A Convergence Proof for Asynchronous Over-Relaxation

Robert Leland and John Rollett

Abstract

We present a proof that the conditions ensuring convergence of successive over-relaxation (SOR) also guarantee convergence of asynchronous over-relaxation (AOR). That is, the standard over-relaxation algorithm will converge regardless of the order of updating, even if the order changes from iteration to iteration. Our proof is a generalization of the Ostrowski theorem and has an attractive simplicity in comparison with previous results in this area. An immediate corollary of practical value is that parallel implementations of over-relaxation, which often use non-standard update orderings, converge under the usual conditions.

This work was supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract No. DE-AC04-76DP00789.
1. Introduction. In serial computers data is accessed one unit at a time and in a predictable order, for example the lexical order used in standard successive over-relaxation (SOR), and it is natural and efficient to organise the calculation around this order. But parallel computers may operate on subdomains of the data concurrently, obscuring this natural order. Sequential algorithms adapted for parallel execution therefore usually employ some device to maintain the correct ordering of data use. If however the restrictions imposed by these devices were removed or relaxed, a new family of asynchronous algorithms would result. These are of interest because part of the efficiency loss in sequential algorithms adapted for parallel use stems from the checks needed to enforce data order and from processor idling when data does not arrive in the desired order.

This technical note concerns an asynchronous form of over-relaxation (AOR) in which meshpoints are updated using either old or new data from neighboring meshpoints. Nodes are not allowed to get more than one step ahead of their neighbors, so this is a one step algorithm and amounts to allowing an arbitrary order of relaxation.

2. Algorithm and convergence theory. The five point form of AOR is written

\[ u_{j,k}^{m+1} = (1 - \omega) u_{j,k}^m + \frac{\omega}{4} [u_{j-1,k}^m + u_{j+1,k}^m + u_{j,k-1}^m + u_{j,k+1}^m] \]

where \( m + 1 \) indicates a non-deterministic choice between the new and old state value from a neighboring node. The following theorem establishes the convergence of this iteration for the model problem and other problems generated by real, self-adjoint operators. The basic strategy of the proof is to apply a permutation matrix at each iteration which rotates the system into the usual ordering. We then apply a standard convergence proof to the rotated system and unpermute the system at the end.

**Theorem 2.1.** The AOR iteration converges for a symmetric, positive definite, real linear system if and only if \( 0 < \omega < 2 \). **Proof** Consider reordering a linear system

\[ By = c \]

which satisfies the theorem conditions by application of an orthogonal permutation matrix, \( P_m \), chosen so that variables in the permuted system are in the sequence in which they are updated on the \( m^{th} \) iteration. We have

\[ P_m B P_m^T P_m y = P_m c \]

or

\[ A_m x_m = b_m \]

where

\[ A_m \equiv P_m B P_m^T \quad B = P_m A P_m \quad x_m \equiv P_m y \quad y = P_m^T x_m \quad b_m \equiv P_m c \quad c = P_m^T b_m. \]

Notice that

\[ x_m^T A_m x_m = y^T B y > 0 \quad \forall y \neq 0 \]

since \( B \) is positive definite. But if \( x_m \neq 0 \), its permutation \( y \neq 0 \), so

\[ x_m^T A_m x_m > 0 \quad \forall x_m \neq 0, \]

i.e. \( A_m \) is positive definite too. It is also symmetric since the rows and columns of the symmetric \( B \) have been permuted in the same way.
Carrying out the standard additive splitting \( A_m = D_m - L_m - U_m \), we can write the AOR iteration matrix for this step as

\[
M_m = (D_m - \omega L_m)^{-1}[(1 - \omega)D_m + \omega U_m]
\]

where \( \omega \) is assumed constant. We now suppress the subscript \( m \) and proceed to show that \( M \) is well defined.

**Lemma 1.** \( D \) is positive definite. Proof Let \( d_i \) be the \( i^{th} \) diagonal element and hence eigenvalue of \( D \). Let \( u_i \) be the unit vector in the \( i^{th} \) coordinate direction. Then

\[
d_i = u_i^T A u_i > 0
\]

since \( A \) is positive definite. \( \square \)

**Lemma 2.** \((D - \omega L)^{-1}\) exists.

Proof Since \((D - \omega L)\) is triangular, it has the same eigenvalues as \( D \). These are nonzero by lemma 1. \( \square \)

These lemmas establish that the permuted system satisfies the preconditions of Ostrowski’s theorem (1954, see Varga [4]) proving the convergence of LexSor, which we now present in modified form.

