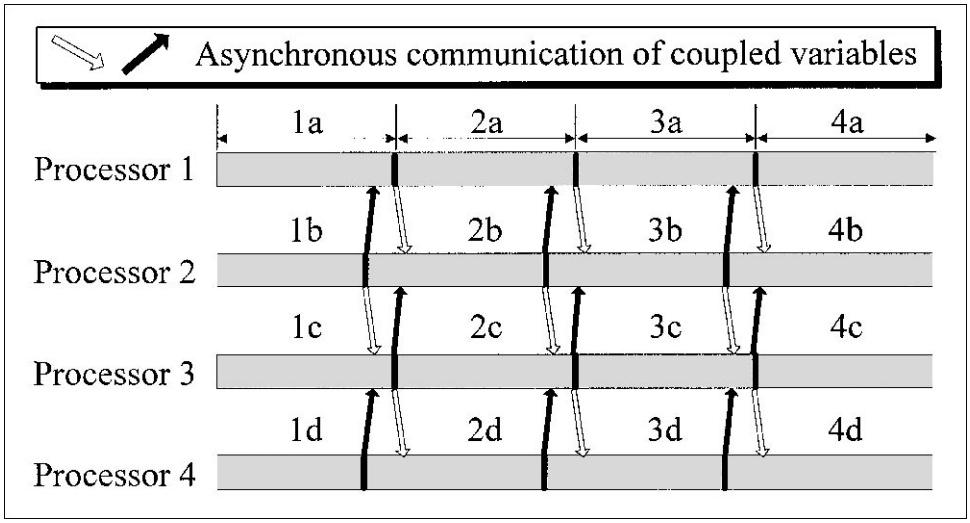

Figure 4. Asynchronous communication of coupled variables among processors.

minimization problem is decomposed into subproblems, which are distributed to a proper number of processors. However, too many processors may result in some processors being idle as shown in figure 3(C), and the overall efficiency of the sequential PSDM algorithm will decrease. To reduce idle processor time, the following methods can be used.

1. Trigger the first processor to run the next iteration as soon as possible. For example, if processor 1 is not coupled to the variables of processor 5, then processor 1 can be triggered to run the next iteration as soon as processor 4 has finished updating the coupled variable.
2. Increase the iterative times for all processors. That is, update the local variables for all processors more than once before triggering the next processor to run.
3. Run all processors simultaneously and communicate coupled variables asynchronously, as shown in figure 4. That is, all parallel processors can run continuously without caring when data are communicated in or out among processors. This method can avoid idle processor periods.

Numerical experiments show that method 3 can get the most benefit from parallel processing. However, asynchronous updating may result in oscillation of the minimization function value, which may affect the convergence characteristics of the original algorithm [4]. Therefore, additional check condition presented in the following section is required.

5. Asynchronous PSDM algorithm

In general, after appropriate initialization, the asynchronous algorithm can run on parallel processors without waiting for other processors to communicate coupled variables. That is,

each processor runs a local and independent algorithm using the data currently available from other processors. The efficiency of the parallel asynchronous operation can be improved since the overhead of the synchronous algorithm is significantly reduced. Therefore, the processor resources in multiprocessor systems can be used more efficiently [4]. The asynchronous PSDM algorithm is presented below.

Algorithm 5.1. (asynchronous PSDM algorithm)

Step 1-2. are the same as those in the synchronous PSDM Algorithm 3.1.

Step 3a. Update all variables on all processors simultaneously by using any convergent algorithm that satisfies (6) and (7).

Step 3b. Write the most recently updated coupled variables to other processors asynchronously as shown in figure 4.

Step 3c. Read the most recently updated coupled variables from other processors asynchronously, then calculate

$$\Delta f_{S_i} = f_{S_i}\left(x_{S_i}^{(k)}, x'_{\overline{S_i}}\right) - f_{S_i}\left(x_{S_i}^{(k)}, x_{\overline{S_i}}\right), \tag{10}$$

where x'_{S_i} is the coupled variables read from other processors. If

$$\Delta f_{S_i} \leq 0, \tag{11}$$

then set $x_{\overline{S_i}} = x'_{\overline{S_i}}$; otherwise obsolete $x'_{\overline{S_i}}$ and continue reading the most recently updated data from other processors until the descent condition (11) satisfied.

Step 3d. Check the convergence criterion $\|\nabla f_{S_i}(x_{S_i}^{(k)})\| \leq \varepsilon$ asynchronously for all subspace. If the convergence criterion has been satisfied for all subspaces, the minimum solution has been found. That is, $x^* = [x^*_{S_1}, \ldots, x^*_{S_q}]$; otherwise, running Steps 3a–3d concurrently on all parallel processors.

From (7) and (11) give that

$$f_{S_i}\left(x_{S_i}^{(k)}, x_{\overline{S_i}}\right) \geq f_{S_i}\left(x_{S_i}^{(k)}, x'_{\overline{S_i}}\right) \geq f_{S_i}\left(x_{S_i}^{(k+1)}, x'_{\overline{S_i}}\right). \tag{12}$$

Therefore, either updating $x_{S_i}^{(k)}$ or $x_{\overline{S_i}}$ will decrease the value of subspace minimization function $f_{S_i}(x_{S_i}^{(k)}, x_{\overline{S_i}})$, where $x_{S_i}^{(k)}$ is updated to $x_{S_i}^{(k+1)}$ in Step 3a of Algorithm 5.1, and $x_{\overline{S_i}}$ is updated to $x'_{\overline{S_i}}$ in Step 3c of Algorithm 5.1. That is, the sequence $\{f_{S_i}(x_{S_i}^{(k)}, x_{\overline{S_i}})\}$ is a non-increasing sequence for either $x_{S_i}^{(k)}$ or $x_{\overline{S_i}}$ updated. Therefore, the convergence of the Algorithm 5.1 can follow the analysis of the Refs. [10, 16].

