Parallel Computing 1 (1984) 3–18
North-Holland

Parallel S.O.R. iterative methods

D.J. EVANS

Department of Computer Studies, Loughborough University of Technology, Loughborough, Leicestershire, U.K.

Received March 1984

Abstract. New explicit group S.O.R. methods suitable for use on an asynchronous MIMD computer are presented for the numerical solution of the sparse linear systems derived from the discretization of two-dimensional, second-order, elliptic boundary value problems. A comparison with existing implicit line S.O.R. schemes for the Dirichlet model problem shows the new schemes to be superior (Barlow and Evans, 1982).

Keywords. Explicit group S.O.R. methods, red-black ordering, asynchronous MIMD computer system, performance analysis.

1. Introduction

In this paper iterative techniques are considered for solving the linear system of equations,

$$A\phi = b, \tag{1}$$

where A is a given $N \times N$ non-singular matrix, b is a given $N \times 1$ vector and ϕ is the $N \times 1$ solution vector on a parallel computer. We shall be concerned with those linear systems which arise in the solution of two-dimensional second-order, elliptic boundary value problems. The problem of interest is as follows: let R be a bounded region in the (x, y) plane and S its boundary. Define the second-order linear operator L as,

$$L[\phi] = A\phi_{xx} + C\phi_{yy} + D\phi_x + E\phi_y + F\phi = G, \tag{2}$$

where A, C, D, E, F and G are continuous and real valued in R, and

$$A > 0,\ C > 0 \text{ and } F \leqslant 0 \text{ in } R. \tag{3}$$

Given a continuous function $\phi_0(x, y)$ defined on S, find a real-valued function ϕ which is continuous in $R + S$, twice differentiable in R, satisfies (2), and also satisfies certain boundary conditions on S. This is summarized by:

$$\begin{aligned} &\text{find } \phi(x, y) \text{ which is continuous on } R + S \text{ and} \\ &\text{is twice continuously differentiable on } R, \\ &\text{where } \phi(x, y) = \phi_0(x, y) \text{ on } S \text{ and } L[\phi] = G(x, y) \text{ on } R. \end{aligned} \tag{4}$$

For simplification we let R be the unit square, $0 \leqslant x \leqslant 1$ and $0 \leqslant y \leqslant 1$, and S be the boundary, and we consider Laplace's differential equation (i.e. the Model Problem),

$$L[\phi] = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0, \tag{5}$$

and the boundary conditions,

$$\phi(x, y) = \phi_0(x, y) \quad \text{on } S. \tag{6}$$

The problem defined is a special case of the Dirichlet problem.

Usually, analytic solutions of (2) are not known and when they are known; it is often very difficult to satisfy prescribed boundary conditions. In the application of finite difference methods the region R is replaced by a finite set of point R_h, and likewise, the boundary S is replaced by a finite set of points S_h. In this discussion, only a square mesh is considered. For each point $P \in R_h$ a linear relation is established between the value of ϕ at P and certain other points of R_h and S_h. The value of ϕ at all points of S_h is determined by the prescribed boundary conditions given by (6). If there are m^2 points in R_h, the resulting linear system (1) is made up of m^2 equations in m^2 unknowns. The values of ϕ at points of R_h obtained by solving (1) are accepted as approximate solutions of (2), and interpolation techniques can be used to obtain approximate solutions of the differential equation (2), for all points of R.

The procedure considered for expressing a linear relationship between points of R_h and S_h is to replace the partial derivatives in (2) by their equivalent three-point central difference formulae, given by

$$\begin{aligned}
&\phi_x(x,y) \approx [\phi(x+h,y)-\phi(x-h,y)]/2h,\\
&\phi_y(x,y) \approx [\phi(x,y+h)-\phi(x,y-h)]/2h,\\
&\phi_{xx}(x,y) \approx [\phi(x+h,y)+\phi(x-h,y)-2\phi(x,y)]/h^2,\\
&\phi_{yy}(x,y) \approx [\phi(x,y+h)+\phi(x,y-h)-2\phi(x,y)]/h^2,
\end{aligned} \tag{7}$$

where h is the size of the mesh (Fig. 1). Substituting (7) into (2) gives the difference equation (8) for the general case

$$\begin{aligned}
L_h[\phi] = {} & A(x,y)[\phi(x+h,y)+\phi(x-h,y)-2\phi(x,y)]/h^2\\
& + C(x,y)[\phi(x,y+h)+\phi(x,y-h)-2\phi(x,y)]/h^2\\
& + D(x,y)[\phi(x+h,y)-\phi(x-h,y)]/2h\\
& + E(x,y)[\phi(x,y+h)-\phi(x,y-h)]/2h + F(x,y)\phi(x,y)\\
= {} & G(x,y)
\end{aligned} \tag{8}$$

which reduces to

$$\begin{aligned}
L[\phi] = {} & \alpha_0\phi(x,y)+\alpha_1\phi(x+h,y)+\alpha_2\phi(x-h,y)\\
& + \alpha_3\phi(x,y+h)+\alpha_4\phi(x,y-h)\\
= {} & G(x,y)
\end{aligned} \tag{9}$$

where

$$\begin{aligned}
&\alpha_1 = \frac{A}{h^2}+\frac{D}{2h}, \quad \alpha_2 = \frac{C}{h^2}+\frac{E}{2h},\\
&\alpha_3 = \frac{A}{h^2}-\frac{D}{2h}, \quad \alpha_4 = \frac{C}{h^2}-\frac{E}{2h},\\
&\alpha_0 = -(\alpha_1+\alpha_2+\alpha_3+\alpha_4)+F.
\end{aligned} \tag{10}$$

For the model problem (9) simplifies to (see also Fig. 1)

