SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 4, pp. 841-859, July 1992

PARALLEL METHODS FOR SOLVING NONLINEAR BLOCK BORDERED SYSTEMS OF EQUATIONS*

XIAODONG ZHANG†, RICHARD H. BYRD‡, AND ROBERT B. SCHNABEL‡

Abstract. A group of parallel algorithms, and their implementation for solving a special class of nonlinear equations, are discussed. The type of sparsity occurring in these problems, which arise in VLSI design, structural engineering, and many other areas, is called a *block bordered* structure. The explicit method and several implicit methods are described, and the new *corrected implicit method* for solving block bordered nonlinear problems is presented. The relationship between the two types of methods is analyzed, and some computational comparisons are performed. Several variations and globally convergent modifications of the implicit method are also described. Parallel implementations of these algorithms for solving block bordered nonlinear equations are described, and experimental results on the Intel hypercube that show the effectiveness of the parallel implicit algorithms are presented. These experiments include a fairly large circuit simulation that leads to a multilevel block bordered system of nonlinear equations.

Key words. systems of nonlinear equations, parallel computation

AMS(MOS) subject classifications. 65H10, 65W05

1. Introduction.

1.1. Definition of block bordered nonlinear problems. In this paper we present a group of parallel algorithms and their implementations for solving a special class of nonlinear equations, instances of which occur in VLSI design, structural engineering, and many other areas. The class of sparsity occurring in these problems is called a block bordered structure. In such a problem the general system of n nonlinear equations in n unknowns may be grouped into $q+1$ subvectors $x_1, \cdots, x_{q+1}$ and $f_1, \cdots, f_{q+1}$ such that the nonlinear system of equations has the form

$$\begin{aligned} &f_i(x_i, x_{q+1}) = 0, \qquad i = 1, \cdots, q \\ &f_{q+1}(x_1, \cdots, x_{q+1}) = 0 \end{aligned} \tag{1.1}$$

where

$$x_i \in R^{n_i},$$

$$f_i \in R^{n_i}, \qquad i = 1, \cdots, q+1,$$

$$\sum_{i=1}^{q+1} n_i = n.$$

The block bordered Jacobian matrix of (1.1) is

$$\begin{pmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_q & B_q \\ C_1 & C_2 & \cdots & C_q & P \end{pmatrix}, \tag{1.2}$$

* Received by the editors January 16, 1990; accepted for publication (in revised form) May 14, 1991. This research was partially supported by Air Force Office of Scientific Research grant AFOSR-85-0251.

† Division of Mathematics and Computer Science, University of Texas, San Antonio, Texas 78285-0664 (zhang@ringer.cs.utsa.edu).

‡ Department of Computer Science, University of Colorado, Boulder, Colorado 80309-0430 (richard@cs.colorado.edu and bobby@cs.colorado.edu).

where

$$A_i = \frac{\partial f_i}{\partial x_i} \in R^{n_i \times n_i}, \quad i = 1, \cdots, q,$$

$$B_i = \frac{\partial f_i}{\partial x_{q+1}} \in R^{n_i \times n_{q+1}}, \qquad i = 1, \cdots, q,$$

$$C_i = \frac{\partial f_{q+1}}{\partial x_i} \in R^{n_{q+1} \times n_i}, \qquad i = 1, \cdots, q,$$

$$P = \frac{\partial f_{q+1}}{\partial x_{q+1}} \in R^{n_{q+1} \times n_{q+1}}.$$

Newton's method is the fundamental approach for solving a general nonlinear system of equations. Several parallel algorithms for solving general systems of nonlinear equations that are based upon Newton's method have been developed and implemented on various parallel computers. These parallel algorithms consist mainly of solving the linear Jacobian system in parallel. Many parallel algorithms have been developed for solving linear systems, such as as parallel factorizations, parallel SOR methods, parallel red-black methods, parallel multicolor, and others (see e.g., Ortega and Voigt [1985], O'Leary and White [1985], White [1986], Fontecilla [1987], and Coleman and Li [1990]).