In Algorithm 5.1, Steps 3a–3d are executed concurrently on all processors. All variables in all subspaces are continuously updated without waiting, and coupled variables are read from or written to other processors asynchronously. As described in the preceding section, asynchronous communication of coupled variables can avoid idle periods of processors.

6. Numerical experiment results and discussion

To demonstrate these parallel algorithms, four large-scale test problems [17] with the number of variables varying from 100 to 10000 were experimented with Algorithms 3.1, 4.1 and 5.1. Processor times for the SDM algorithm run on a single processor are shown in Table 1. Numerical results for the synchronous, sequential, and asynchronous PSDM algorithms with variant numbers of parallel processors are shown in Tables 1–3. In these tables, all processor times were measured in seconds and the notation used in the tables is defined below:

NV := number of variables;
T1 := processor time for the SDM algorithm run on a single processor;
Tn := processor time for the synchronous, sequential, and asynchronous PSDM algorithm with n processors;

Table 1. Numerical results for the SDM and PSDM algorithm with 1 and 2 processors.

			T2 (sec.)			Speedup/Efficiency (%)		
Problem	NV	T1 (sec.)	Synchronous PSDM	Sequential PSDM	Asynchronous PSDM	Synchronous PSDM	Sequential PSDM	Asynchronous PSDM
#1	100	1.572	1.853	1.962	1.282	0.848/42.42	0.801/40.06	1.226/61.31
	500	6.429	3.696	3.435	3.435	1.739/86.97	1.872/93.58	1.872/93.58
	1000	12.689	6.810	6.809	6.829	1.863/93.16	1.864/93.18	1.858/92.91
	5000	65.093	35.100	35.080	35.291	1.855/92.73	1.856/92.78	1.844/92.22
	10000	133.685	69.230	69.210	69.260	1.931/96.55	1.932/96.58	1.930/96.51
#2	100	0.220	2.393	2.113	1.412	0.092/4.60	0.104/5.21	0.156/7.79
	500	4.787	6.649	6.018	4.166	0.720/36.00	0.795/39.77	1.149/57.45
	1000	15.472	10.014	9.854	7.411	1.545/77.25	1.570/78.51	2.088/104.39
	5000	115.280	93.354	93.895	91.282	1.235/61.74	1.228/61.39	1.263/63.15
	10000	401.107	325.398	324.666	321.632	1.233/61.63	1.235/61.77	1.247/62.36
#3	100	3.545	5.778	5.819	3.655	0.614/30.68	0.609/30.46	0.970/48.50
	500	25.587	18.056	17.325	15.092	1.417/70.86	1.477/73.84	1.695/84.77
	1000	54.768	32.407	32.487	30.895	1.690/84.50	1.686/84.29	1.773/88.64
	5000	293.241	156.945	157.457	154.623	1.868/93.42	1.862/93.12	1.896/94.83
	10000	597.890	314.823	315.944	312.489	1.899/94.96	1.892/94.62	1.913/95.67
#4	100	0.251	0.621	0.601	0.461	0.404/20.21	0.418/20.88	0.544/27.22
	500	1.372	1.272	1.302	1.052	1.079/53.93	1.054/52.69	1.304/65.21
	1000	2.894	2.063	1.893	1.743	1.403/70.14	1.529/76.44	1.660/83.02
	5000	15.222	7.921	7.942	7.631	1.922/96.09	1.917/95.83	1.995/99.74
	10000	28.711	16.023	15.733	15.623	1.792/89.59	1.825/91.25	1.838/91.89

Table 2. Numerical results for the PSDM algorithm with 4 processors.

		T2 (sec.)			Speedup/Efficiency (%)		
Problem	NV	Synchronous PSDM	Sequential PSDM	Asynchronous PSDM	Synchronous PSDM	Sequential PSDM	Asynchronous PSDM
#1	100	3.155	3.505	1.522	0.498/12.456	0.449/11.213	1.033/25.821
	500	3.225	3.505	2.223	1.993/49.837	1.834/45.856	2.892/72.301
	1000	3.855	3.766	4.236	3.292/82.289	3.369/84.234	2.996/74.888
	5000	17.075	17.305	17.525	3.812/95.305	3.762/94.038	3.714/92.857
	10000	33.508	33.759	33.829	3.990/99.741	3.960/99.000	3.952/98.795
#2	100	3.886	2.233	1.732	0.057/1.415	0.099/2.463	0.127/3.176
	500	8.553	4.887	3.455	0.560/13.992	0.980/24.488	1.386/34.638
	1000	9.975	5.899	4.587	1.551/38.777	2.623/65.570	3.373/84.325
	5000	34.389	31.916	29.943	3.352/83.806	3.612/90.300	3.850/96.250
	10000	113.423	110.509	110.970	3.536/88.410	3.630/90.741	3.615/90.364
#3	100	9.224	8.803	3.555	0.384/ 9.608	0.403/10.068	0.997/24.930
	500	13.239	12.608	8.332	1.933/48.317	2.029/50.736	3.071/76.773
	1000	20.179	21.041	15.853	2.714/67.853	2.603/65.073	3.455/86.369
	5000	82.689	82.198	78.332	3.546/88.658	3.567/89.187	3.744/93.589
	10000	168.712	166.820	156.755	3.544/88.596	3.584/89.601	3.814/95.354
#4	100	0.861	1.532	0.410	0.292/7.288	0.164/4.096	0.612/15.305
	500	1.162	1.282	0.471	1.181/29.518	1.070/26.755	2.913/72.824
	1000	1.673	1.652	0.841	1.730/43.246	1.752/43.795	3.441/86.029
	5000	4.447	4.086	3.756	3.423/85.575	3.725/93.135	4.053/101.318
	10000	8.532	8.232	7.601	3.365/84.127	3.488/87.193	3.777/94.432

Speedup and Efficiency defined below can be used for comparing the performance between different algorithms [20].

$$\text{Speedup} \equiv \frac{\text{T1}}{\text{T}n}, \tag{13}$$

$$\text{Efficiency} \equiv \frac{\text{Speedup}}{\text{Processor number } n}. \tag{14}$$

The numerical results were obtained using Pentium 166 MHz machines connected by Ethernet Networks with 10Mbps transmission speed. The MIMD (multiple instructions multiple data) parallel computing systems with distributed memory were used and the convergence criterion is set to $\varepsilon = 10^{-4}$. Figures 5–8 show the speedups for NV = 1000 and 10000 with respect to variant number of parallel processors. These figures demonstrated that the synchronous PSDM algorithm can have good performance with two and four processors. However, as the processor number is greater than four processors, the efficiency decreases

Table 3. Numerical results for the PSDM algorithm with 6 processors.