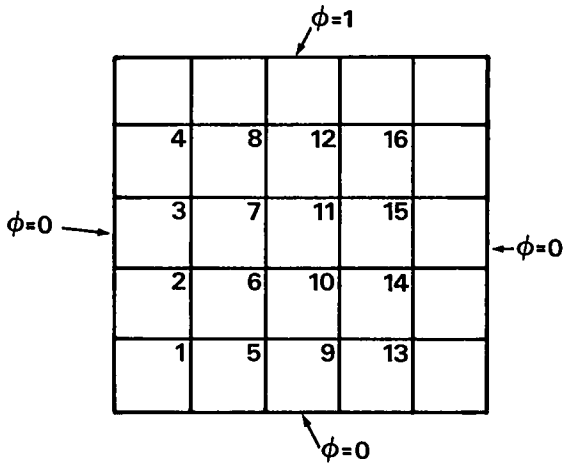
$$L_h[\phi] = \frac{\phi(x+h,y)+\phi(x-h,y)+\phi(x,y+h)+\phi(x,y-h)-4\phi(x,y)}{h^2} = 0, \tag{11}$$

or

$$\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j} = 0, \tag{12}$$

where $\phi_{i,j} = \phi(ih, jh)$, and is represented conveniently by the 'molecule' of Fig. 2.

Now consider the following numbering of the internal mesh points of a (6×6) mesh with Dirichlet boundary conditions:



(13)

Then, by applying equation (12) to each of the internal mesh points, we produce the following system of linear equations,

$$\left[\begin{array}{cccc|cccc|cccc|cccc}
4 & -1 & & & -1 & & & & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & & & & \\
 & -1 & 4 & -1 & & & -1 & & & & & & & & & \\
 & & -1 & 4 & & & & -1 & & & & & & & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & & & & & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & & & & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & & & & & \\
 & & & -1 & & & -1 & 4 & & & & -1 & & & & \\
\hline
 & & & & -1 & & & & 4 & -1 & & & -1 & & & \\
 & & & & & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & & & & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & & & & & -1 & & & -1 & 4 & & & & -1 \\
\hline
 & & & & & & & & -1 & & & & 4 & -1 & & \\
 & & & & & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & & & & & -1 & & & -1 & 4 & -1 \\
 & & & & & & & & & & & -1 & & & -1 & 4
\end{array}\right]
\left[\begin{array}{c}\phi_1\\ \phi_2\\ \phi_3\\ \phi_4\\ \phi_5\\ \phi_6\\ \phi_7\\ \phi_8\\ \phi_9\\ \phi_{10}\\ \phi_{11}\\ \phi_{12}\\ \phi_{13}\\ \phi_{14}\\ \phi_{15}\\ \phi_{16}\end{array}\right]
=
\left[\begin{array}{c}0\\0\\0\\1\\0\\0\\0\\1\\0\\0\\0\\1\\0\\0\\0\\1\end{array}\right]$$

(14)

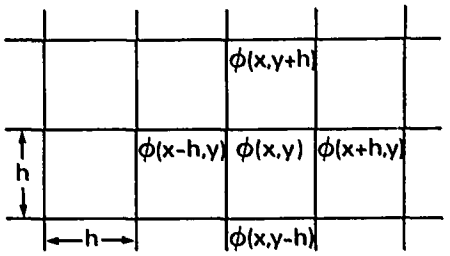


Fig. 1.

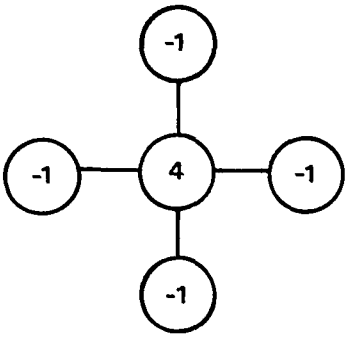


Fig. 2.

Thus, a system of linear equations has been created by replacing the partial differential equation by a finite-difference equation at each of the internal mesh points.

2. The solution of a large sparse system of linear equations

In this section we shall define some basic iterative formulae that may be used to solve the system of equations (14).

2.1 The Point Jacobi Method

Since, at each internal mesh point we have

$$\phi_{i,j} = \tfrac{1}{4}\left(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}\right), \tag{15}$$

a simple iterative formula would be

$$\phi_{i,j}^{(n+1)} = \tfrac{1}{4}\left(\phi_{i+1,j}^{(n)} + \phi_{i-1,j}^{(n)} + \phi_{i,j+1}^{(n)} + \phi_{i,j-1}^{(n)}\right), \tag{16}$$

where $\phi_{i,j}^{(n)}$ represents the nth iterate or approximation to ϕ at point (ih, jh). This is called the Point Jacobi method. Clearly the $(n+1)$st iterates are expressed exclusively in terms of nth iterates and so the order in which they are evaluated with respect to the mesh points does not effect their values or the rate of convergence to the solution. Hence this method is called the Simultaneous Displacement Method. Unfortunately, the rate of convergence of this method is slow and hence it is rarely used.

An improvement in the convergence rate can be obtained by applying the formula,

$$\phi_{i,j}^{(n+1)} = (1-\omega)\phi_{i,j}^{(n)} + \tfrac{1}{4}\omega\left(\phi_{i+1,j}^{(n)} + \phi_{i-1,j}^{(n)} + \phi_{i,j+1}^{(n)} + \phi_{i,j-1}^{(n)}\right) \tag{17}$$

at each point (i,j) of the grid where $\omega > 1$. This method is called the J.O.R. and the basic principle involved in the acceleration process is described in the next section.

2.2 The Gauss–Seidel Method

The Point Jacobi formula (16) may be improved by using the latest values of $\phi_{i,j}$ as soon as they are available. If we assume that the $(n+1)$st iterative values have been calculated along

columns 1, 2,...,$(j-1)$ and as far as point $(i-1, j)$ along column j, and that the $(n+1)$st value at point (i, j) is the next to be calculated, then the Gauss–Seidel formula gives

$$\phi_{i,j}^{(n+1)} = \tfrac{1}{4}\left(\phi_{i+1,j}^{(n)} + \phi_{i-1,j}^{(n+1)} + \phi_{i,j+1}^{(n)} + \phi_{i,j-1}^{(n+1)}\right). \tag{18}$$

With this method, we have the added advantage of only needing to store the latest value of each $\phi_{i,j}$. This method is a Successive Displacement Method.

