Reducing the Deteriorating Effect of Old History in Asynchronous Iterations*

Yasemin Yalçınkaya[1] and Trond Steihaug[2]

[1] Università degli Studi di Firenze, Dipartimento di Energetica "Sergio Stecco", Firenze, Italy
[2] University of Bergen, Department of Informatics, Bergen, Norway

Abstract. The deteriorating effect of old history in asynchronous iterations is investigated on an application based on the specialization of parallel variable distribution approach of Ferris and Mangasarian [4] to linear least squares problem. A partially asynchronous algorithm is developed which employs a combination of synchronization, a relaxation parameter and a certain form of overlap between subproblems. It is shown by numerical experiments that this combined effort to decrease the effect of old history is more effective than the single attempts considered in [9, 11].

The aim of the paper is to provide an efficient method to compensate for the deteriorating effect of old history on the convergence of asynchronous iterative methods. The issue of old history is investigated on an application based on the specialization of parallel variable distribution (PVD) approach of Ferris and Mangasarian [4] to linear least squares problem.

We have seen in [9, 11] that in the implementation of totally asynchronous iterations on linear least squares problems convergence is deteriorated due to the existence of *old history* in the system. Yet, some measures can be taken to reduce the effect of old history. It has been observed that increasing the number of processors increases the effect of old history, whereas introducing some form of synchronization, either in the form of local synchronization, or as barrier synchronization decreases the deterioration. The use of a relaxation parameter as a line search also improves the convergence rate.

Here we will formulate a partially asynchronous algorithm that combines most of the methods studied in [9, 11] plus a certain form of overlap to decrease/eliminate the effect of old history. This new algorithm is based on our observations in our former studies and the work of Dennis and Steihaug [2]. We will see that the new algorithm eliminates the deterioration in convergence rate observed due to the effect of old history.

In the following, we first give some notations and definitions that are used throughout the paper. In Sect. 2 we introduce the concepts of planar search and expanded blocks. The final algorithm is outlined and explained in Sect. 3. We give some experimental results in Sect. 4. Concluding remarks are included in Sect. 5.

* Work is a part of the first author's Dr. Scient. thesis completed at the University of Bergen, Department of Informatics, Bergen, Norway.

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 698–707, 2004.

1 Preliminaries

The specialization of PVD to linear least squares problem was presented in [2, 8] and a more general approach was presented in [6]. The same domain decomposition approach to linear least squares problem has been employed in [1, 9–11] but under a different name.

Let A be an $m \times n$ real matrix, $b \in \mathbb{R}^m$, and M be an $m \times m$ positive definite matrix. The weighted linear least squares problem is

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_M \ , \tag{1}$$

where $\|y\|_M^2 = y^T M y$.

To apply the PVD approach, the space $\mathbb{R}^n$ is decomposed as a Cartesian product of lower dimensional spaces $\mathbb{R}^{n_i}$, $i = 1, \ldots, g$, where $\sum_{i=1}^{g} n_i = n$. Accordingly, any vector $x \in \mathbb{R}^n$ is decomposed as $x = [x_1 \ \cdots \ x_g]^T$, and the matrix A is partitioned as $A = [A_1 \ \cdots \ A_g]$, where $A_i \in \mathbb{R}^{m \times n_i}$, and each A_i is assumed to have full rank. Then, the least squares problem (1) is equivalent to

$$\min_{x \in \mathbb{R}^n} \left\{ \| \sum_{i=1}^{g} A_i x_i - b \|_M : x_i \in \mathbb{R}^{n_i}, \ i = 1, \ldots, g \right\} \ . \tag{2}$$

In a parallel processing environment, consistent with the application of the PVD approach we assume that there are g processors each of which can update one block component, x_i. These processors that solve the subproblems are called as *slave* processors and are coordinated by a *master* processor which computes the global solution vector.