In the case of very large nonlinear problems we cannot expect a single parallel algorithm to efficiently handle all the instances of the system of nonlinear equations problem, but rather the algorithm must take into account the sparsity structure and other special characteristics of the problem. In fact, many nonlinear problems arising in applications have their own special sparsity structure. Parallel algorithms taking advantage of this special structure may be much more efficient than the algorithms ignoring the special structure. This paper is an instance of developing special algorithms for a special, important structure.

1.2. Background on block bordered problems. Block bordered problems of the form (1.1) arise in many areas of science and engineering, and a few algorithms have been developed to efficiently solve linear block bordered systems of equations. In applications such as structural engineering, large spatial models may be divided into q regions such that each region only interacts directly with neighboring regions. The variables x_i for each region are chosen so that the model can determine their values, given the values of the linking variables (the x_{q+1}) at the boundaries of the regions. The linking variables are tied together by a $(q+1)$st set of equations representing the interactions between the regions. Thus the equilibrium equations for such a model will be of the form (1.1). In addition, the Jacobian matrix is symmetric. These problems, and parallel algorithms for solving the linear block bordered systems that arise from them, are discussed in Farhat and Wilson [1986] and Nour-Omid and Park [1986].

Mu and Rice [1989] study parallel Gaussian elimination for the block bordered matrices arising from the discretization of partial differential equations (PDEs). Christara and Houstis [1988], [1989] implement a domain decomposition spline collocation method and a preconditioned conjugate gradient (PCG) method for this linear block bordered system on both NCUBE/7 and Sequent multiprocessors.

All the work described above concerns parallel methods for solving linear block bordered equations. Our research is to develop, implement, and analyze parallel methods for solving nonlinear block bordered problems. To our knowledge, no one has done similar work.

Block bordered equations also arise in VLSI circuit design, where parts of the circuits may be divided into regions. The concept of macromodeling the circuit is to decompose the circuit into subcircuits and to analyze the subcircuits separately (see Rabbat, Sangiovanni-Vincentelli, and Hsieh [1979] and Rabbat and Sangiovanni-Vincentelli [1980]). Macromodeling of the circuit results in a system of nonlinear equations of form (1.1). A_i and B_i $(i=1,\cdots,q)$ in the Jacobian matrix are usually used to represent internal and input-output variables in each of the q independent subcircuits. The variables represent voltages and currents. The bottom block f_{q+1} represents the voltages or currents between subcircuits. Since each voltage or current is used only in one block of equations f_i plus possibly the bottom block f_{q+1}, the nonzero columns of the B_i's (and A_i) are disjoint, meaning that the matrices $B_i,\cdots B_q$ in (1.2) also follow a block diagonal pattern. In addition, since f_{q+1} describes the point-to-point connections of voltage and current, it is a linear function. The size of the function f_{q+1} depends on the number of connections among the subcircuits.

We have studied parallel methods for solving block bordered nonlinear equations extensively from both theoretical and practical viewpoints. Section 2 considers the explicit method and several implicit methods for solving block bordered nonlinear equations, and presents a new implicit approach—the corrected implicit method. A mathematical analysis of the two types of methods and a computational comparison in a sequential context is made. Section 3 briefly discusses techniques used to make these methods globally convergent. In § 4, we give a group of parallel algorithms for solving block bordered nonlinear systems of form (1.1), which may be implemented and distributed on both shared and distributed memory multiprocessors. The implementations and experimental results of these algorithms on the Intel hypercube, a distributed memory multiprocessor, are presented in § 5. Finally, our conclusions and some future research directions are summarized in § 6,

2. Explicit and implicit methods.

2.1. Introduction. There are two basic ways in which Newton's method can be applied to the nonlinear block bordered system of equations (1.1), which we refer to as the explicit and implicit approaches. The explicit approach is to simply apply Newton's method to (1.1). This involves iteratively solving the linear system

$$J(X^k)\Delta X^k=-F(X^k),\qquad k=0,1,\cdots \tag{2.1}$$

for ΔX^k, where $J(X^k)$ is the Jacobian of F, which has the block bordered structure (1.2), and $X=(x_1,\cdots,x_q,x_{q+1})$.