First we deduce that the iterative procedure has an essential property.

**Lemma 3.** The solution \( x \) is the unique fixed point of the iteration

\[
x_{n+1} = M x_n + \omega(D - \omega L)^{-1} b.
\]

Proof Let \( ^1 \)

\[
x_n = x + \epsilon
\]

\[
x_{n+1} = x + \delta.
\]

Substitution in (10) yields

\[
\delta = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]\epsilon
\]

so \( x \) is a fixed point since \( \epsilon = 0 \rightarrow \delta = 0 \). It is unique since \( \epsilon \neq 0 \rightarrow \delta \neq \epsilon \). Otherwise (if \( \delta = \epsilon \))

\[
(D - \omega L)^{-1}[(1 - \omega)D + \omega U] = I
\]

implying \( A\delta = 0 \). Now, either \( \delta = 0 \), in which case \( \epsilon = 0 \) in contradiction of the assumption, or \( A \) is singular, contradicting the positive definiteness of \( A \). \( \square \)

Clearly \( \epsilon \) is the error \( e_n \equiv x_n - x \) at the \( n^{th} \) step, and similarly \( \delta = e_{n+1} \). The error obeys the homogeneous equation

\[
e_{n+1} = (D - \omega L)^{-1}[(1 - \omega)D + \omega U]e_n
\]

and it will be convenient to proceed in terms of this rather than the full iteration (10). Combining it with a shift \( s_n \equiv e_{n+1} - e_n \) yields

\[
s_n = -\omega(D - \omega L)^{-1} A e_n
\]

which we can relate to another expression for the shift in the \( A \) norm:

\[
e_{n+1}^T A e_{n+1} - e_n^T A e_n = (e_n + s_n)^T A(e_n + s_n) - e_n^T A e_n
\]

\[
= e_n^T A s_n + s_n^T A e_n + s_n^T A s_n.
\]

\(^1\) Note that \( x_n \) is the \( n^{th} \) approximation to the \( m^{th} \) permutation of the solution vector, \( y \). If not suppressing permutation subscripts, we would write \( (x_m)_n = (x_m) + \epsilon_m, \) etc.
From (15) we have

\[(17) \quad A e_n = \left( L - \frac{D}{w} \right) s_n \]

and (using the symmetry of A)

\[(18) \quad e_n^T A = s_n^T \left( U - \frac{D}{w} \right) . \]

These allow us to recast (16) as

\[(19) \quad e_{n+1}^T A e_{n+1} - e_n^T A e_n = \left( 1 - \frac{2}{\omega} \right) s_n^T D s_n . \]

By lemma 3 we get a nonzero shift unless we have arrived at the answer, hence from (19)

\[(20) \quad e_{n+1}^T A e_{n+1} < e_n^T A e_n \quad \forall \omega \text{ s.t. } 0 < \omega < 2 \]

We now know the error decreases monotonically in the A norm, and need only rule out asymptotic decay to a nonzero value. From (16) we get

\[(21) \quad \| s_n \| \geq c \| e_n \| \]

where

\[(22) \quad c = \frac{\omega}{\| A^{-1} \| \| D - \omega L \| } = \frac{\omega \lambda_{\text{min}}}{\| D - \omega L \|} \]

and \( \lambda_{\text{min}} \) is the smallest eigenvalue of the symmetric A. Hence (19) becomes

\[(23) \quad \frac{e_{n+1}^T A e_{n+1}}{e_n^T A e_n} \leq 1 - \frac{1 - \frac{2}{\omega} \| e_n^T D e_n \|}{\| e_n^T A e_n \|} \leq 1 - \frac{1 - \frac{2}{\omega} \| e_n^T \|}{\lambda_{\text{max}}} \]

where \( \lambda_{\text{max}} \) is the largest eigenvalue of A. So

\[(24) \quad \frac{e_{n+1}^T A e_{n+1}}{e_n^T A e_n} \leq 1 - \omega (2 - \omega) \frac{\lambda_{\text{min}}^2}{\| D - \omega L \|^2 \lambda_{\text{max}}} \frac{d_{\text{min}}}{d_{\text{max}}} . \]