Let $t \in \mathcal{T} = \{1, 2, \ldots\}$ be a normalized integer counter of actual times that one or more components x_i of x are updated, and let $x(t)$ be the approximation to x^*, the solution of (1), at a given time instant t. Further, let $\mathcal{T}^i \subset \mathcal{T}$ be the set of times component i is updated and $x_i(t)$ be the value of component x_i at time t. For a given t, each slave i for which $t \in \mathcal{T}^i$, is assigned the subproblem

$$\text{Solve for } \ d_i(t) \in \mathbb{R}^{n_i} : \min_{d_i \in \mathbb{R}^{n_i}} \|A_i d_i + r(t-1)\| \ , \tag{3}$$

where $d_i(t) = x_i(t) - x_i(t-1)$ is the *direction* vector and $r(t) = \sum_{i=1}^{g} A_i x_i(t) - b = r(t-1) + \sum_{i=1}^{g} A_i d_i(t)$ is the *residual*. The master computes the global solution

$$x(t) = x(t-1) + \sum_{i=1}^{g} \bar{d}_i(t), \tag{4}$$

where $\bar{d}_i \in \mathbb{R}^n$ is the vector d_i complemented with zeros in places of blocks $j \in \{1, \ldots, g\}$, $j \neq i$. Note that $\bar{d}_i(t)$ is nonzero only for the components i for which $t \in \mathcal{T}^i$.

If for all $t \in \mathcal{T}$ we have $t \in \mathcal{T}^i$, $i = 1, \ldots, g$, then we get a synchronous iteration in the form of Jacobi method on normal equations $A^T A x = A^T b$ [10],

where the values of all the components $x_i(t)$ are computed using the values from the previous iteration, namely $x_j(t-1),\ j = 1, \ldots, g$. However, in an asynchronous setting, the processor computing d_i may not have access to the most recent values of other components of x, and $d_i(t)$ is computed using a residual $r = \sum_{j=1}^{g} A_j x_j(\tau_j^i(t)) - b$, where $\tau_j^i(t)$ are *time-stamps* satisfying

$$0 \leq \tau_j^i(t) < t, \quad \forall\ t \in \mathcal{T}\ . \tag{5}$$

In (5) the difference $t - \tau_j^i(t)$ can be seen as a communication delay. This communication delay, or time-lag, causes that old data is used in the computation of a component x_i. Consequently, these updates computed using older values deteriorate the convergence property of the algorithm. We call these older values in the system as *old history* and their deteriorating effect as the *effect of old history*.

When the coefficient matrix A is partitioned, the blocks A_i are formed by non-overlapping consecutive columns. Nevertheless, in the implementation of (3) the effect of row dependence between the blocks comes into picture. The dependence between these subproblems can be represented by a directed graph called *dependency graph*, and for this particular problem the dependency graph is an adjacency graph since we have a symmetric system [10]. For the PVD formulation of the linear least squares problem as given above, the set of *essential neighbors* for a given node i in the dependency graph is defined as $E_i = \{j \mid$ block i and block j have nonzero elements on the same row positions$\}$.

2 From Jacobi to a Partially Asynchronous Algorithm

We first introduce *planar search* as an alternative to line search for finding a relaxation parameter. Then we show a way of making all the blocks essential neighbors of each other, which eliminates the need for waiting for an update from one's essential neighbors during the computations to avoid zero updates.

2.1 Planar Search

In (4), after each direction vector d_i is computed, a line search can be done to compute a relaxation parameter ω_i that will give the most decrease in the residual vector r for this direction. The relaxation parameter ω_i given in [11] is computed as the solution of the one dimensional least squares problem

$$\min_{\omega_i \in \mathbb{R}} \|(A_i d_i(t))\omega_i + r(t-1)\|_M\ , \tag{6}$$

and only the direction vectors that give ω_i values that are in the interval $(0, 2)$ are used to update the residual vector. However, instead of doing a line search to correct each of these d_i vectors one by one, we can wait for receiving all the direction vectors $d_i,\ i = 1, \ldots, g$, and introduce a planar search where the values of the relaxation parameters $\omega_i,\ i = 1, \ldots, g$, are computed as the solution of a new least squares problem.