		T2 (sec.)			Speedup/Efficiency (%)		
Problem	NV	Synchronous PSDM	Sequential PSDM	Asynchronous PSDM	Synchronous PSDM	Sequential PSDM	Asynchronous PSDM
#1	100	3.545	3.394	1.983	0.443/7.391	0.463/7.720	0.793/13.212
	500	4.086	3.856	3.866	1.573/26.224	1.667/27.788	1.663/27.716
	1000	4.416	3.845	4.006	2.873/47.890	3.300/55.002	3.167/52.792
	5000	20.890	11.887	11.987	3.116/51.933	5.476/91.266	5.430/90.505
	10000	36.042	22.552	22.963	3.709/61.819	5.928/98.798	5.822/97.029
#2	100	4.356	3.234	1.842	0.051/0.842	0.068/1.134	0.119/1.991
	500	8.593	9.143	2.143	0.557/9.285	0.524/8.726	2.234/37.230
	1000	11.046	13.520	3.555	1.401/23.345	1.144/19.073	4.352/72.536
	5000	33.067	26.559	20.399	3.486/58.104	4.341/72.342	5.651/94.188
	10000	191.916	92.553	90.640	2.090/34.834	4.334/72.230	4.425/73.755
#3	100	11.376	14.611	3.005	0.312/5.194	0.243/4.044	1.180/19.662
	500	15.042	13.139	6.049	1.701/28.351	1.947/32.457	4.230/70.499
	1000	20.300	18.838	11.626	2.698/44.966	2.907/48.455	4.711/78.514
	5000	101.546	60.397	55.169	2.888/48.129	4.855/80.920	5.315/88.589
	10000	223.091	115.416	112.632	2.680/44.667	5.180/86.338	5.308/88.472
#4	100	1.412	1.202	1.251	0.178/2.963	0.209/3.480	0.201/3.344
	500	1.773	1.032	0.460	0.774/12.897	1.329/22.158	2.983/49.710
	1000	1.732	1.793	0.641	1.671/27.848	1.614/26.901	4.515/75.247
	5000	5.348	3.275	2.744	2.846/47.438	4.648/77.466	5.547/92.456
	10000	11.356	5.778	5.588	2.528/42.138	4.969/82.817	5.138/85.633

significantly. This is due to the synchronization overhead will become significant when the parallel processor number increased. The synchronization overhead can be decreased by using high speed networks, such as the 100 Mbps Ethernet Networks. However, other cheaper method to overcome the synchronization overhead is by the asynchronous PSDM algorithm. Figures 5–8 also shown that the asynchronous PSDM algorithm can get the most benefit from parallel processors. The efficiency is decreased slightly as the processor number increased. It can also be observed from these figures that the sequential PSDM algorithm can have good performance for large scale problems (NV = 10000). However, for the small scale problems (NV = 1000), the synchronization overhead will also become significant when the parallel processor number increased.

For the synchronous and the sequential PSDM algorithms, the coupled variables can be read and wrote in synchronized. Therefore, the communication processes can be easily tracked by programming. However, the asynchronous PSDM algorithm can result access violation due to the same memory can not be simultaneously read and wrote by more than one processor. Therefore, the direct connection (figure 9(A)) for the datum communication

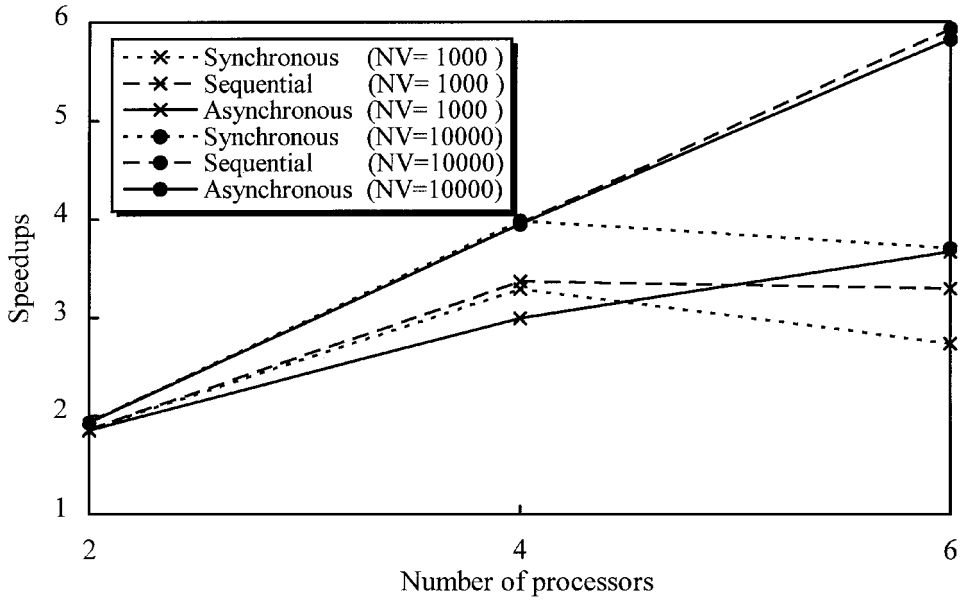

Figure 5. Speedups of the PSDM algorithm for Problem #1.