Since \( \| D - \omega L \| \geq \lambda_{\text{max}} \) of \((D - \omega L)\), and \( \lambda_{\text{max}} \geq d_{\text{max}} \), this reduces to

\[(25) \quad \frac{e_{n+1}^T A m e_{n+1}}{e_n^T A m e_n} \leq 1 - \omega (2 - \omega) \left( \frac{\lambda_{\text{min}}}{d_{\text{max}}} \right) \left( \frac{\lambda_{\text{max}}}{d_{\text{max}}} \right) \left( \frac{d_{\text{min}}}{d_{\text{max}}} \right) \leq 1\]  

where the permutation subscript has been restored and the iteration index changed to \( m \) since the \( n^{th} \) iteration is also the \( m^{th} \). With the condition on \( \omega \), this shows that the A norm of the error decreases by at least some constant factor on each iteration, regardless of permutation. Therefore

\[(26) \quad \lim_{m \to \infty} e_m = 0 \implies \lim_{m \to \infty} e_m = 0\]

since A is positive definite.

We now return to the original system, \( B y = c \), and define an error vector

\[(27) \quad f_m \equiv y_m - y . \]

Since

\[(28) \quad P_m f_m = x_m - x = e_m \]
we can write

$$\lim_{m \to \infty} f_m = \lim_{m \to \infty} P_m^T e_m = 0. \quad \Box$$

**Corollary 2.2.** The theorem holds for complex Hermitian systems since the lemmas and Ostrowski theorem hold.

A natural way to approach parallelising relaxation would be to have processors iterate using a standard order within subdomains and then communicate their results along subdomain boundaries. But this results in unusual or even unpredictable global orderings which in general depend upon the geometry and timing of the system. Usually it is taken for granted that these orderings converge, but we can now make a rigorous statement about both these methods and the multi-colour schemes proposed by Adams.

**Corollary 2.3.** Multi-colour and block versions of SOR converge under the standard conditions since they correspond to AOR with some (perhaps constant) permutation matrix.

It is tempting to use (24) to reason about the convergence rate since the quantities involved are known or can be reasonably estimated, but this is probably not worthwhile. For the model problem, for example, the best error ratio bound given by (24) is $1 - \frac{h^2}{k^2}$ for $w = 1$. But Goodman and Madaras [3] have shown using random walk theory that the convergence of Gauss-Seidel is independent of any fixed ordering, and this is necessarily true of the Jacobi algorithm as well. Numerical simulations performed on a standard serial computer showed that for a variety of boundary and initial conditions the number of iterations unaccelerated AOR (i.e. randomly ordered Gauss-Seidel) required to converge was almost exactly half the $O(h^2)$ steps required by Jacobi. This is the relationship commonly observed with fixed orderings on serial machines, and we have said that there is no distinction between any of the possible fixed orderings for either algorithm, so it does not seem unreasonable that there should be no difference when random orders are permitted. Both theory and observation therefore suggest that the error bound is too weak to help in estimation of convergence rates.

In their well known paper, Chazan and Miranker [2] considered chaotic relaxation, in which nodes are allowed to get some finite number of steps out of time and gave conditions under which these multistep schemes converge. But while their work is much more general than theorem (2.1), it is also much harder to understand and therefore check. In contrast, theorem (2.1) is relatively easy to understand and still gives a very useful result.

### 3. Parallel Implementation

**The ONDE Kernel for AOR**

```plaintext
PAR
NorthIn? n
EastIn? e
SouthIn? s
WestIn? w
NorthOut! state
EastOut! state
SouthOut! state
WestOut! state
```

allows the state to be updated with any mix of new or old data since the order of execution of the statements under the PAR operator is logically unconstrained. There might in fact be internal constraints, but nothing can be assumed about the statement order. This means that the kernel is actually not legal Occam since it would allow, for example, the memory location of the variable n to be overwritten by the NorthIn? n statement while the line of code updating state was trying to read the same location. The result would be spurious, so the compiler rejects the code. To circumvent this, we can turn off resource conflict checking, but then the results do seem spurious. This is discussed further in section (2.4).