Let $\widehat{A} = \left[A\bar{d}_1 \ \cdots \ A\bar{d}_g\right]$, where $\widehat{A} \in \mathbb{R}^{m\times g}$. Then the planar search step is:

$$\text{Solve for } \ s(t) \in \mathbb{R}^g : \ \min_{s\in\mathbb{R}^g} \|\widehat{A}(t)s + r(t-1)\|_M \ . \tag{7}$$

The new iterate is

$$x(t) = x(t-1) + \sum_{i=1}^{g} s_i(t)\bar{d}_i(t) \ .$$

Planar search as defined in (7) is an extension of (6) where the restriction on the values of ω_i is eliminated. Obviously, the planar search step implemented in this fashion introduces a blocking synchronization point in the parallel implementation whereas line search is done without synchronization.

2.2 Expanded Blocks

We pointed out in [11] the need for waiting for an update from any one of the essential neighbors of a given block i between two consecutive updates from this block in order to avoid redundant zero computations. If the subproblems are formed such that each block is an essential neighbor of all the other blocks, the computations can go on continuously with no delay. One method to obtain blocks that are all essential neighbors of each other is to take the original blocks $i, \ i = 1, \ldots, g$, as they are and expand these blocks by $g-1$ new columns, one for each block $j, \ j \neq i, \ j \in \{1, \ldots, g\}$. Each of these new columns is the one dimensional aggregate of subspace $j, \ j \neq i, \ j \in \{1, \ldots, g\}$.

Let $p \in \mathbb{R}^n$ be the *weight vector* partitioned as $p = [p_1 \ \cdots \ p_g]^T$, where $p_i \in \mathbb{R}^{n_i}, \ i = 1, \ldots, g$. Let $\bar{p}_i \in \mathbb{R}^n$ be the vector obtained by starting with a zero vector and placing the non-zero entries of p_i in the positions corresponding to the column indices in A of A_i. Define the $n \times (g - 1 + n_i)$ matrix P_i:

$$P_i = \left[\bar{p}_1 \ \cdots \ \bar{p}_{i-1} \ \bar{I}_i \ \ \bar{p}_{i+1} \ \cdots \ \bar{p}_g\right] \ , \tag{8}$$

where $I_i \in \mathbb{R}^{n_i\times n_i}$ is the identity matrix and $\bar{I}_i$ is the $n \times n_i$ matrix formed from columns of the $n \times n$ identity matrix so that $A\bar{I}_i = A_i$.

For $\widetilde{n}_i = n_i + g - 1$, define the $m \times \widetilde{n}_i$ matrix

$$\widetilde{A}_i = AP_i = [A\bar{p}_1 \ \cdots \ A\bar{p}_{i-1} \ A_i \ A\bar{p}_{i+1} \ \cdots \ A\bar{p}_g] \ . \tag{9}$$

The expanded subspace is now the range space of $\widetilde{A}_i$ and $A\bar{p}_j = A_j p_j$ is the one dimensional aggregate of A_j.

Unless the weight vector p is very sparse, all the newly formed expanded blocks $\widetilde{A}_i, \ i = 1, \ldots, g$, will be essential neighbors of each other.

For a given $p(t) \in \mathbb{R}^n$ at time t the subproblem (3) is replaced by

$$\text{Solve for } \ \widetilde{d}_i(t) \in \mathbb{R}^{\widetilde{n}_i} : \ \min_{\widetilde{d}_i\in\mathbb{R}^{\widetilde{n}_i}} \|\widetilde{A}_i(t)\widetilde{d}_i + r(t-1)\|_M \ , \tag{10}$$

where $\widetilde{A}_i(t)$ is defined in (9) for the given vector $p(t)$.