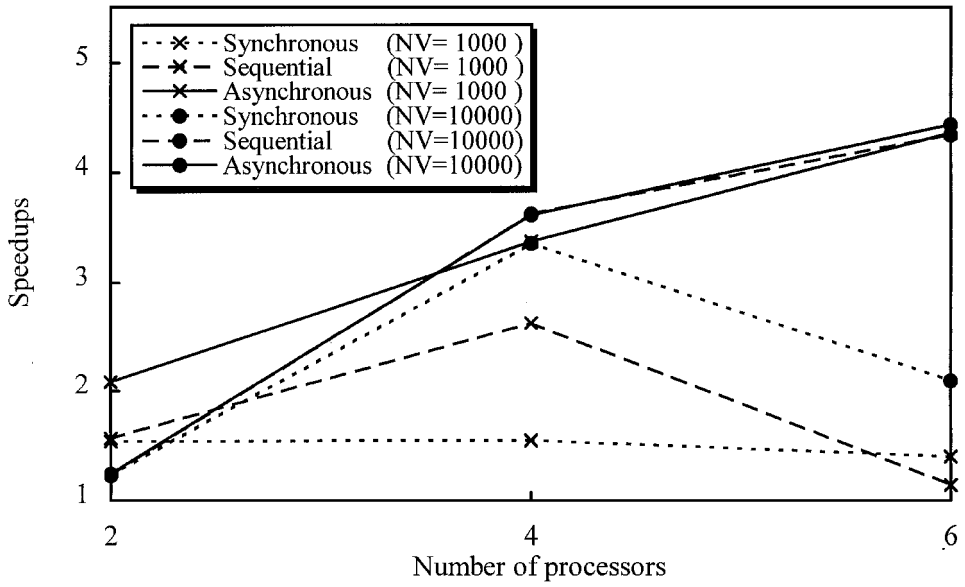

Figure 6. Speedups of the PSDM algorithm for Problem #2.

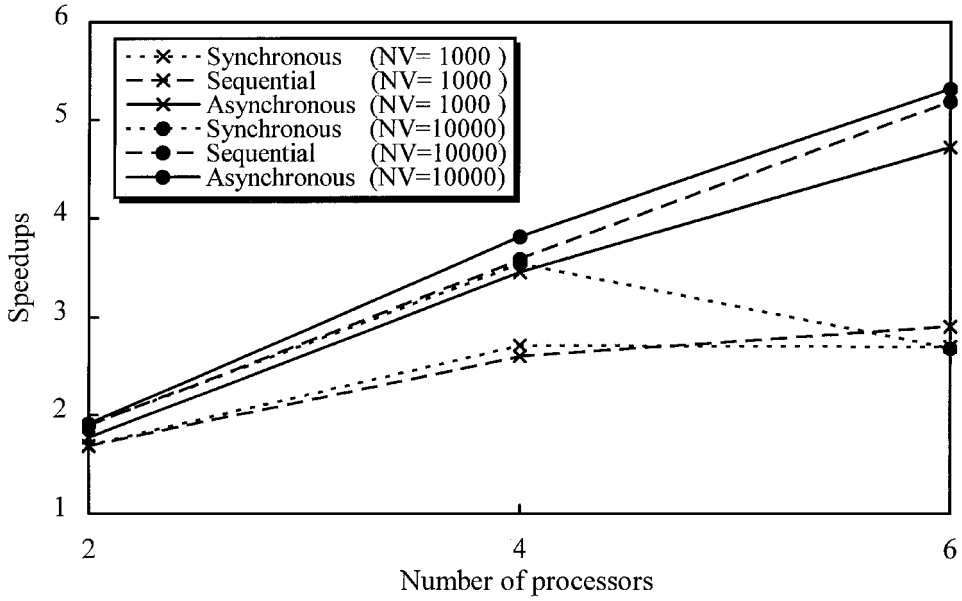

Figure 7. Speedups of the PSDM algorithm for Problem #3.

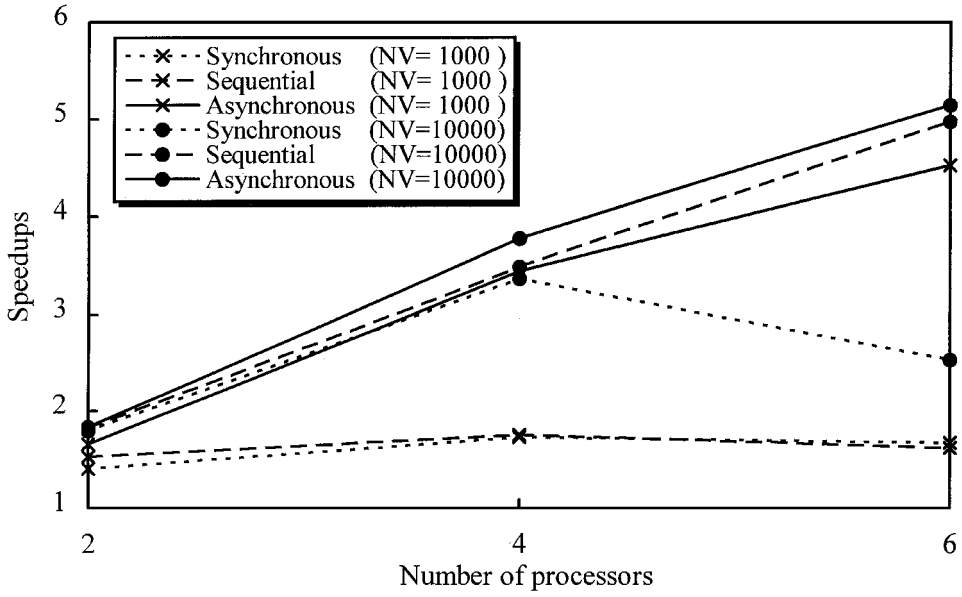

Figure 8. Speedups of the PSDM algorithm for Problem #4.

