Asynchronous Methods and Least Squares: An Example of Deteriorating Convergence

Y. Yalçınkaya, T. Steihaug
University of Bergen, Department of Informatics,
Høyteknologisenteret, N-5020 Bergen, Norway
e-mail: yasemin@ii.uib.no

Keywords: Parallel algorithms, large overdetermined linear systems, asynchronism

ABSTRACT

We use a block iterative method for solving linear least squares problems. The subproblems are solved asynchronously on a distributed memory multiprocessor. It is observed that increased number of processors results in deteriorating convergence. We illustrate the deteriorating convergence by some numerical experiments. The deterioration of the convergence can be explained by contamination of the residual by *old information*. Our purpose is to reduce the effect of old information. The issues investigated here are the effect of the number of processors, the role of *essential neighbors* [7] and heterogeneous processors. We include two heuristics to identify the information to be discarded and reduce the effect of old information: a relaxation factor and synchronization. The characterization of old information remains as an open problem.

INTRODUCTION

In this paper we use a block iterative method for solving sparse linear least squares problems. A general framework for this method is introduced by Dennis and Steihaug [4], and their preliminary tests indicate that this leads quickly to cheap solutions of limited accuracy.

Baudet's [1] experimental results on systems of linear equations show a considerable advantage for iterative methods on parallel computers with no synchronization at all. His statement has led us to experiment with *totally asynchronous* [2] block iterative methods for the solution of linear least squares problems. We partition the columns of the coefficient matrix into disjoint blocks of columns and then project the updated residual into each column subspace. This algorithm which is called *column oriented successive subspace correction* (CSSC) in [6], is, in fact, Gauss-Seidel iteration on the normal equations. Each subproblem is substantially smaller than the original problem and hence is solved directly using QR factorization and semi-normal equations (SNE) [3] on one processor. Each processor computes a correction of the solution vector restricted to the variables associated with the blocks of columns. The computation requires the residual which is the global data. The processors use the residual available at the start of their computations without waiting for the newest data. This way, the disadvantages resulting from the execution of synchronization primitives are avoided.

We do not address the issues of timing and speedup in this paper. Some timing and speedup results can be found in [8]. In the asynchronous implementation it is observed that increased number of processors results in contaminated residual and hence deteriorating convergence. This is due to the existence of *old information* in the system. We intend to reduce the effect of old information, thus decreasing the deteriorations in convergence.

THE LINEAR LEAST SQUARES PROBLEM

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

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

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

We divide the columns of A into g blocks $A_1, A_2, \ldots, A_g$, where $A_i \in \mathbb{R}^{m\times n_i}$. We assume, without loss of generality, that each A_i has full rank. The least squares problem (1) is then equivalent with

$$\min\{\|A_1x_1 + A_2x_2 + \cdots + A_gx_g - b\|_M : x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}, \ldots, x_g \in \mathbb{R}^{n_g}\}\ . \tag{2}$$

Suppose that we have an approximation x^k to a solution x^* to (1) and we divide x^k into $x_1^k, x_2^k, \ldots, x_g^k$ as above. Then from (2) we can write the following successive replacements iteration:

for $i = 1, 2, \ldots, g$ **do**
  Solve for $x_i^{k+1} \in \mathbb{R}^{n_i}$:
    $\min \| \sum_{j=1}^{i-1} A_jx_j^{k+1} + A_ix_i^{k+1} + \sum_{j=i+1}^{g} A_jx_j^k - b\|_M$.

Consider:

$$\begin{aligned}\textstyle\sum_{j=1}^{i-1} A_jx_j^{k+1} + A_ix_i^{k+1} &+ \textstyle\sum_{j=i+1}^{g} A_jx_j^k - b\\ &= A_i(x_i^{k+1} - x_i^k) + \textstyle\sum_{j=1}^{i-1} A_jx_j^{k+1} + \sum_{j=i}^{g} A_jx_j^k - b\\ &= A_is_i^k + r^{k+(i-1)/g},\end{aligned}$$

where $s_i^k = x_i^{k+1} - x_i^k$ is the step or correction, and

$$r^{k+(i-1)/g} = \sum_{j=1}^{i-1} A_jx_j^{k+1} + \sum_{j=i}^{g} A_jx_j^k - b \tag{3}$$

is the residual. The least squares subproblem, now, is:

Solve for $s_i^k \in \mathbb{R}^{n_i} : \min\{\|A_is_i + r^{k+(i-1)/g}\|_M\}$.