The step $c \in \mathbb{R}^n$ is

$$c = \sum_{i=1}^{g} P_i \widetilde{d}_i \ . \tag{11}$$

For a given $c \in \mathbb{R}^n$, define the $n \times g$ matrix

$$C = [\bar{c}_1 \ \cdots \ \bar{c}_g] \ . \tag{12}$$

Consider the $m \times g$ matrix

$$\widehat{A} = AC = [A\bar{c}_1 \ \cdots \ A\bar{c}_g] \ . \tag{13}$$

Then the planar search step (7) is executed replacing the matrix $\widehat{A}(t)$ with the $\widehat{A}$ of (13). We use the vector $c(t)$ defined in (11), and the new iterate is

$$x(t) = x(t-1) + C(t)s(t),$$

where $C(t)$ is defined in (12).

One question is how to choose the weight vector p. When the only aim is to form new blocks that are all essential neighbors of each other any choice of a nonzero p that is not very sparse is acceptable. Yet, while forming the aggregations of each block we can also try to increase the convergence rate of the new algorithm. If we could choose $p(t) = x^* - x(t) \equiv e(t)$, then each $P_i(t)\widetilde{d}_i(t)$ would be $e(t)$ and we would get convergence in one step [2]. Since $e(t)$ is not known, a good approximate is taking $p(t) = x(t) - x(t-1) = C(t)s(t)$. This choice leads to a significant reduction in the number of iterations in sequential and synchronous implementations of this approach [2]. Other choices for p are constant values, e.g. $p = [1 \ \cdots 1]^T$, or a *predictor/corrector* scheme defined by keeping p fixed for several predictor iterations without having to redo any factorizations [2]. Note that when p is a zero vector, we get the block Jacobi iteration on normal equations.

3 A Partially Asynchronous Algorithm

The two issues introduced in the former section, if applied directly on the block Jacobi iteration on normal equations, will give a synchronous parallel algorithm since planar search brings out the need for a blocking synchronization step. We know that too frequent blocking synchronization points in parallel implementations should be avoided due to the performance penalties imposed on the system [10]. Hence, we decide to do a planar search after every l sweeps, i.e., after l updates received from each block i, $i = 1, \ldots, g$. Between two planar search steps the weight vector p and consequently $\widetilde{A}_i$ matrices are kept constant. The l updates received between two synchronization points from each slave are accumulated and matrix $\widehat{A}$ is formed using these accumulated values. The accumulation of updates in this manner is an application of *reliable updating technique* which leads to very accurate approximations provided that the accumulations do not become too large [3].

Between two synchronization points a running copy of the residual is kept, which is updated on receiving a new result from any one of the slaves and its updated value is sent back to the slave who has communicated the last result. Hence, asynchronism is introduced in the system, which adds the flavor of Gauss–Seidel in the iterations. The original residual vector is used in the planar search step and is updated thereafter.

Observe that in (10), $\widetilde{d}_i = [\delta_1^i \ \cdots \ \delta_{i-1}^i \ d_i \ \delta_{i+1}^i \ \cdots \ \delta_g^i]^T$, where $d_i \in \mathbb{R}^{n_i}$ is the variable representing the space spanned by the ith block of variables in x_i, and the scalar values δ_j^i are the aggregate variables expanding this space. Expanding the subproblems with these aggregate variables not only makes all the new blocks essential neighbors of each other, but also creates a sort of "overlap", though the overlapping part of the newly formed blocks is in aggregated form. In asynchronous implementations overlap to some degree is found to accelerate the overall iteration [7]. However, the benefit of overlap is shown to be in the inclusion of extra variables in the minimization for the local variables only. The updated value on the overlapped portion of the domain should not be utilized [5]. In our case, the inclusion of the scalar values δ_j^i in the update of the residual vector will increase the effect of old history in the system. Therefore, we form and solve the expanded subproblems, but throw away the calculated δ_j^i values. The c vector in (11) is replaced by $c = \sum_{i=1}^{g} \bar{d}_i$, where d_i are extracted from $\widetilde{d}_i$ of (10). Since we use accumulated updates in $\widehat{A}$, the d_i used in forming c are, in fact, $d_i = \sum_{j=1}^{l} d_i^j$.