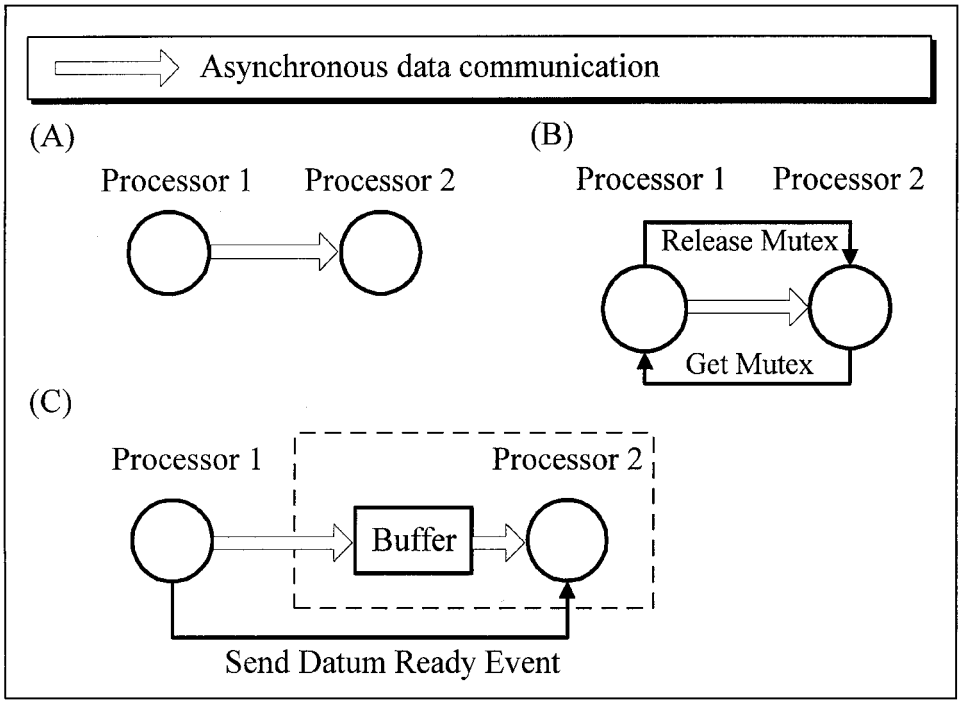

Figure 9. Communication mechanism of data for asynchronous PSDM algorithm (A) direct connection (B) through Mutex object (C) through buffer and Event.

among parallel processors can suddenly result in access violation. Two methods, as shown in figure 9(B) and (C), can overcome such problem:

(1) The "Mutex" object, as shown in figure 9(B), is an unique object that can be owned by only one processor. A memory block can be assigned a unique Mutex object. Therefore, the processor 1 can access to such memory block of processor 2 only when the processor 1 obtains the Mutex object from processor 2, and vice versa. The Mutex object should be released to original processor as soon as possible to allow the other processor to access such memory block.
(2) Figure 9(C) shows that the data are communicated to the data buffer instead to processor 2 directly. When the data are fully communicated, the "Data Ready Event" is sent from processor 1 to processor 2. Then, the processor 2 can be triggered to access data from the data buffer.

Although the second method will take more buffer memory, it does not result the waiting for Mutex object. Since the "Data Ready Event" is sent and received asynchronously, the second method does take the most benefit of asynchronous mechanism. In this "Asynchronously Event Driven" mechanism, the computation can take place on the foreground

task and meanwhile, the data can be communicated among processors in the background task. When the data are communicated ready, the background task can send an "Event" to the foreground task to read the data. No processor will be idle in such multitask computation and communication processes. Therefore, this method is particularly suited for the modern multitask (multithread) and event driven computer operation systems.

7. Application of PSDM algorithms to parallel Madaline networks training

7.1. Introduction

The space decomposition minimization algorithm can be applied to the training of multilayer neural networks. The multilayer Adaptive Linear Neurons (Madaline) [23] is a typical multilayer neural networks that can be trained by minimizing the mean-squared error function, which is generally defined as

$$E(W) \equiv \frac{1}{2} \sum_{p=1}^{P} \sum_{j} [\hat{u}_{pj} - u_{pj}]^2, \tag{15}$$

where P is the number of training patterns, $\hat{u}_{pj}$ and u_{pj} are the desired linear outputs and actual linear outputs of the Madaline networks, respectively. The minimization of the mean-squared error function (15) is a typical unconstrained minimization problem. Therefore, a lot of unconstrained minimization methods can be applied to the training of the multilayer neural networks.

To apply the space decomposition minimization algorithm to the training of Madaline networks, the multilayer mean-squared error function is defined as

$$\widetilde{E}(W, \hat{u}) \equiv \frac{1}{2} \sum_{\ell=1}^{L} \sum_{j} \left[E_j^{(\ell)}\left(w_j^{(\ell)}, \hat{u}_j^{(\ell)}\right)\right]^2, \tag{16}$$

where $E_j^{(\ell)}(w_j^{(\ell)}, \hat{u}_j^{(\ell)})$ is the mean-squared error function of the jth neuron in the ℓth layer. The linear outputs of the ℓth layer neurons are computed by $u_j^{(\ell)} = [x^{(\ell-1)}]^T w_j^{(\ell)}$, where $x^{(\ell-1)}$ and $w_j^{(\ell)}$ are the input vector and the adaptive weights of the ℓth layer neurons, respectively. The relationship between the input vector $x^{(\ell-1)}$ and the desired linear output $\hat{u}_j^{(\ell-1)}$ is $x^{(\ell-1)} = S(\hat{u}_j^{(\ell-1)})$, where $S(\cdot)$ is the activation function of the neurons in the hidden layers.