The residual can be shown to satisfy $r^{k+i/g} = r^{k+(i-1)/g} + A_is_i^k$. The new approximate solution is $x_i^{k+1} = x_i^k + s_i^k,\ \ i = 1, \ldots, g$. For $s_i \in \mathbb{R}^{n_i}$, introduce the vector $\bar{s}_i \in \mathbb{R}^n$,which is obtained by starting with a zero vector and placing the nonzero entries of s_i in the right positions. Then $x^{k+1} = x^k + \Sigma_{i=1}^{g}\bar{s}_i^k$. Introducing $\hat{r}$ and a relaxation parameter, we can now define the block successive over-relaxation (SOR) algorithm for the solution of (1).

Algorithm 1

Subdivide A into g blocks.
Choose $0 < \omega_i < 2, i = 1, 2, \ldots, g$.
Choose $x_i^0, i = 1, \ldots, g$.
Compute $r^0 = Ax^0 - b$.
for $k = 0$ **step** 1 **until** *convergence* **do**
  for $i = 1, 2, \ldots, g$ **do**
    $\hat{r} = r^{k+(i-1)/g}$.
    Solve for s_i^k : $\min\{\|A_is_i^k + \hat{r}\|_M\}$.   (4)
    $r^{k+i/g} = r^{k+(i-1)/g} + \omega_iA_is_i^k$.
    $x^{k+i/g} = x^{k+(i-1)/g} + \omega_i\bar{s}_i^k$.
  Check for *convergence*.

The series of approximations $\{x^k\}$ we get from Algorithm 1 converge to x^*, a solution of the least squares problem (1), and $\|r^k\|_M$ is strictly monotonically decreasing [4].

PARALLELIZATION

Jacobi type of iterations are straightforward to implement in parallel. In Algorithm 1, if use $\hat{r} = r^k$, we get Jacobi method. The main computation in the inner loop is now the solution of (4). This system can be solved concurrently for each block i on multiple processors provided that the submatrices A_i are

available on the processors. The processors, after computing their corrections on block components of x have to synchronize at the end of the current iteration before starting with the next iteration. Now that we are able to compute the corrections from each block in one step, we have gained a considerable advantage over the sequential algorithm. But we can do better. In synchronized algorithms, the faster processors waiting for the slower ones to complete their computations before they can enter the critical section causes an overhead. To get higher utilization of the available CPU power we can remove synchronization and the restriction on the order of the updates. By removing synchronization from the synchronous Jacobi algorithm and letting $\hat{r}$ get the latest available value of the residual in the system we obtain a totally asynchronous algorithm.
In asynchronous algorithms processors can execute more iterations when they are not constrained to wait for the results of the computation on other processors. In the computation of an iterate, nothing is imposed on the use of the values of the previous iterates. The only thing that is required is that, eventually, the values of an early iterate cannot be used any more in further evaluations. This condition is met as long as no processor falls out of the system. However, removing restrictions on the former iterates used in computations brings out the danger that the iterations are performed on the basis of outdated (old) information.
An important disadvantage of asynchronism is that it can destroy convergence properties that the algorithm may possess when executed synchronously or sequentially. In some cases, it is necessary to place limitations on the size of communication delays to guarantee convergence. In all cases, the analysis of asynchronous algorithms is considerably more difficult than for their synchronous counterparts. Necessary conditions for the convergence of linear problems is given by Bertsekas and Tsitsiklis in [2]. They also provide a rate of convergence analysis and a comparison between synchronous and asynchronous algorithms.

The Totally Asynchronous Algorithmic Model

Let $\mathcal{T} = \{1, 2, \ldots\}$ be a set of times at which one block x_i of x is updated by some processor and $\mathcal{T}^i$ = set of times at which x_i is updated.
The processor computing s_i may not have access to the most recent values of x_j in (3). For $t \in \mathcal{T}^i$, $s_i(t)$ is computed using a residual $\hat{r} = \sum_{j=1}^{g} A_j x_j(\tau_j^i) - b$, where $0 \leq \tau_j^i(t) \leq t-1$. At all times $t \notin \mathcal{T}^i$, x_i is left unchanged:

$$x_i(t) = x_i(t-1), \; t \notin \mathcal{T}^i, \text{ or } \; x(t) = x(t-1) + \bar{s}_i(t) \; .$$

Asynchronous Implementation

We will now give the main algorithm for the asynchronous implementation of Algorithm 1 on an MPMD machine with $p = g + 1$ processors. In the algorithm, processor p_0 is used as the master and processors p_i, $i = 1, 2, \ldots, g$ act as slaves. **Send** and **Receive** are communications with the master. **Broadcast** is done by the master processor.