2.3 The Successive Over-Relaxation Method (S.O.R.)

If $\phi_{i,j}^{(n)}$ is added and subtracted to the right-hand side of equation (16), we have

$$\begin{aligned}\phi_{i,j}^{(n+1)} &= \phi_{i,j}^{(n)} + \tfrac{1}{4}\left(\phi_{i+1,j}^{(n)} + \phi_{i-1,j}^{(n+1)} + \phi_{i,j+1}^{(n)} + \phi_{i,j-1}^{(n+1)} - 4\phi_{i,j}^{(n)}\right)\\ &= \phi_{i,j}^{(n)} + r_{i,j}.\end{aligned} \tag{19}$$

Obviously, $r_{i,j}$ is the change in value of $\phi_{i,j}$ for one Gauss–Seidel iteration. The rate of convergence of the Gauss–Seidel method can be 'accelerated' by making a larger change to $\phi_{i,j}$ thus,

$$\phi_{i,j}^{(n+1)} = \phi_{i,j}^{(n)} + \omega r_{i,j}, \tag{20}$$

where ω is positive constant called the acceleration factor which in practice lies between 1 and 2. This equation is called the Successive Over-Relaxation formula and may be rewritten in the form

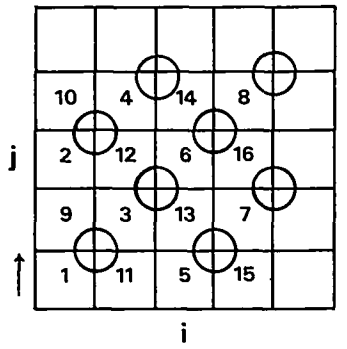
$$\phi_{i,j}^{(n+1)} = (1-\omega)\phi_{i,j}^{(n)} + \tfrac{1}{4}\omega\left(\phi_{i+1,j}^{(n)} + \phi_{i-1,j}^{(n+1)} + \phi_{i,j+1}^{(n)} + \phi_{i,j-1}^{(n+1)}\right) \tag{21}$$

from which it is clear that it is a linear combination of the Gauss–Seidel iterate (18) and the nth iterate. (Note that when $\omega = 1$, the S.O.R. method becomes the Gauss–Seidel method.) In this method it is also only necessary to store the latest values of $\phi_{i,j}$ and it is a Successive Displacement Method.

3. The solution of the Dirichlet problem by S.O.R. methods on a parallel computer

It is obviously a trivial problem to perform the Point Jacobi method on a parallel computer, since each iteration comprises of m^2 independent evaluations defined by (15). The use of Successive Displacement methods on a parallel computer is not so simple since the order in which the $(n+1)$th iterates are evaluated, particularly with S.O.R., is important. We have seen, in the previous section, that with the S.O.R. method it is desirable for matrix A to possess property A (Young, [8]) and be consistently ordered, and so, when S.O.R. is performed on a parallel computer, it is useful but not vital to preserve these properties.

If the order in which the $(n+1)$st iterates are evaluated is called an ordering, then an ordering that produces a coefficient matrix A which is consistently ordered is a consistent ordering. The row-wise or column-wise ordering of the mesh points in (13) is a consistent ordering. This ordering, however, is of little use for a parallel computer because it is essentially sequential. A much more useful ordering is the red-black ordering



(22)

Clearly, the red-black ordering consists of two passes over the meshes. During the first pass we evaluate the $(n+1)$st iterates at alternate mesh points (circled in (22)) beginning at 1, and in the second pass the remaining points (uncircled in (22)), are dealt with. If the finite difference equation (12) is applied in this order, then using S.O.R. we have for the first pass,

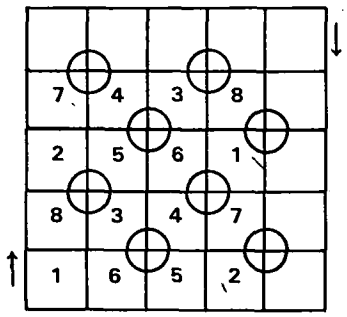
$$\phi_{i,j}^{(n+1)} = (1-\omega)\phi_{i,j}^{(n)} + \tfrac{1}{4}\omega\left(\phi_{i+1,j}^{(n)} + \phi_{i-1,j}^{(n)} + \phi_{i,j+1}^{(n)} + \phi_{i,j-1}^{(n)}\right) \quad \text{for all } i+j \text{ odd} \tag{23}$$

and during the second pass,

$$\phi_{i,j}^{(n+1)} = (1-\omega)\phi_{i,j}^{(n)} + \tfrac{1}{4}\omega\left(\phi_{i+1,j}^{(n+1)} + \phi_{i-1,j}^{(n+1)} + \phi_{i,j+1}^{(n+1)} + \phi_{i,j-1}^{(n+1)}\right) \quad \text{for all } i+j \text{ even.} \tag{24}$$

Thus, the first pass consists of independent evaluations which may be carried out simultaneously and similarly so does the second pass. It is unimportant how the evaluations in each pass are shared between the processors provided it is done evenly and the first pass is completed before the second pass is commenced (this is to ensure that during the second pass, the $(n+1)$st iterates are available when required).

An alternative application of the red-black ordering for a two processor parallel computer system can be produced by applying the technique of folding, which we shall call the parallel R.B. ordering. In this ordering, the two passes of the red-black ordering are executed simultaneously, in the following manner:



(25)

Processor 1 evaluates the $(n+1)$st iterates at the uncircled mesh points while processor 2 evaluates the $(n+1)$st iterates at the circled mesh points in the order 1 and ①, 2 and ② etc. The $(n+1)$st iterates are defined by equation (23) before the processors cross, and by (24) after they have crossed, for both processors.