From the SDM Algorithm 2.4, the minimization of the mean-squared error function (16) can be decomposed into two subproblems. The first subproblem is the mean-squared error function

$$\tilde{E}_{\hat{u}}(\hat{u}) = \frac{1}{2} \sum_{\ell=1}^{L} \sum_{j} \left[E_j^{(\ell)}\left(w_j^{(\ell)*}, \hat{u}_j^{(\ell)}\right)\right]^2, \tag{17}$$

and the second subproblem is the mean-squared error function

$$\tilde{E}_W(W) = \frac{1}{2}\sum_{\ell=1}^{L}\sum_{j}\left[E_j^{(\ell)}\left(w_j^{(\ell)}, \hat{u}_j^{(\ell)*}\right)\right]^2, \tag{18}$$

where $w_j^{(\ell)*}$ are the adaptive weights that will be adapted in the second subproblems, and $\hat{u}_j^{(\ell)*}$ is the solution of the first subproblem.

From the uncoupled space-decomposition theorem (Theorem 2.3), the subproblem (18) can be further decomposed into a set of uncoupled subproblems $E_j^{(\ell)}(w_j^{(\ell)}, \hat{u}_j^{(\ell)*})$ that can be distributed among parallel processors. These uncoupled subproblems $E_j^{(\ell)}(w_j^{(\ell)}, \hat{u}_j^{(\ell)*})$ are the mean-squared error functions of a single Adaptive Linear Neuron (Adaline). They are defined as

$$E_j^{(\ell)}\left(w_j^{(\ell)}, \hat{u}_j^{(\ell)*}\right) \equiv \frac{1}{2}\sum_{p=1}^{P}\left[\hat{u}_{pj}^{(\ell)*} - u_{pj}^{(\ell)}\right]^2, \tag{19}$$

where $u_{pj}^{(\ell)} = [x_p^{(\ell-1)}]^T w_j^{(\ell)}$. The mean-squared error function (19) can be further expanded as

$$\begin{aligned} E_j^{(\ell)}\left(w_j^{(\ell)}, \hat{u}_j^{(\ell)*}\right) = {} & \frac{1}{2}\sum_{p=1}^{P}\left[\hat{u}_{pj}^{(\ell)*}\right]^2 - \sum_{p=1}^{P}\hat{u}_{pj}^{(\ell)*}\left[x_p^{(\ell-1)}\right]^T w_j^{(\ell)} \\ & + \frac{1}{2}\left[w_j^{(\ell)}\right]^T \sum_{p=1}^{P}\left[x_p^{(\ell-1)}\left[x_p^{(\ell-1)}\right]^T\right] w_j^{(\ell)}. \end{aligned} \tag{20}$$

If we define

$$b = \sum_{p=1}^{P}\hat{u}_{pj}^{(\ell)*}x_p^{(\ell-1)}, \tag{21}$$

and

$$Q = \sum_{p=1}^{P}\left[x_p^{(\ell-1)}\left[x_p^{(\ell-1)}\right]^T\right], \tag{22}$$

the Eq. (20) can thus be simplified to a standard quadratic function

$$E_j^{(\ell)}\left(w_j^{(\ell)}\right) = \frac{1}{2}\left[w_j^{(\ell)}\right]^T Q w_j^{(\ell)} - b^T w_j^{(\ell)} + c, \tag{23}$$

where $c = \frac{1}{2}\sum_{p=1}^{P}[\hat{u}_{pj}^{(\ell)*}]^2$ is a constant value in these uncoupled subproblems. The matrix Q is a symmetric and positive definite, or in rare case, positive semi-definite matrix [23]. That is, the second subproblems (18) and its further decomposed subproblems (19) are both convex problems. Therefore, the three PSDM algorithm presented in this paper can be applied to these convex subproblems.

7.2. Application of the parallel Madaline networks training

The mechanical part recognition is a typical application of Madaline networks. In this application, the Madaline networks are trained by the image patterns of mechanical parts. Then, the trained Madaline networks can be used to recognize the mechanical part on the manufacturing line. Training time is an important factor for such on-line applications because a long time of training will suspend the manufacturing line. The parallel training of the Madaline networks can thus be utilized to reduce the training time.

In this application, the Madaline networks are trained by twenty color image patterns. The architecture of the parallel processors for such parallel training is shown in figure 10. The first set of subproblems (17) are assigned to a shared memory computer with dual processors and the second set of subproblems (18) are assigned to all parallel processors. In this architecture, only the actual linear outputs u and desired linear output $\hat{u}$ are communicated among parallel processors. No communication is required among the second set of subproblems. Therefore, the communication overhead can be significantly reduced. Other architecture of parallel processors can also be applied to the parallel training. For example, both of the first subproblem (17) and the second subproblem (18) can be decomposed into subproblems that can be distributed to parallel processors with shared memory.

The numerical experiment results of variant processor numbers for the parallel training are listed on the Table 4, which shows that the efficiency of the parallel training varies from 69.65 to 98.13. The Speedups of the parallel training are shown in figure 11. The good performance of such architecture is due to the low communication overhead of the parallel processor architecture. Further research can focus on the improvement of the parallel minimization on the first set of subproblem (17), which can further improve the efficiency of the parallel training.

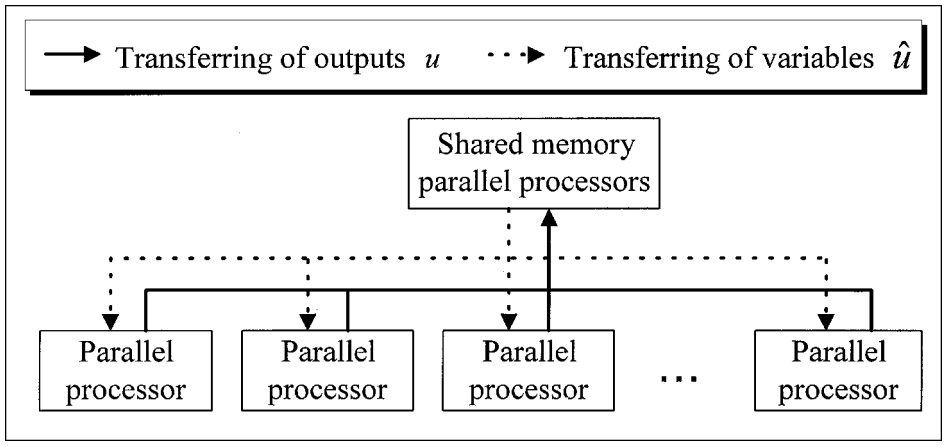

Figure 10. Communication of data for parallel training of Madaline networks.