Algorithm 2

Initialization:
  As in Algorithm 1.
  Initiate each processor $i = 1, \ldots, g$:
    Receive(A_i, p_0).
    Receive$(x_i(0), p_0)$.
    Receive(ω_i, p_0).
    Preprocess. *{*Compute *QR* factorization and store *R }*
  Broadcast$(r(0))$.
Loop :

Let t be a global counter of corrections, and let t_1^i and t_2^i be two consecutive elements in $\mathcal{T}^i$.

while not *termination* **do**

  if *slave* **then**

    Solve for s_i :

      $\min\{\|A_i s_i + r(t_1^i)\|_M : s_i \in \mathbb{R}^{n_i}\}$.

    Update $x_i : x_i = x_i + \omega_i s_i$.

    Send$(A_i s_i, p_0)$.

    Receive$(r(t_2^i), p_0)$.

  else *{*master*}*

    Receive$(A_i s_i, p_i)$. *{*s_i computed using r at t_1^i*}*

    $r(t+1) = r(t) + \omega_i A_i s_i$.

    Check for *termination.*

    if not *termination* **then**

      Send$(r(t), p_i)$. *{*$t_2^i = t$*}*

EXPERIMENTS

In this section we will present the results of some numerical experiments. The first two experiments illustrate the deteriorating convergence when one passes from sequential to parallel implementation, and from few processors to many. The next set of four experiments aim at decreasing the contamination of the residual.

Test problems used in the experiments are taken from Harwell-Boeing sparse matrix test collection. Groups are formed by taking blocks of consecutive columns. If not otherwise stated g is 30. Each subproblem is solved using QR factorization and SNE. Both parallel and sequential implementations are done on Intel Paragon. Static assignment of blocks to processors is chosen to avoid the overhead of assignments during the computation phase. The number of processors p reported is the number of slave processors.

Figure 1 depicts the main result of [8]. Here, a totally asynchronous implementation on $p = g$ processors is compared to sequential Gauss-Seidel and Jacobi methods. The figure shows the deterioration of convergence when one moves from sequential Gauss-Seidel and Jacobi to totally asynchronous implementation.

In the second experiment we implement totally asynchronous iterations on different number of processors p, $p \leq g$ and $g \bmod p = 0$. We see in Figure 2 that for some small number of processors the rate of convergence lies in the neighborhood of sequential Gauss-Seidel method. If we increase the number of processors further to make better use of the available CPU power, the rate of convergence is degraded. Increased number of processors result in increased deterioration.

The above experiments are done on homogeneous processors where the computing speed of each processor is nearly the same. When we use heterogeneous processors with computing speeds varying between approximately 1-7 the deterioration in the residual is damped (Figure 3). However, the improvement in the convergence rate is not significant.

We need to consider the effect of dependence between blocks on the convergence rate in the asynchronous implementation. Let $E_i = \{j \mid$ block i and block j have nonzero elements on the same row positions$\}$ be called the *essential neighbors* [7] of block i. Let $t_1, t_2 \in \mathcal{T}^i$ be consecutive times of update from block i. When we compute

$$\min\{\|A_i s_i + r(t_1))\|_M\}, \;\; r(t_2) = r(t_2 - 1) + \omega_i A_i s_i$$

at time t_2, $A_i^T r(t_2) = 0$ unless any block $j, j \in E_i$ has sent an update between t_1 and t_2. In the next experiment, to avoid zero corrections, block i is forced to wait until an update from *any* block $j, j \in E_i$ arrives at the master. We meet a similar enforcement in [6]. However, there the processors wait for updates from *all* their essential neighbors before they go on with the next iteration. The curve