For both the red-black and parallel R.B. orderings the number of parallel operations per iteration is

$$3\left\lfloor\frac{m^2}{2}\right\rfloor \text{multiplications} + 5\left\lfloor\frac{m^2}{2}\right\rfloor \text{additions,} \tag{26}$$

where m^2 is the number of internal mesh points.

In order to compare the two mesh-point orderings, they are both used to solve the Dirichlet problem for different mesh sizes. In each case, $\rho(G)$ the spectral radius of the Gauss–Seidel method is estimated using the power method and, by using this value the optimum value of ω_b is obtained. An experimental optimum value of ω is also found by solving the problem using different values of ω. The value of ω is initially set to 1 and incremented by $\Delta\omega$ until the number of iterations required to satisfy the conditions of convergence begins to increase. Then, in the vicinity of the value of ω that requires the least number of iterations, a smaller value of $\Delta\omega$ is used. The process is repeated until the region, in which the least number of iterations are required for convergence, is found to the required degree of accuracy. The experimental best value of ω, say ω_e, is the average value of ω for which the least number of iterations is required.

The condition for the convergence of the S.O.R. method is

$$|\phi_{i,j}^{(n+1)} - \phi_{i,j}^{(n)}| < \varepsilon \quad \text{for all } i, j, \tag{27}$$

where $\varepsilon = 5 \times 10^{-5}$, and also for the power method, the difference between successive estimates of the spectral radius $\rho(G)$ is chosen to be less than the value of ε. Four different mesh sizes are used which produce (10×10), (20×20), (40×40) and (60×60) networks. The results obtained from these experiments are recorded in Table 1, where n_A is the minimum number of iterations required for convergence, n_E is the estimated number of iterations, and ω_e and ω_b are as previously defined. Obviously, there is, as might be expected, little difference between the results achieved by the two orderings.

One fact that is not so obvious, however, is that the parallel R.B. ordering is not consistent. This is the result of the simultaneous evaluation of adjacent mesh values, since the $(n+1)$st iterates at the two points are evaluated using the nth iterative values of each other, whereas when evaluated sequentially, the second point to be evaluated would use the $(n+1)$st iterative value of the first point. Hence, the sequential ordering is not preserved. Clearly, in this case, the fact that the ordering is inconsistent, has no serious effect on the performance of the algorithm. One question that cannot be answered however until the algorithm is actually implemented, is, "will the 2 asynchronous processors cross over at different points during each iteration?" and if so "will it have a more serious effect on the algorithm's performance?"

The necessity of consistent ordering is an important question and from some of the following orderings it will be seen that the lack of consistency can be a serious problem. The main advantage is that the S.O.R. theory does not hold, which makes it impossible to estimate ω_b accurately.

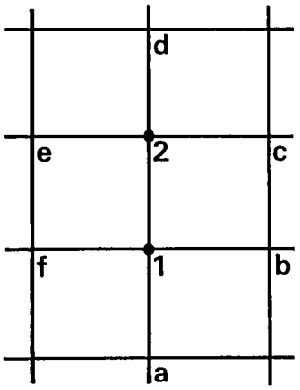
4. Explicit block iterative schemes

The number of iterations required for the convergence of the iterative process may be reduced by evaluating iterates at groups of mesh points by a direct method. This technique leads to new block iterative methods which we shall now discuss.

Table 1

Method	Mesh size	n_A	n_E	ω_e	ω_b
Red-black	10	15	14	1.495	1.490
point S.O.R.	20	29	30	1.717	1.717
	40	57	58	1.846	1.842
	60	82	76	1.890	1.877
Parallel	10	15	14	1.503	1.490
R.B.	20	32	30	1.732	1.718
point S.O.R.	40	58	58	1.848	1.842
	60	83	76	1.894	1.877

Consider, for instance, the following group of mesh points.



(28)

If equation (12) is applied to points 1 and 2 we obtain the formulae

$$4\phi_1 = \phi_a + \phi_b + \phi_2 + \phi_f \quad \text{and} \quad 4\phi_2 = \phi_1 + \phi_c + \phi_d + \phi_e \tag{29}$$

which may be rearranged to give

$$\phi_1 = \tfrac{1}{15}\left(4(\phi_a + \phi_b + \phi_f) + \phi_c + \phi_d + \phi_e\right)$$

and

$$\phi_2 = \tfrac{1}{15}\left(4(\phi_c + \phi_d + \phi_e) + \phi_a + \phi_b + \phi_f\right) \tag{30}$$

or in terms of i and j,

$$\phi_{i,j} = \tfrac{1}{15}\left(4(\phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}) + \phi_{i+1,j+1} + \phi_{i+2,j} + \phi_{i+1,j-1}\right)$$

and

$$\phi_{i+1,j} = \tfrac{1}{15}\left(4(\phi_{i+1,j+1} + \phi_{i+2,j} + \phi_{i+1,j-1}) + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}\right). \tag{31}$$

Clearly, these two equations are independent of each other and so may be evaluated simultaneously. Thus, by partitioning the system of meshes into (2×1) blocks we can evaluate the iterates at the points within each block simultaneously using two processors. The order in which the blocks are considered is important and so, remembering that the processors of an MIMD computer are not synchronous, we shall use the red-black ordering as in (22) except that each point represents a (2×1) block. Any consistent ordering of the blocks may, of course, be used but with the ordering defined in (13) for instance, we cannot be sure that all of the latest iterative values will be available when required.