Table 4. Numerical results for the PSDM algorithm application on the training of Madaline networks.

		24 hidden nodes		36 hidden nodes	
Input nodes	Processor numbers	Processor time (sec.)	Speedup/Efficiency (%)	Processor time (sec.)	Speedup/Efficiency (%)
64×64	1	14.01	—	22.70	—
	2	8.74	1.60/80.15	14.01	1.62/81.01
	4	4.61	3.04/75.98	8.05	2.82/70.50
	6	2.57	5.45/90.86	4.10	5.54/92.28
128×128	1	59.01	—	89.61	—
	2	31.92	1.85/92.44	49.85	1.80/89.88
	4	20.01	2.95/73.73	22.83	3.93/98.13
	6	14.12	4.18/69.65	16.81	5.33/88.85

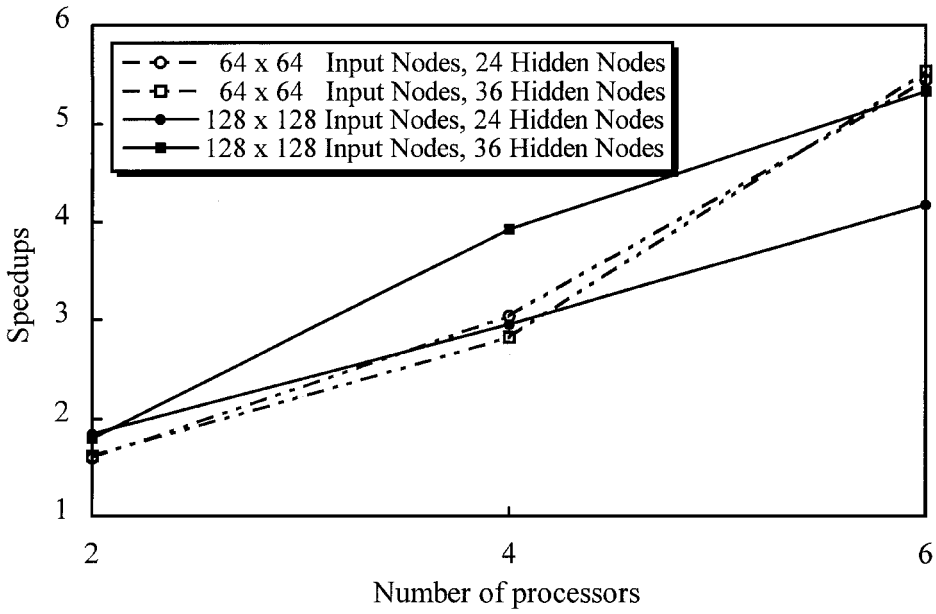

Figure 11. Speedups for parallel training of Madaline networks.

8. Conclusions

This paper discusses three parallel PSDM algorithms for solving the unconstrained minimization problem (1) among parallel processors. These algorithms enable minimization problem to be decomposed into subproblems that can be distributed among parallel processors. It is shown that if the decomposed subproblems are not coupled to each other, these subproblems can be solved among parallel processors independently without data

communication; otherwise, the parallel algorithms discussed in this paper can be used. Since only the coupled variables are communicated among processors, the data communication time among processors can be reduced. The PSDM algorithms can also be applied to the training of Madaline networks. A new parallel architecture of processors based on the PSDM algorithms is presented in this paper for the parallel training of Madaline networks. Numerical experiments show that such architecture can benefit well from parallel training.

Among the three parallel algorithms, the synchronous and sequential PSDM algorithms are particularly suitable for problem that can be decomposed into non-highly coupled subproblems. Numerical experiments show that these two PSDM algorithms can speed up the SDM algorithm that run on a single processor. However, due to the synchronization overhead, these two algorithms will be suspended if any processor or any network line slows or suspends for any reason. Such problems can be overcome by the asynchronous PSDM algorithm.

The asynchronous PSDM algorithm can benefit most from parallel computing among the algorithms presented in this paper. However, the convergence of the original sequential algorithm may be destroyed by the asynchronous algorithm. Therefore, the additional check condition (11) is required. It is shown in this paper that the asynchronous PSDM algorithm is particularly suited for the modern multitask (multithread) and event driven computer operation systems. Through the "Asynchronously Event Driven" mechanism, the coupled variables can be communicated among processors in the background tasks while the foreground tasks can be continuously computing without caring when the data are communicated to or from other processors.

The SDM algorithm can also be applied to multilayer discrete neural networks [15]. Since the activation functions of the multilayer discrete neural networks are non-differentiable hard-limiting function, the gradient descent methods can not be directly applied to train such neural networks. By the SDM algorithm, the multilayer discrete neural networks can be decomposed into a set of single layer neural networks, which can thus be trained by the perceptron learning rule that was developed for single layer discrete neural networks. Further research can focus on the application of the PSDM algorithms to the multilayer discrete neural networks.