marked WFE in Figure 4 illustrates the implementation where the processors wait for their essential neighbors. We see a decrease in the deteriorations and as a result an improvement in the convergence. Let t_1 and t_2 be two consecutive updates of block i as above. For $t_2 \gg t_1$ old information is contaminating the residual. It is observed in the sequential case that the residual $\|r(t)\|_M$ is monotonically decreasing for SOR method, and $\|r(t)\|_M \leq \|r(t-g)\|_M$ for Jacobi [4]. In our experiment for checking the effect of a relaxation parameter we choose ω_i such that $\|r(t_2)\|_M$ is minimized. The values of s_i that give $(A_i s_i)^T Mr \geq 0$ are discarded. Otherwise, ω_i is chosen as $\omega_i = -(A_i s_i)^T Mr/\|A_i s_i\|_M^2, \;\; 0 < \omega_i < 2$. As seen in Figure 5, this choice for ω_i reduces the effect of old information on the residual.
Elsner, Neumann and Vemmer [5] showed that increasing the number of processors means that older information is used to calculate any new iterate. This reduces the rate of convergence. In their experiment there is a time-lag of p on the updates. To avoid that too old information is used to update the residual in the asynchronous implementation we introduce a limit Np on the magnitude of time-lag, i.e., $t - Np \leq \tau_j^i(t) \leq t-1$ for all i and j, and all $t \geq 0, t \in \mathcal{T}^i$. In [2], this is called a partially asynchronous iterative method. We use $p = g$ processors in a heterogeneous environment. When each processor has updated the residual a fixed number of times (N), we flush the queue at the master and broadcast the new residual. Letting $N = 1, 2, 3$, we observe that $N = 1$ gives (approximately) Jacobi's method and $N \geq 3$ gives (approximately) totally asynchronous method (Figure 6).

CONCLUDING REMARKS

Increasing the number of processors decreases the convergence rate since the effect of old information is increased, as proved by Elsner, Neumann and Vemmer [5]. In our attempt to decrease the effect of old information using heterogeneous processors, we find out that though the deterioration of convergence is damped, the damping effect is not significant. When the role of essential neighbors in the computation for corrections is taken into consideration, the result is an improvement in convergence and degradation in deteriorations. Choice of ω as a line search prevents the contamination of the residual by old information. Introduction of synchronization after an a priori chosen number of corrections on the residual lessens the effect of old information and improves the convergence rate. However, the effect of synchronization decreases rapidly with the "age" of the updates.
To our knowledge, the results of using asynchronous iterations on linear least squares problems have not been discussed in literature. We have introduced two open problems: the effect of a relaxation parameter and synchronization. So far there is no theory to characterize old information, only heuristics. A relaxation parameter can be used to reduce the deteriorations. Synchronization is needed in many cases, but there is no theory to support *when* to synchronize. Synchronization after only one or two updates from each block has a damping effect on the old information, but later synchronizations do not cure the deteriorations.

REFERENCES

[1] G. M. Baudet, *Asynchronous Iterative Methods for Multiprocessors*, J. of the ACM 25, p. 226 (1978).

[2] D. P. Bertsekas, J. N. Tsitsiklis, Parallel and Distributed Computation, Numerical Methods, Prentice-Hall Inc., Englewood Cliffs, NJ (1989).

[3] Å. Björck, Numerical Methods for Least Squares Problems, SIAM, Philadelphia, PA (1996).

[4] J. E. Dennis, Jr., T. Steihaug, *On the Successive Projections Approach to Least Squares Problems*, SIAM J. Numer. Anal 23, p. 717 (1986).

[5] L. Elsner, M. Neumann, B. Vemmer, *The Effect of the Number of Processors on the Convergence of the Parallel Block Jacobi Method*, Lin. Alg. Appl 154–156, p. 311 (1991).

[6] P. Kolm, P. Arbenz, W. Gander, *Generalized Subspace Correction Methods for Parallel Solution of Linear Systems*, Tech. Rep. TRITA-NA-9509, C2M2, Nada, KTH, Sweden (1995).

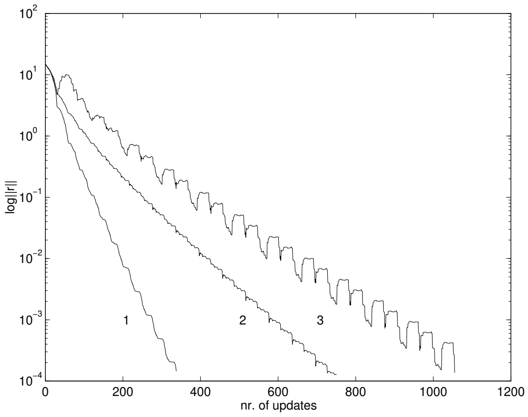

Figure 1: *Sequential versus totally asynchronous. (1): Gauss-Seidel (2): Jacobi (3): Totally asynchronous*