Algorithm 1 (PALSQ).

if *master*
  Partition A into g blocks and **Send** one to each *slave*.
  Initialize $x(0),\ r(0),\ v^0,\ c(1),\ \widehat{A}(1),\ \bar{A}(1)$.
  Broadcast $v^0,\ \bar{A}(1)$.
else $\{$*slave* $i,\ i = 1, \ldots, g\}$
  Receive A_i.
  Receive $v^0,\ \bar{A}(1)$.
  Form and factorize $\widetilde{A}_i(1)$.
$t = 1$.
while not *converged* **do**
  if *slave* $i,\ i = 1, \ldots, g$
    for $j = 0, \ldots, l-1$
      Solve for $\widetilde{d}_i^j :\ \min \|\widetilde{A}_i(t)\widetilde{d}_i + v^j\|_M$.
      Compute $A_i d_i$.
      Send $A_i d_i,\ d_i$.
      if $j \neq l-1$ **Receive** v^{j+1}.
    Receive $v^0,\ \bar{A}(t+1)$.
    $t = t + 1$.
    Form and factorize $\widetilde{A}_i(t)$.
  else $\{$*master*$\}$

for $j = 0, \ldots, l-1$
  for $i = 1, \ldots, g$
    Receive $A_i d_i$, d_i.
    $v^{j+i/g} = v^{j+(i-1)/g} + A_i d_i$.
    Send $v^{j+i/g}$.
    $\widehat{A}_i(t) = \widehat{A}_i(t) + A_i d_i$.
    $c(t) = c(t) + \bar{d}_i$.
Solve for $s(t) : \min \|\widehat{A}(t)s + r(t-1)\|_M$.
$x(t) = x(t-1) + C(t)s(t)$.
$r(t) = r(t-1) + \widehat{A}(t)s(t)$.
Compute convergence criteria.
if not *converged*
  $v^0 = r(t)$.
  $\bar{A}(t+1) = \widehat{A}(t)s(t)$.
  Broadcast v^0, $\bar{A}(t+1)$.
  $t = t + 1$.
  Initialize $\widehat{A}(t), c(t)$.
else
  break.

In Algorithm 1, initially the weight vector is $p = [1 \ \cdots \ 1]^T$. The matrix $\bar{A}$ is defined as $\bar{A}(t+1) = [A_1 p_1(t) \ \cdots \ A_g p_g(t)]$, and is utilized in forming the new $\widetilde{A}_i$ matrices after each planar search step. Notice that in the algorithm $\bar{A}(t+1)$ is set to $\widehat{A}(t)s(t)$. We know from (13) that $\widehat{A}(t) = AC(t)$. Then

$$\widehat{A}(t)s(t) = AC(t)s(t) = A(C(t)s(t)) = Ap(t) = \bar{A}(t+1) \ .$$

Thus, there is no explicit update of the weight vector $p(t)$, since its new value is packed in the matrix $\bar{A}(t+1)$. Obviously, if $p(t) = p$ is constant throughout the algorithm, the assignment and the broadcast of $\bar{A}$ is done only once at the start of the implementation, so as the forming and factorization of $\widetilde{A}_i, \ i = 1, \ldots, g$.

4 Numerical Experiments

The experiments are done using the "time-lagged" analytical model of [9]. The example test problem used in the graphs is problem ASH958 from the Harwell-Boeing sparse matrix test collection. The number of blocks and slaves g is 15. In the graphs the markers on the continuous lines give the value of the residual vector after each planar search. The dotted lines around the continuous lines illustrate the temporary residual vector v. The curve of continuous line with no markers depicts the behavior of a totally asynchronous implementation with no relaxation parameter and synchronization of any form, where g slaves operate on the original blocks $A_i, \ i = 1, \ldots, g$, receiving the latest available residual vector on the master whenever they send a new update.