The pure implicit approach is to use each of the q systems of nonlinear equations

$$f_i(x_i,x_{q+1})=0,\qquad i=1,\cdots,q \tag{2.2}$$

to solve for x_i, given a fixed value of x_{q+1}. This means that each of the x_i is implicitly given by a function of x_{q+1}. The whole problem (2.2) is then equivalent to solving

$$f_{q+1}(x_1(x_{q+1}),\cdots,x_q(x_{q+1}),x_{q+1})=0. \tag{2.3}$$

The Jacobian of this system is given by

$$\hat{J}=\frac{\partial f_{q+1}}{\partial x_{q+1}}-\sum_{i=1}^{q}\frac{\partial f_{q+1}}{\partial x_i}\left(\frac{\partial f_i}{\partial x_i}\right)^{-1}\frac{\partial f_i}{\partial x_{q+1}} \tag{2.4}$$

or

$$\hat{J}=P-\sum_{i=1}^{q}C_iA_i^{-1}B_i \tag{2.5}$$

and we may solve (2.3) by Newton's method. We will be considering practical variants of the implicit method that do not calculate $x_i(x_{q+1})$ exactly. We assume here and for the rest of § 2 that the matrices A_i and $\hat{J}$ are nonsingular. The situation where this does not hold will be discussed in § 3.

In this section we describe the explicit and implicit methods and their relations to each other, introduce the new corrected implicit method, and give some simple experimental results using these methods on a sequential computer.

2.2. Algorithms and analysis for the explicit and implicit methods. Newton's method applied to (1.1) in the explicit method consists of solving the following linear equations at iteration k $(k=0,1,\cdots)$: from $f_i(x)=0$, $i=1,\cdots,q$,

$$A_i\Delta x_i^k + B_i\Delta x_{q+1}^k + f_i(x_i^k, x_{q+1}^k) = 0, \tag{2.6}$$

and from $f_{q+1}(x_1^k,\cdots,x_q^k,x_{q+1}^k)=0$,

$$\sum_{i=1}^{q} C_i\Delta x_i^k + P\Delta x_{q+1}^k + f_{q+1}(x_1^k,\cdots,x_q^k,x_{q+1}^k) = 0. \tag{2.7}$$

(For simplicity, we omit superscripts k on A_i, B_i, C_i, and P, but note that at least the A_i's and B_i's can change at each iteration.) Solving for Δx_i^k in (2.6) and substituting into (2.7), we obtain

$$\hat{J}\Delta x_{q+1}^k = -f_{q+1}(x_1^k,\cdots,x_q^k,x_{q+1}^k) + \sum_{i=1}^{q} C_iA_i^{-1}f_i(x_i^k,x_{q+1}^k) \tag{2.8}$$

where $\hat{J}$ is given by (2.5). So

$$x_{q+1}^{k+1} = x_{q+1}^k + \Delta x_{q+1}^k \tag{2.9}$$

can be determined from (2.8), and

$$x_i^{k+1} = x_i^k + \Delta x_i^k, \qquad i=1,\cdots,q \tag{2.10}$$

can be determined from (2.6).

In the pure implicit method, Newton's method is applied to (2.3) and gives

$$\hat{J}\Delta x_{q+1}^k + f_{q+1}(x_1(x_{q+1}^k),\cdots,x_q(x_{q+1}^k),x_{q+1}^k) = 0, \tag{2.11}$$

where $x_i(x_{q+1}^k)$ $(i=1,\cdots,q)$ is implicitly determined by solving the nonlinear system

$$f_i(x_i, x_{q+1}^k) = 0 \tag{2.12}$$

for x_i. To turn this into a practical computational procedure, we use a second (or inner) Newton process on (2.12) to calculate $x_i(x_{q+1}^k)$, which solves (2.12) approximately. For each $i=1,\cdots,q$, this yields the inner iteration

$$\hat{A}_i\Delta x_i^{k,j-1} + f_i(x_i^{k,j-1}, x_{q+1}^k) = 0, \quad i=1,\cdots,q, \quad j=1,2,\cdots,I_{in}. \tag{2.13}$$