The AOR kernel was actually implemented within the node structure for RbSor since that way the statement order within the red AOR kernel could be different from that within the black kernel.
The statement orders were thoroughly scrambled many different ways to make sure that there was no implicit synchronisation.

4. **Deadlock analysis.** Once the standard idealisations have been made, we see that the AOR node is exactly the canonical node and is therefore deadlock free by theorem (??).

5. **Single processor performance.** The single processor execution times for AOR were about 1% longer per iteration than those for RbSor, probably because in RbSor corresponding inputs and outputs on neighboring nodes line up naturally, whereas some internal checking is needed to secure a rendezvous between scrambled statements in AOR. So AOR has, ironically, *greater* synchronisation overhead than its synchronous counterpart.

6. **Multi-processor performance.** If all resource conflict checking is turned off, the code does run, but with occasional dramatic "glitches" in the value of state. These might be the result of logical errors in the program, but it seems more likely that they are in fact due to the overwriting error -- only the kernel is new, and the usual symptom of a logical error is deadlock, but we have shown that this kernel should not deadlock and in practice it does not. It may seem puzzling that overwriting would occur given that any particular kernel is actually executed on a single processor, but the processor shares time between processes declared in parallel, so overwriting could in effect occur between machine instructions comprising an Occam statement.

Various guarding strategies could be used to prevent harmful overwriting, but those would certainly add to the kernel overhead which is already greater than that of RbSor. AOR would need to achieve sufficiently faster iterative convergence on a consistent basis to justify its use on our system. Numerical simulations suggest that this is not the case. Sometimes accelerated AOR required more or fewer iterations than optimally accelerated SOR as problem size, error norm and simulated processor speed distribution were varied, but no clear pattern could be discerned from the limited testing performed, except that $\omega_{opt}$ for AOR was higher than for RbSor. As the number of steps to tolerance was generally comparable, we will conjecture that

$$\langle \rho_{AOR} \rangle = \rho_{opt}(RbSor)$$

where $\langle \rho_{AOR} \rangle$ is the expectation value of the AOR spectral radius. If correct, that implies

$$\kappa_{AOR}^{RbSor} = \frac{\eta_{RbSor}(P,N)}{\eta_{opt}(P,N)} \frac{\Omega_{Popt}(1,N)}{\Omega_{RbSor}(1,N)} \frac{\rho_{opt}(1,N)}{\rho_{RbSor}(1,N)} \approx 1$$

since the efficiencies and sweep costs are so similar. With additional guarding to guarantee correctness, we get $\kappa_{AOR} < \kappa_{RbSor}$, and there is no reason to prefer AOR over RbSor.

This does not mean that asynchronous algorithms have no practical value. As the canonical algorithm in the ONDE programming style, AOR served a worthwhile purpose in unifying the convergence and deadlock freedom theory of the stationary methods, but it is only a single step asynchronous method operating in an environment with a great deal of implicit synchronisation on a local scale. Multistep asynchronous methods running on different architectures have been shown to be significantly better than their synchronous counterparts in some cases. Baudet, for example, achieved positive results with asynchronous Jacobi and Gauss-Seidel algorithms, and his paper is now a standard reference in the field [1]. More recently, Womble [5] achieved similar results on the 1024 node hypercube at Sandia National Laboratories. Although both Jacobi and Gauss-Seidel required substantially more iterations in asynchronous form, total run time was approximately halved because the machine is very sensitive to synchronisation overhead. What is perhaps more significant than the specific result is that the synchronisation sensitivity *increased* with the number of processors used. As this is a much larger distributed memory, MIMD machine than has been previously studied, it seems likely that the merits of asynchrony have yet to be fully appreciated.

Finally we note that accelerated asynchronous algorithms do not seem to have been thoroughly studied. With acceleration added, the competitive interaction between efficiency, iterative convergence and sweep complexity as a function of architecture will probably be more complex and difficult.

---

\(^2\) An excellent overview of both theoretical and practical work on asynchronous algorithms.
to understand. The parallel effectiveness model may provide a useful framework for modeling this interaction.

REFERENCES