So, using a red-black ordering of the blocks, the $(n+1)$st iterates of the S.O.R. iterative scheme will be defined by

$$\phi_{i,j}^{(n+1)} = \phi_{i,j}^{(n)} + \tfrac{1}{15}\omega\left(4r_{i,j}^{(n)} + r_{i+1,j}^{(n)} - 15\phi_{i,j}^{(n)}\right)$$

and (32)

$$\phi_{i+1,j}^{(n+1)} = \phi_{i+1,j}^{(n)} + \tfrac{1}{15}\omega\left(4r_{i+1,j}^{(n)} + r_{i,j}^{(n)} - 15\phi_{i+1,j}^{(n)}\right)$$

where

$$r_{i,j}^{(n)} = \left(\phi_{i-1,j}^{(n)} + \phi_{i,j+1}^{(n)} + \phi_{i,j-1}^{(n)}\right) \quad \text{and} \quad r_{i+1,j}^{(n)} = \left(\phi_{i+1,j+1}^{(n)} + \phi_{i+2,j}^{(n)} + \phi_{i+1,j-1}^{(n)}\right)$$

during the first pass and

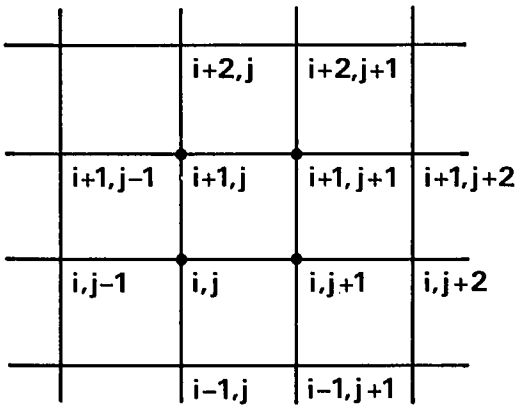
$$\phi_{i,j}^{(n+1)} = \phi_{i,j}^{(n)} + \tfrac{1}{15}\omega\left(4r_{i,j}^{(n+1)} + r_{i+1,j}^{(n+1)} - 15\phi_{i,j}^{(n)}\right)$$

and (33)

$$\phi_{i+1,j}^{(n+1)} = \phi_{i+1,j}^{(n)} + \tfrac{1}{15}\omega\left(4r_{i+1,j}^{(n+1)} + r_{i,j}^{(n+1)} - 15\phi_{i+1,j}^{(n)}\right)$$

during the second pass.

In order to use more than two processors, we can either solve for more than one block at a time, or, alternatively, partition the system of meshes into larger blocks; so consider the following (2×2) block of mesh points, as in Evans and Biggins [6].



(34)

Applying the same technique as was applied to the (2×1) block, we obtain the formulae,

$$\begin{aligned}
\phi_{i,j} &= \tfrac{1}{24}\left(2\left(\phi_{i-1,j+1} + \phi_{i,j+2} + \phi_{i+2,j} + \phi_{i+1,j-1}\right)\right. \\
&\quad \left. + 7\left(\phi_{i-1,j} + \phi_{i,j-1}\right) + \phi_{i+1,j+2} + \phi_{i+2,j+1}\right) \\
&= \tfrac{1}{24} r_{i,j}, \\
\phi_{i,j+1} &= \tfrac{1}{24}\left(2\left(\phi_{i-1,j} + \phi_{i+1,j+2} + \phi_{i+2,j+1} + \phi_{i,j-1}\right)\right. \\
&\quad \left. + 7\left(\phi_{i-1,j+1} + \phi_{i,j+2}\right) + \phi_{i+2,j} + \phi_{i+1,j-1}\right) \\
&= \tfrac{1}{24} r_{i,j+1}, \\
\phi_{i+1,j+1} &= \tfrac{1}{24}\left(2\left(\phi_{i-1,j+1} + \phi_{i,j+2} + \phi_{i+2,j} + \phi_{i+1,j-1}\right)\right. \\
&\quad \left. + 7\left(\phi_{i+1,j+2} + \phi_{i+2,j+1}\right) + \phi_{i-1,j} + \phi_{i,j-1}\right) \\
&= \tfrac{1}{24} r_{i+1,j+1},
\end{aligned}$$

and

$$\begin{aligned}
\phi_{i+1,j} &= \tfrac{1}{24}\left(2\left(\phi_{i-1,j} + \phi_{i+1,j+2} + \phi_{i+2,j+1} + \phi_{i,j-1}\right)\right. \\
&\quad \left. + 7\left(\phi_{i+2,j} + \phi_{i+1,j-1}\right) + \phi_{i-1,j+1} + \phi_{i,j+2}\right) \\
&= \tfrac{1}{24} r_{i+1,j}.
\end{aligned} \tag{35}$$

Again, using a red-black ordering of the blocks, the $(n+1)$st iterates during the first pass are defined by

$$\phi_{i,j}^{(n+1)} = \phi_{i,j}^{(n)} + \tfrac{1}{24}\omega\left(r_{i,j}^{(n)} - 24\phi_{i,j}^{(n)}\right),$$
$$\phi_{i,j+1}^{(n+1)} = \phi_{i,j+1}^{(n)} + \tfrac{1}{24}\omega\left(r_{i,j+1}^{(n)} - 24\phi_{i,j+1}^{(n)}\right),$$
$$\phi_{i+1,j+1}^{(n+1)} = \phi_{i+1,j+1}^{(n)} + \tfrac{1}{24}\omega\left(r_{i+1,j+1}^{(n)} - 24\phi_{i+1,j+1}^{(n)}\right),$$

and

$$\phi_{i+1,j}^{(n+1)} = \phi_{i+1,j}^{(n)} + \tfrac{1}{24}\omega\left(r_{i+1,j}^{(n)} - 24\phi_{i+1,j}^{(n)}\right), \tag{36a}$$