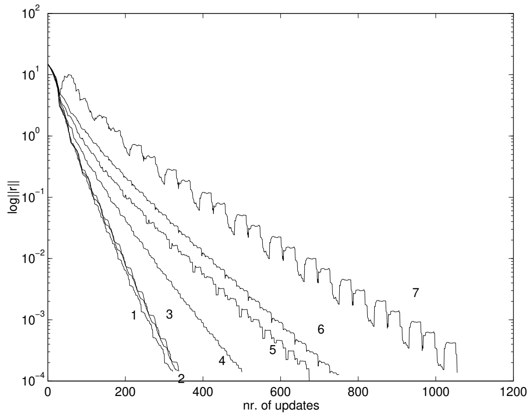

Figure 2: *Totally asynchronous on p processors. (1): $p = 6$ (2): $p = 3$ (3): Gauss-Seidel (4): $p = 10$ (5): $p = 15$ (6): Jacobi (7): $p = 30$*

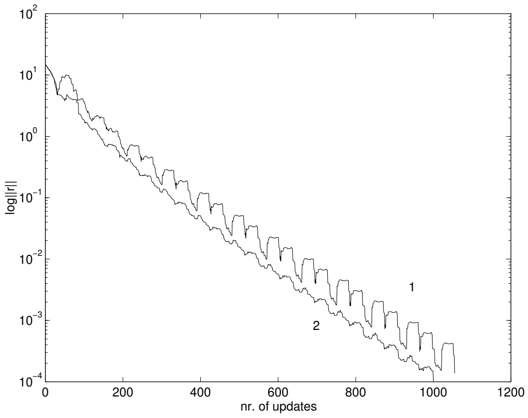

Figure 3: *Homogeneous versus heterogeneous processors. (1): $p = 30$ identical processors (2): $p = 30$ processors with computing speed varying with 1-7*

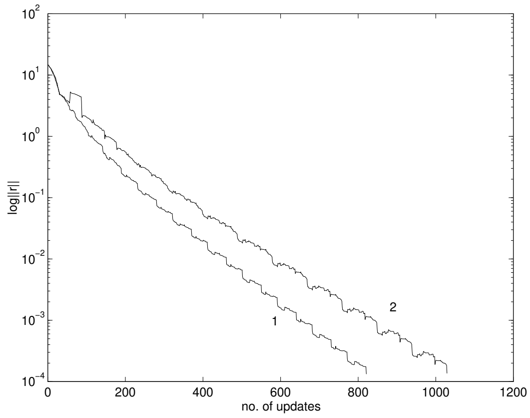

Figure 4: *"Wait-for-essential-neighbor"(WFE) versus totally asynchronous. (1): WFE on $p = 30$ processors (2): Totally asynchronous on $p = 30$ processors*

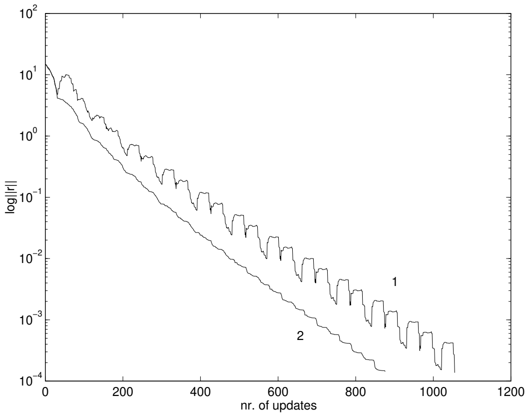

Figure 5: *Asynchronous SOR. (1): $\omega = 1.0$, (2): Line search*

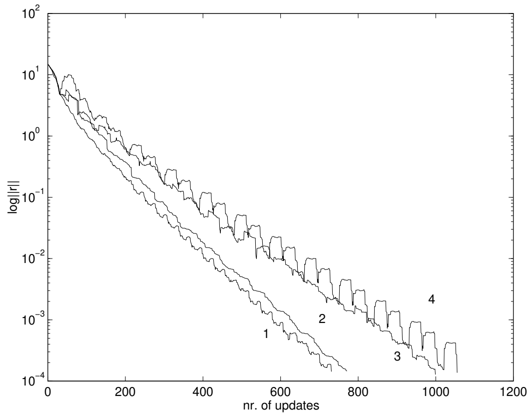

Figure 6: *Synchronization after N updates versus totally asynchronous. (1): $N = 1$ ($\approx$ Jacobi) (2): $N = 2$ (3): $N = 3$ (4): Totally asynchronous*

[7] S. A. Savari, D. P. Bertsekas, *Finite Termination of Asynchronous Iterative Algorithms*, Parallel Computing 22, p. 39 (1996).

[8] Y. Yalçınkaya, *Asynchronous Solution of Linear Least Squares Problems Using Generalized Group Iterative Methods*, Master's Thesis, University of Bergen, Norway (1995).