We first check the effect of planar search on old history. In Fig. 1 totally asynchronous implementation is depicted against a partially asynchronous implementation where the blocks are the same as in the totally asynchronous one, but a planar search is done before updating the residual after l sweeps. The residual decreases monotonically for all choices of l although we observe a difference in the number of planar searches and total updates on the temporary residual vector v. Going from a synchronous implementation ($l = 1$) with planar search to a partially asynchronous implementation with a synchronization step after $l = 3$ sweeps, the number of synchronization points before convergence decreases from 21 to 9 (-57.1%), while the number of total updates computed increases from 315 to 405 (+28.6%). Considering that the updates are computed in parallel whereas during a synchronization step all the slaves remain idle, the case of $l = 3$ makes more efficient use of the available resources.

The results depicted in Fig. 1 compared to Fig. 5 of [9], where the effect of synchronization on old history is studied, point out that a synchronization step combined with planar search is more effective than a simple barrier synchronization in decreasing the magnitude of time-lag in the system.

In the second experiment we introduce expanded blocks formed using a constant weight vector $p = [1 \ \cdots \ 1]^T$. Here, $l = 1$ gives a Jacobi implementation with overlap and planar search. Figure 2 demonstrates that for all values of l the residual is monotonically decreasing. The number of planar searches before convergence decreases from 17 for $l = 1$ to 8 for $l = 3$ (-52.9%), whereas the total number of updates on the temporary residual vector increases from 255 to 360 (+41.2%). When we compare the results of this experiment with the former one, we see the positive effect of overlap on the convergence rate.

Lastly, we implement Algorithm 1 as it is given in the previous section, i.e., partially asynchronous implementation with expanded blocks and $p(t) = C(t)s(t)$. Here, different than the former two experiments, we observe that the cases of $l = 2$ and $l = 3$ have a better converge rate than the case of $l = 1$. Again, for all values of l we get a monotonically decreasing residual vector which is not contaminated by old history. Comparing the case of $l = 1$ with Dennis and Steihaug's Jacobi-Ferris-Mangasarian algorithm [2] with the "forget-me-not" variables of Ferris and Mangasarian [4] contributing to the planar search, we observe that inclusion of the δ_j^i values is necessary in a synchronous implementation, but in an asynchronous setting it is crucial and beneficial to discard them, since their contribution increases the effect of old history in the system if they are not discarded.

In this last experiment, the number of planar searches necessary before convergence decreases from 21 for $l = 1$ to 7 for $l = 3$, whereas the total number of updates on the temporary residual vector v remains constant at 315 for both $l = 1$ and $l = 3$. This means that the number of synchronization points is decreased by 66.7% while the same number of updates are computed asynchronously ($l = 3$) instead of following a synchronous pattern ($l = 1$).

All these experiments indicate that some degree of synchronization introduced in an asynchronous system combined with a form of overlap and a relax-

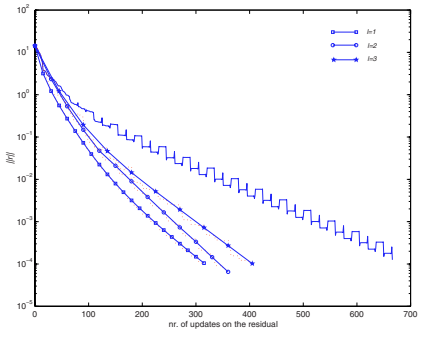

Fig. 1. Partially asynchronous implementation with planar search versus totally asynchronous implementation.

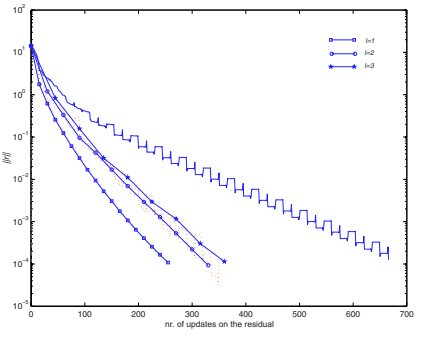

Fig. 2. Partially asynchronous implementation with planar search and expanded blocks in which $p = [1 \ \cdots \ 1]^T$ versus totally asynchronous implementation.