and during the second pass by

$$\phi_{i,j}^{(n+1)} = \phi_{i,j}^{(n)} + \tfrac{1}{24}\omega\left(r_{i,j}^{(n+1)} - 24\phi_{i,j}^{(n)}\right),$$
$$\phi_{i,j+1}^{(n+1)} = \phi_{i,j+1}^{(n)} + \tfrac{1}{24}\omega\left(r_{i,j+1}^{(n+1)} - 24\phi_{i,j+1}^{(n)}\right),$$
$$\phi_{i+1,j+1}^{(n+1)} = \phi_{i+1,j+1}^{(n)} + \tfrac{1}{24}\omega\left(r_{i+1,j+1}^{(n+1)} - 24\phi_{i+1,j+1}^{(n)}\right),$$

and

$$\phi_{i+1,j}^{(n+1)} = \phi_{i+1,j}^{(n)} + \tfrac{1}{24}\omega\left(r_{i+1,j}^{(n+1)} - 24\phi_{i+1,j}^{(n)}\right). \tag{36b}$$

Obviously, with this scheme, we may use 4 processors simultaneously.

Considering now the number of parallel operations per iteration, we have for the (2×1) block scheme,

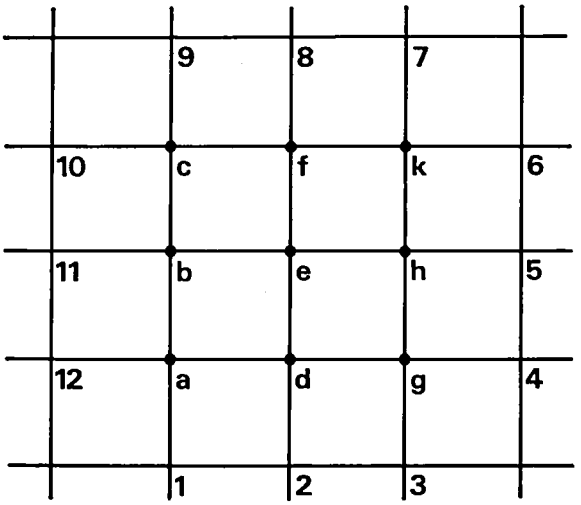
$$\left(2m^2 \text{ multiplications} + \tfrac{7}{2}m^2 \text{ additions}\right), \tag{37}$$

when using 2 processors and for the (2×2) block scheme,

$$\left(\tfrac{5}{4}m^2 \text{ multiplications} + \tfrac{9}{4}m^2 \text{ additions}\right), \tag{38}$$

when using 4 processors.

The next size of block to be considered is the (3×3) block,



(39)

Applying the same technique as before, we obtain the formulae,

$$\phi_a = \tfrac{1}{224}(67(\phi_1+\phi_{12})+22(\phi_2+\phi_{11}) + 7(\phi_3+\phi_4+\phi_9+\phi_{10})+6(\phi_5+\phi_8)+3(\phi_6+\phi_7)),$$

$$\phi_b = \tfrac{1}{112}(37\phi_{11}+11(\phi_1+\phi_9+\phi_{10}+\phi_{12})+7(\phi_2+\phi_8)+5\phi_5+3(\phi_3+\phi_4+\phi_6+\phi_7)),$$

$$\phi_c = \tfrac{1}{224}(67(\phi_9+\phi_{10})+22(\phi_8+\phi_{11}) + 7(\phi_1+\phi_6+\phi_7+\phi_{12})+6(\phi_2+\phi_5)+3(\phi_3+\phi_4)),$$

$$\phi_d = \tfrac{1}{112}(37\phi_2+11(\phi_1+\phi_3+\phi_4+\phi_{12})+7(\phi_5+\phi_{11})+5\phi_8+3(\phi_6+\phi_7+\phi_9+\phi_{10})),$$

$$\phi_e = \tfrac{1}{16}(2(\phi_2+\phi_5+\phi_8+\phi_{11})+\phi_1+\phi_3+\phi_4+\phi_6+\phi_7+\phi_9+\phi_{10}+\phi_{12})$$

$$\phi_f = \tfrac{1}{112}(37\phi_8+11(\phi_6+\phi_7+\phi_9+\phi_{10})+7(\phi_5+\phi_{11})+5\phi_2+3(\phi_1+\phi_3+\phi_4+\phi_{12})),$$

$$\phi_g = \tfrac{1}{224}(67(\phi_3+\phi_4)+22(\phi_2+\phi_5) + 7(\phi_1+\phi_6+\phi_7+\phi_{12})+6(\phi_8+\phi_{11})+3(\phi_9+\phi_{10})),$$

$$\phi_h = \tfrac{1}{112}(37\phi_5+11(\phi_3+\phi_4+\phi_6+\phi_7)+7(\phi_2+\phi_8)+5\phi_{11}+3(\phi_1+\phi_9+\phi_{10}+\phi_{12})),$$

and

$$\phi_k = \tfrac{1}{224}(67(\phi_6+\phi_7)+22(\phi_5+\phi_8) + 7(\phi_3+\phi_4+\phi_9+\phi_{10})+6(\phi_2+\phi_{11})+3(\phi_1+\phi_{12})), \tag{40}$$

from which it is not difficult to produce the corresponding S.O.R. formulae. Thus, by evaluating the iterative values at all of the mesh points within a block simultaneously, we can use 9 processors. Again, it is preferable to employ a red-black ordering of the blocks, and for this scheme m must be divisible by 3.