Appendix: Test problems

Problem 1. Modified Powell Function [3, 12, 21]:

$$F = \sum_{i=4}^{n} [(x_{i-3} + 10x_{i-2})^2 + 5(x_{i-1} - x_i)^2 + (x_{i-2} - 2x_{i-1})^4 + 10(x_{i-3} - x_i)^4],$$

$$x^{(1)} = [3, -1, 0, 1, \ldots, 3, -1, 0, 1]^T.$$

Problem 2. TRIDIA Function [3]:

$$F = \sum_{i=2}^{n} [i\,(2x_i - x_{i-1})^2],$$

$$x^{(1)} = [3, -1, 0, 1, \ldots, 3, -1, 0, 1]^T.$$

Problem 3. Modified Wood Function [11, 12]:

$$F = \sum_{i=4}^{n} \{100(x_{i-3}^2 - x_{i-2})^2 + (x_{i-3} - 1)^2 + 90(x_{i-1}^2 - x_i)^2 + (1 - x_{i-1})^2 + 10.1[(x_{i-2} - 1)^2 + (x_i - 1)^2] + 19.8(x_{i-2} - 1)(x_i - 1)\},$$
$$x^{(1)} = [-3, -1, -3, -1, \ldots, -3, -1, -3, -1]^T.$$

Problem 4. Modified Zangwill Function [19]:

$$F = \sum_{i=3}^{n} [(x_{i-2} - x_{i-1} + x_i)^2 + (-x_{i-2} + x_{i-1} + x_i)^2 + (x_{i-2} + x_{i-1} - x_i)^2],$$
$$x^{(1)} = [100, -1.0, 2.5, \ldots, 100, -1.0, 2.5]^T.$$

Acknowledgments

The research reported in this paper was supported under a project sponsored by the National Science Council Grant, Taiwan, R.O.C., NSC-86-2212-E-009-030. We are grateful to the referees for their helpful comments and for the Prof. Fukushima for the kind assistance.

References

1. K.P. Bennett and O.L. Mangasarian, "Serial and parallel multicategory discrimination," SIAM Journal on Optimization, vol. 4, no. 4, pp. 722–734, 1994.
2. A. Bouaricha and J.J. Moré, "Impact of partial separability on large-scale optimization," Computational Optimization and Applications, vol. 7, no. 1, pp. 27–40, 1997.
3. A. Buckley and A. Lenir, "QN-link variable storage conjugate gradients," Mathematical Programming, vol. 27, pp. 155–175, 1983.
4. D. Conforti, L. Grandinetti, R. Musmanno, M. Cannataro, G. Spezzano, and D. Talia, "A model of efficient asynchronous parallel algorithms on multicomputer systems," Parallel Computing, vol. 18, pp. 31–45, 1992.
5. D. Conforti and R. Musmanno, "A parallel asynchronous Newton algorithm for unconstrained optimization," Journal of Optimization Theory and Applications, vol. 77, no. 2, pp. 305–322, 1993.
6. D. Conforti and R. Musmanno, "Convergence and numerical results for a parallel asynchronous quasi-Newton method," Journal of Optimization Theory and Applications, vol. 84, no. 2, pp. 293–310, 1995.
7. M.C. Ferris and O.L. Mangasarian, "Parallel variable distribution," SIAM Journal on Optimization, vol. 4, no. 4, pp. 815–832, 1994.
8. H. Fischer and K. Ritter, "An asynchronous parallel Newton method," Mathematical Programming, vol. 42, pp. 363–374, 1988.
9. M. Fukushima, M. Haddou, V.H. Nguyen, J.-J. Strodiot, T. Sugimoto, and E. Yamakawa, "A parallel descent algorithm for convex programming," Computational Optimization and Applications, vol. 5, pp. 5–37, 1996.
10. M. Fukushima, "Parallel variable transformation in unconstrained optimization," SIAM Journal on Optimization, vol. 8, pp. 658–672, 1998.
11. E.C. Housos and O. Wing, "Pseudo-conjugate directions for the solution of the nonlinear unconstrained optimization problem on a parallel computer," Journal of Optimization Theory and Applications, vol. 42, no. 2, pp. 169–180, 1984.
12. G.E. Johnson and M.A. Townsend, "A generalized direct search acceptable-point technique for use with descent-type multivariate algorithms," Journal of the Franklin Institute, vol. 309, pp. 163–177, 1980.

13. V.M. Kibardin, "Decomposition into functions in the minimization problem," Automation and Remote Control, vol. 40, part 1, pp. 1311–1323, 1980.
14. C.S. Liu and C.H. Tseng, "Space-decomposition minimization method for large-scale minimization problems," Computers & Mathematics with Applications, vol. 37, pp. 73–88, 1999.
15. C.S. Liu and C.H. Tseng, "A new training algorithm for multilayer discrete perceptrons," Neural, Parallel & Scientific Computations, vol. 6, pp. 217–228, 1998.
16. O.L. Mangasarian, "Parallel gradient distribution in unconstrained optimization," SIAM Journal on Control and Optimization, vol. 33, no. 6, pp. 1916–1925, 1995.
17. J.J. Moré, B.S. Garbow, and K.E. Hillstrom, "Testing unconstrained optimization software," ACM Transactions on Mathematical Software, vol. 7, no. 1, pp. 17–41, 1981.
18. K. Mouallif, V.H. Nguyen, and J.-J. Strodiot, "A perturbed parallel decomposition method for a class of nonsmooth convex minimization problems," SIAM Journal on Control and Optimization, vol. 29, no. 4, pp. 829–847, 1991.
19. M.V.C. Rao and K.S. Chandra, "A direct search package for unconstrained minimization," International Journal for Numerical Methods in Engineering, vol. 20, pp. 1643–1660, 1984.
20. R.B. Schnabel, "Concurrent function evaluations in local and global optimization," Computer Methods in Applied Mechanics and Engineering, vol. 64, pp. 537–552, 1987.
21. Y.A. Shpalenskii, "Iterative aggregation algorithm for unconstrained optimization problems,"Automation and Remote Control, vol. 42, part 1, pp. 76–82, 1981.
22. M.V. Solodov, "New inexact parallel variable distribution algorithms," Computational Optimization and Applications, vol. 7, pp. 165–182, 1997.
23. B. Widrow and M.A. Lehr, "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation," Proceedings of the IEEE, vol. 78, no. 9, pp. 1415–1442, 1990.