Here $x_i^{k,0}=x_i^k$, I_{in} is the number of inner iterations, and $\hat{A}_i=A_i$ if it is only evaluated once at the beginning of each outer iteration; or, it may be evaluated up to I_{in} times. At the end of each inner iteration, we set

$$x_i^{k,j} = x_i^{k,j-1} + \Delta x_i^{k,j-1}, \qquad i=1,\cdots,q. \tag{2.14}$$

When we exit the inner iterations, we set

$$x_i(x_{q+1}^k) = x_i^{k+1} = x_i^{k,j}. \tag{2.15}$$

Then x_{q+1} is determined from (2.11) and

$$x_{q+1}^{k+1} = x_{q+1}^k + \Delta x_{q+1}^k. \tag{2.16}$$

The implicit Newton iteration (2.11) is also considered by Rabbat, Sangiovanni-Vincentelli, and Hsieh [1979], but instead of focusing on the number of inner iterations, they assume that (2.12) is solved to some tolerance. By considering a method with a fixed number of inner iterations, we are led to the following theorems, which show a close relationship between the explicit method and the implicit methods just described. We are also led to a new method—the corrected implicit method.

THEOREM 1. *If f_{q+1} is linear, the matrices A_i and $\hat{J}$ are nonsingular, and only one inner Newton iteration is applied to solve for $x_i^{k,j}$ $(i=1,\cdots,q)$ in the implicit method, i.e., $I_{in}=1$, then for a fixed k, the steps Δx_{q+1}^k are identical in both methods.*

Proof. In this case $x_i^{k+1}=x_i^k-A_i^{-1}f_i(x_i^k,x_{q+1}^k)$ and $f_{q+1}(x_1^{k+1},\cdots,x_q^{k+1},x_{q+1}^k)=f_{q+1}(x_1^k,\cdots,x_q^k,x_{q+1}^k)-\sum_{i=1}^q C_iA_i^{-1}f_i(x_i^k,x_{q+1}^k)$. Substituting this into (2.11) gives

$$\hat{J}\Delta x_{q+1}^k=-f_{q+1}(x_1^k,\cdots,x_q^k,x_{q+1}^k)+\sum_{i=0}^{q} C_iA_i^{-1}f_i(x_i^k,x_{q+1}^k), \tag{2.17}$$

which is identical to the explicit formula (2.8). □

The corrected implicit method. In the implicit method described above, the steps Δx_i for $i\leqq q$ do not involve any information about f_{q+1} or Δx_{q+1}, and are not the same as in the explicit method. If the value of $x_i^{k+1}(i\leqq q)$ calculated by the implicit method is corrected after each iteration to account for the change in x_{q+1}, however, the implicit method can be made closer to the explicit method and quadratically convergent. The problem may be defined to find a correction term δ such that

$$f_i(x_i^{k+1}+\delta,\ x_{q+1}^{k+1})\approx 0 \tag{2.18}$$

or

$$f_i(x_i^{k+1}+\delta,\ x_{q+1}^k+\Delta x_{q+1}^k)\approx 0. \tag{2.19}$$

Making a linear approximation to f_i in (2.19) yields the condition

$$f_i(x_i^{k+1},x_{q+1}^k)+A_i\delta+B_i\Delta x_{q+1}^k=0. \tag{2.20}$$

The correction term δ obtained from (2.20) would then be

$$\delta=-A_i^{-1}[f_i(x_i^{k+1},x_{q+1}^k)+B_i\Delta x_{q+1}^k]. \tag{2.21}$$

However, after I_{in} inner iterations of solving for $x_i^{k+1}, f_i(x_i^{k+1},x_{q+1}^k)\approx 0$. Thus we make a further approximation giving the correction term

$$\delta_i=-A_i^{-1}B_i\Delta x_{q+1}^k. \tag{2.22}$$

We call the new implicit method with this correction the *corrected implicit method.* The cost of the correction term (2.22) is small since the matrices $A_i^{-1}B_i$ $(i=1,\cdots,q)$ have been calculated already and in a parallel implementation, the matrix-vector products can be parallelized fully.

We can now see that when f_{q+1} is linear, the explicit method is a special case of the corrected implicit method.