An unfortunate property of this block size is that the equations (40) are not all of the same form and so the rates at which each of the processors traverse the system of meshes will not be the same. However, by considering the processor that has the most work to do, the number of parallel operations will be

$$\left(\tfrac{8}{9}m^2 \text{ multiplications} + \tfrac{13}{9}m^2 \text{ additions}\right) \text{ per iteration.} \tag{41}$$

If we compare the number of parallel operations per iteration of the three block S.O.R. schemes considered so far, it can be seen that, as might be expected, this quantity decreases as

Table 2

Method	Mesh size	n_A	n_E	ω_e	ω_b
(2×1) Block	10	13	13	1.449	1.439
S.O.R.	20	26	26	1.681	1.681
	40	50	52	1.822	1.824
	60	73	70	1.874	1.867
(2×2) Block	10	11	10	1.371	1.365
S.O.R.	20	22	22	1.617	1.625
	40	42	43	1.784	1.793
	60	61	61	1.846	1.850
(3×3) Block	11	10	10	1.345	1.333
S.O.R.	20	18	18	1.561	1.563
	41	36	37	1.749	1.760
	62	52	54	1.826	1.830

the block size (and therefore the number of processors) is increased. However, going from the (2×1) block to the (2×2) block, for instance, does not halve the number of operations and so it is important to compare the respective rates of convergence. For this purpose the experiments performed using the point iterative schemes were repeated using the block schemes, the results of which are contained in Table 2. The headings of Table 2 are the same as those of Table 1. The differences in the mesh sizes for the (3×3) block scheme are to allow for the fact that m (the square root of the total number of internal mesh points) must be divisible by 3.

As expected, by increasing the block size, the number of iterations required for convergence of the iterative schemes is decreased. The results contained in Table 2 can be combined with the number of operations per iteration required by each method given in (37), (38) and (41), to give the total number of parallel operations required by each of the block S.O.R. methods, and are recorded in Table 3.

Clearly, by increasing the block size from (2×1) to (2×2), we see that the number of parallel operations is approximately halved and so would be justified if sufficient processors are available. The effect of increasing the block size to (3×3) is not quite so successful but still impressive. However, it must be remembered that the equations generated by the (3×3) block are not identical in form and also the parallel system overheads will be considerably more for 9 processors than for 2 or 4 processors.

Finally, attention will now be concentrated on the 4-point block method as a suitable parallel strategy to recommend for the iterative solution of this class of problems, on an MIMD asynchronous multiprocessor, since it satisfies the following desirable properties, i.e. it is explicit, possesses an improved convergence rate over the point method (Evans and Biggins [6]) and requires only a moderately small number of processors for its implementation.

5. The performance analysis of parallel algorithms

It is well-known that different algorithmic designs produce different timing results and speed-up ratios on parallel computers depending on whether the implementation is done synchronously or asynchronously. In both cases the overhead measurements should be borne in mind when more than one processor is cooperating.

In fact, parallel computing requires three resources: multiple processors, communication for shared data, and synchronisation to ensure any necessary time ordering. Thus, parallel programs always require more than one processor, and there has to be some communication between the processors even if it is only as much as that required to start processing in the first instant.

The main feature in the analysis of the demand and supply of resources is that several

Table 3

Mesh size		10		20		40		60	
Method	P	M	A	M	A	M	A	M	A
(2×1) Block S.O.R.	2	$26m^2$	$\frac{91}{2}m^2$	$52m^2$	$91m^2$	$100m^2$	$175m^2$	$146m^2$	$\frac{511}{2}m^2$
(2×2) Block S.O.R.	4	$\frac{55}{4}m^2$	$\frac{99}{4}m^2$	$\frac{55}{2}m^2$	$\frac{99}{2}m^2$	$\frac{105}{2}m^2$	$\frac{189}{2}m^2$	$\frac{305}{4}m^2$	$\frac{549}{4}m^2$
(3×3) Block [a] S.O.R.	9	$\frac{80}{9}m^2$	$\frac{130}{9}m^2$	$16m^2$	$26m^2$	$32m^2$	$52m^2$	$\frac{416}{9}m^2$	$\frac{676}{9}m^2$

P = no. of processors, M = multiplications and A = additions

[a] mesh sizes are 11, 20, 41 and 62 as in Table 2.

demands may compete for the supply of a shared resource: i.e., processors, shared data structure or a memory block. This competition or contention has three consequences:

(1) Since the shared resource has a limited availability it can satisfy only a finite number of demands in a finite time and this can limit the maximum performance of the program.

(2) A mechanism is required to arbitrate between requests and keep all but one waiting, and this mechanism itself then imposes an overhead on the resource access even if there is no competition.

(3) When requests contend, then all but one request will have to wait.

The second and the third factors degrade the performance and these factors will be called respectively the *static* and *dynamic* costs of a shared resource access (Baudet [4]).

Barlow et al. [2] have discussed the performance analysis of the algorithm on an asynchronous type machine and classified the sources of overheads as two types:

(1) The *static* overhead which corresponds to the design of software and hardware. This includes the subdivision of the tasks, the allocation of these tasks to the available processors, and the checking by hardware and software for contention on accesses to the data base.

(2) The *dynamic* overhead which corresponds to the interference between two or more subtasks running on different processors causing one or more of the processors to wait.

The performance of a multiprocessor can be expressed either in the form of a speed-up factor $S_P = T(1)/T(P)$, or in terms of the time wasted. However, the wasted time must be equal to the sum of the static and dynamic overheads. Let $W = P * T(P) - T(1)$, be the wasted time which is the sum of times taken by the P processors to complete their subtasks *minus* the time taken on a uniprocessor. It is clear from this that either all processors complete processing together or some processors take longer than others. Thus,

$$T(P) \geqslant \frac{T(1) + W}{P}.$$

It follows that,

$$S_P \leqslant \frac{PT(1)}{T(1) + W}.$$

Therefore, the maximum possible speed-up factor for a given algorithm can be determined by assuming that the dynamic overheads are zero. The dynamic overheads are zero only if every request for a resource occurs when that resource is not being used. This is true only if the demands for a resource are less than the supply of that resource.

We discuss now the three factors i.e., the processors, shared data, and the critical sections and their effect on a parallel system.

For the processors, the software measures the number of subtasks allocated to the processors and it counts the cycles that a processor is idle because there are no ready sub-tasks which are available to run.