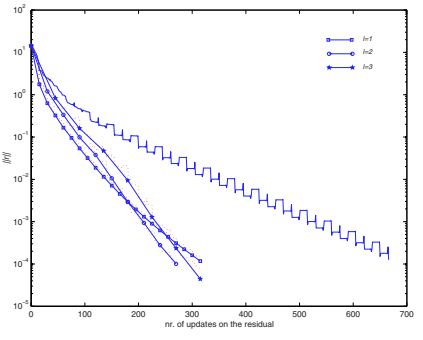

Fig. 3. Partially asynchronous implementation with planar search and expanded blocks in which $p(t) = C(t)s(t)$ versus totally asynchronous implementation.

ation parameter eliminate the deteriorating effect of old history. This approach of combining different issues that affect old history in the system is more effective than the single attempts carried out in [9, 11].

5 Concluding Remarks

Algorithm PALSQ is, in fact, a *predictor-corrector algorithm*, since in the asynchronous phase we accumulate updates (prediction) and only in the synchronization step the residual is updated (correction) with a combination of these accumulated updates and a relaxation parameter. During the asynchronous phase only the temporary residual vector v is updated. Therefore, the residual vector is concealed from the effect of old history.

We eliminate the effect of old history in a given asynchronous system introducing blocking synchronization, a relaxation parameter computed in a planar

search, and overlap between subproblems in form of aggregated columns expanding the subproblem spaces, all applied together in Algorithm PALSQ. Slightly increasing the size of the subproblems for the sake of increasing dependency between slave processors and benefiting from the synchronization point for computing a relaxation factor turns out to be more effective than introducing only simple synchronization points or decreasing the number of processors in the system as means of decreasing the effect of old history.

We have considered only two values for the weight vector p, which is used in aggregating the blocks and forming overlap between the subproblems. Both the experiment with p having a constant value and the experiment with a variable $p(t)$ vector indicate that overlap is an effective factor in decreasing the deterioration caused by old history. Therefore, it is important to investigate further what other values can be assigned to the weight vector.

References

1. Dennis, Jr., J. E., Steihaug, T.: On the successive projections approach to least squares problems. SIAM J. Numer. Anal. **23** (1986) 717–733
2. Dennis, Jr., J. E., Steihaug, T.: A Ferris-Mangasarian technique applied to linear least squares problems. Tech. Rep. No. 150, Department of Informatics, University of Bergen, Norway (1998)
3. Duff, I. S., Van der Vorst, H. A.: Developments and trends in the parallel solution of linear systems. Parallel Comput. **25** (1999) 1931–1970
4. Ferris, M. C., Mangasarian, O. L.: Parallel variable distribution. SIAM J. Optim. **4** (1994) 815–832
5. Frommer, A., Pohl, B.: A comparison result for multisplittings and waveform relaxation methods. Numer. Linear Algebra Appl. **2** (1995) 335–346
6. Frommer, A., Renaut, R. A.: A unified approach to parallel space decomposition methods. J. Comput. Appl. Math. **110** (1999) 205–223
7. Frommer, A., Szyld, D. B.: On asynchronous iterations. J. Comput. Appl. Math. **123** (2000) 201–216
8. Renaut, R. A.: A parallel multisplitting solution of the least squares problem. Numer. Linear Algebra Appl. **5** (1998) 11–31
9. Steihaug, T., Yalçınkaya, Y.: Deteriorating convergence for asynchronous methods on linear least squares problem. In C. Lengauer, M. Griebl, S. Gorlatch (Eds.): Euro-Par'97 Parallel Processing, LNCS 1300, Springer Verlag (1997) 750–759
10. Yalçınkaya, Y.: Reducing the Effect of Old History in Asynchronous Iterations: An Empirical Study. Dr. Scient. Thesis, Department of Informatics, University of Bergen, Norway (2003)
11. Yalçınkaya, Y., Steihaug, T.: Asynchronous methods and least squares: An example of deteriorating convergence. In A. Sydow (Ed.): Proc. of the 15th IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics, Vol. 1 Computational Mathematics, Wissenschaft & Technik Verlag (1997) 535–540