THEOREM 2. *If f_{q+1} is linear, the matrices A_i and $\hat{J}$ are nonsingular, and only one Newton iteration is applied to solve for $x_i^{k,j}$ $(i=1,\cdots,q)$ in the implicit method, i.e., $I_{in}=1$, and the system is corrected by adding $-A_i^{-1}B_i\Delta x_{q+1}$ to $x_i^{k+1}(i\leqq q)$ after each iteration, then the explicit method and implicit method are identical.*

Proof. From Theorem 1, Δx_{q+1}^k is identical for the two methods. Combining (2.14) with $j=1$ and (2.22) gives

$$x_i^{k+1}=x_i^k-A_i^{-1}[f_i(x_i^k,,x_{q+1}^k)-B_i\Delta x_{q+1}^k] \tag{2.23}$$

which is identical to (2.6) in the explicit method. □

This equivalence, together with the standard convergence analysis for Newton's method, gives the following convergence result.

COROLLARY. *Suppose $J(X)$ is continuously differentiable near a solution X^*, the matrices $A_i(x^*)$ and $J(\hat{X}^*)$ are nonsingular, and f_{q+1} is linear. Then the corrected implicit method with $I_{in}=1$ inner iterations per outer iteration is locally quadratically convergent to the solution.*

Quadratic convergence can also be shown for $I_{in}>1$ and when f_{q+1} is nonlinear. The proof is given in Zhang [1989], and is based on the fact that the extra inner iterations tend to move X closer to the solution, and the nonlinearity in f_{q+1} at most adds a term of order $\|X^k - X^*\|^2$ to the error at X^{k+1}.

2.3. Some experiments on a sequential processor. The previous subsection shows that a variant of the implicit method is equivalent to the explicit method, but does not indicate why the implicit method might be preferred. The main reason is that, by using more than one inner iteration per outer iteration in the implicit method, the number of outer iterations can be reduced substantially, which is advantageous, especially for parallel computation. In this subsection we give a first indication of the sort of computational behavior that we have found.

We initially tested the methods discussed in this section on several artificial problems. Here we report results on a simple 20×20 nonlinear block bordered system of quadratic equations that has four 4×4 blocks, $A_1, \cdots, A_4$, and a 4×4 bottom block P, with f_{q+1} linear. In all cases, the starting value of x was close to the solution, and no global strategy (e.g., line search) was used. All these experiments were run on a Pyramid P90 computer.

First we compare the performance of the three methods when only one inner iteration ($I_{in}=1$) is used in the uncorrected implicit and corrected implicit methods (Table 2.1). The explicit method and the corrected implicit method with $I_{in}=1$ are identical in this case (see Theorem 2). Thus the same number of iterations is required to converge to the solutions. The computing times are slightly different since the implementations of the two methods are different. The uncorrected implicit method converges more slowly than the other two methods, which is reasonable since the correction step is needed to make it quadratically convergent.

TABLE 2.1
Experiments with the three methods.

($I_{in}=1$) Outer iterations (seconds)		
Explicit	Implicit	Corrected implicit
13 (0.44)	14 (0.40)	13 (0.40)

Next we increased the number of inner iterations in the uncorrected and corrected implicit methods. The experimental results (Tables 2.2 and 2.3) show that the number of outer iterations is sharply decreased when the number of inner iterations is two. However, the number of outer iterations decreases more slowly as I_{in} increases further. There exists an optimal value I_{in} for computing time in both the methods, but it is problem dependent. Our experiments also show that the corrected implicit method converges a little bit faster than the uncorrected implicit method when $I_{in}>1$, which is consistent with our convergence analysis. In § 5 we will see that for larger problems,

TABLE 2.2
Experiments with the (uncorrected) implicit method.

($I_{in} > 1$) Outer iterations (seconds)				
$I_{in} = 1$	$I_{in} = 2$	$I_{in} = 3$	$I_{in} = 4$	$I_{in} = 5$
14 (0.40)	8 (0.34)	7 (0.40)	6 (0.44)	6 (0.54)

TABLE 2.3
Experiments with the corrected implicit method.