For the shared data, this can be measured by counting the number of accesses to a shared data by going through the user's program.

Whilst for the critical regions, the software can measure the number of accesses made by a processor to a critical region and the number of cycles a processor remains idle because this resource is accessed by other processors (Evans [5]).

For the performance analysis of the algorithms discussed in this paper we present results based on measurements obtained from actual experiments carried out on the NEPTUNE system, a 4 processor partial MIMD system at Loughborough University which indicates the total computational complexity performed by each path and the calculation of how many parallel paths and critical sections a single processor had made. For our measurements the resource times of the NEPTUNE system are required and these are obtained from Barlow et al.

Table 4
Resource demands of algorithms (the standard point method and the 4-point block methods)

Program	Processors (P)		Shared data		Parallel path		Mutual exclusion	
	Numbers	Speed-up	Access rate	Overhead amount	Access rate	Overhead amount	Access rate	Overhead amount
Point method	$P \leqslant m^2$	$O(P)$	7:11 flops	0.06%	–	–	–	–
4 point block S.O.R.	$P \leqslant \frac{1}{4}m^2$	$O(P)$	12:27 flops	0.04%	–	–	1:53 flops	1%

Table 5
Performance measurements of algorithms on the NEPTUNE system

Program	Idle time	Speed-up			ω	Parallel path		Critical regions	
		2	3	4		Static	Contention	Static	Contention
Point method	–	1.9	–	3.8	1.0	0.002%	0.01%	–	–
		1.9	–	3.7	1.70	0.01%	0.03%	–	–
4-point block	–	1.96	2.92	3.84	1.0	0.002%	0.002%	1%	0.06%
S.O.R.		1.96	2.92	3.83	1.61	0.002%	0.002%	1%	0.05%

Table 6
The asynchronous 4-point block method (asynchronous list of blocks)

Mesh size ($N \times N$)	P	ω	Time (seconds)	Number of iterations	Speed-up
(16×16)	1	1.0	271.830	146	1
	2		138.980	147	1.96
	3		93.100	148	2.92
	4		70.820	150	3.84
	1	1.61	52.120	28	1
	2		27.720	29	1.95
	3		17.880	30	2.92
	4		13.620	31	3.83

[3]. Thus, Table 4 illustrates the resource demands of the standard point method and the 4-point block method both implemented asynchronously as well as the mean rate of accesses to the shared data, parallel path scheduling and critical sections. Finally, estimates of the potential speed-up from using P processors are given, where m represents the number of rows in the mesh to be solved and a flop represents a floating-point operation. In addition Table 5 illustrates the experimental results obtained when both algorithms were run on the NEPTUNE system. The parallel paths and critical sections measurements are taken for the case of 4 processors and the parallel control access overheads and the shared data access overheads are taken for the case $P = 1$, while m was 16 in all experiments (Evans and Yousif [7]).

From the basic concepts of the 4-point block iterative method, we implement the method asynchronously in parallel to solve the model problem, i.e. the two-dimensional Dirichlet problem using the strategy of distributing the blocks of 4 points onto the processors as given in Barlow and Evans [1].

The row and column indices of the blocks of 4-points are stored in a shared list of two dimensions. The arrangement of the blocks in the list is represented by the red-black ordering. A shared index to the column of the list is used. Since the number of the rows are stored in the first row of the list and the columns in the second row, therefore, updating the shared index by a critical section will allow only one processor to evaluate the block which corresponds to that index. For the convergence test, a list of the number of flags that is equal to the number of blocks in the linear system of equations is used so that the flag for the block under consideration is set to 1 if any single component of the block has not converged, otherwise the flag is set to 0. In order to iterate again, each processor tests all of the flags of all blocks and if it finds any of them set to 1 then it iterates again choosing any available block. Otherwise, a global flag is set indicating that the convergence has been achieved for all components in the mesh. Since setting the global flag is performed by one processor which terminates its path afterwards, the other processors will only terminate when they attempt to uptake another block.

The algorithms were run for $\omega = 1.0$, i.e., similar to the Gauss–Seidel method and $\omega = \omega_{opt}$, i.e. similar to the S.O.R. method and the timing results listed in Table 6.

Finally, from the results presented here and those reported in Yousif [9] concerning a large number of different implementations it can be confirmed that the asynchronous implementation of the 4-point explicit block method when the blocks of 4 points are arranged in a red-black ordering in a list achieves the best speed-up factor which is almost linear with the number of processors available. This is because the processors were fully occupied and busy doing useful work most of the time.

References

[1] R.H. Barlow and D.J. Evans, Parallel algorithms for the iterative solution to linear systems, *Comp. J.* **25** (1982) 56–60.
[2] R.H. Barlow, D.J. Evans, I.A. Newman, J. Shanehchi and M. Woodward, Performance analysis of parallel algorithms on asynchronous parallel computers, Comp. Stud. Rep., Loughborough University (1982).
[3] R.H. Barlow, D.J. Evans, I.A. Newman and M.C. Woodward, A guide to using the Neptune parallel processing system, Comp. Stud. Rep., Loughborough University (1981).
[4] G.M. Baudet, Asynchronous iterative methods for multiprocessors, *J. ACM* **25** (1978) 226–244.
[5] D.J. Evans, *Parallel Processing Systems* (Cambridge University Press, London, 1982).
[6] D.J. Evans and M. Biggins, The solution of elliptic partial differential equations by a new block over-relaxation technique, *Int. J. Comp. Math.* **10** (1982) 269–282.
[7] D.J. Evans and N.Y. Yousif, Asynchronous and synchronous iterative methods for solving linear equations, to appear.
[8] D.M. Young, *Iterative Solution of Large Linear Systems* (Academic Press, New York, 1971).
[9] N.Y. Yousif, Parallel algorithms for asynchronous multiprocessors, Ph.D. Thesis, Loughborough University (1983).