($I_{in} > 1$) Outer iterations (seconds)				
$I_{in} = 1$	$I_{in} = 2$	$I_{in} = 3$	$I_{in} = 4$	$I_{in} = 5$
13 (0.40)	8 (0.38)	6 (0.36)	6 (0.50)	5 (0.54)

the improvements in time for the corrected implicit method with $I_{in} > 1$ can be considerably larger than those seen here. Also, we will see in §§ 4 and 5 that the decrease in iterations is advantageous for parallel computation.

3. Globally convergent modifications of the explicit and implicit methods.

The explicit method and corrected implicit method are locally quadratically convergent to the solution. In other words, when the initial solution approximation is good enough, those methods are guaranteed to converge rapidly to a solution. However, it is often hard to find a good initial approximation for nonlinear problems in practice. In addition, many practical problems, such as the circuit equations, are highly nonlinear, and if the current solution approximation is not close enough, a Newton step may easily result in an increase in the function norm. For example, a small change in some voltage difference in a nonlinear circuit equation may result in a great change in an exponential term in a diode or transistor's function evaluation. Also, many block bordered equations result in nearly singular or singular Jacobians in the process of the iterations, for example, because a transistor with an exponential model is turned on at a nearly flat function curve (see Zhang, Byrd, and Schnabel [1989]). For these reasons, the methods need to be modified to handle unacceptable steps and singular Jacobian matrices in order to converge to a solution. In this section, we briefly describe the modifications we have used, which are motivated in part by their appropriateness for parallel distributed computation. They are described in more detail in Zhang [1989]. A global convergence analysis will be given in a forthcoming paper.

We achieve convergence from poor starting points by using a line search. The explicit method is just a standard Newton's method, so we can use a standard line search. That is, we calculate the overall step direction $d^k = (\Delta x_1^k \cdots \Delta x_q^k, \Delta x_{q+1}^k)$ as described in § 2.2, and then set

$$X^{k+1} = X^k + \lambda^k d^k,$$

where the steplength parameter $\lambda^k > 0$ is chosen by a line search procedure that assures sufficient descent on $\|F(X)\|_2$. Our line search is based upon Algorithm A6.3.1 in Dennis and Schnabel [1983].

The implicit method is more complicated since we have both inner and outer iterations. We need to choose the steps $\Delta x_i^{k,j} = (x_i^{k,j+1} - x_i^{k,j})$ in the inner iterations so that the overall step direction

$$d^k = \left(\sum_{j=0}^{I_{in}-1} \Delta x_1^{k,j} + \delta_1^k, \cdots, \sum_{j=0}^{I_{in}-1} \Delta x_q^{k,j} + \delta_q^k, \Delta_{q+1}^k \right), \tag{3.1}$$

where δ_i^k is the correction step (2.22), is a descent direction on $\|F(x)\|_2$. In addition, we would like the calculation of the steps $\Delta x_i^{k,j}$ for different values of i to be independent, so that the calculations of the inner iterations can be parallelized easily and efficiently.

Zhang [1989] shows that if each $\Delta x_i^{k,j}$ is calculated by

$$\Delta x_i^{k,j} = -\lambda_i^{k,j} A_i^{-1} f_i(x_i^{k,j}, x_{q+1}^k), \qquad i = 1, \cdots, q \tag{3.2}$$

(i.e., the step discussed in § 2.2 multiplied by a line search parameter $\lambda_i^{k,j} > 0$), the correction steps δ_i^k are calculated by (2.22), Δx_{q+1}^k is calculated by (2.11) as before, and f_{q+1} is linear, then d^k given by (3.1) satisfies

$$J(X^k)d^k$$
$$= -\left(\sum_{j=0}^{I_{in}-1} \lambda_1^{k,j} f_1(x_1^{k,j}, x_{q+1}^k), \cdots, \sum_{i=0}^{I_{in}-1} \lambda_q^{k,j} f_q(x_q^{k,j}, x_{q+1}^k), f_{q+1}(x_1^k, \cdots, x_q^k, x_{q+1}^k) \right).$$

Since the derivative of $\|F(X)\|_2^2$ in the direction d^k is equal to

$$F(X_k)^T J(X^k) d^k = -\sum_{i=0}^{q} \sum_{j=0}^{I_{in}-1} \lambda_i^{k,j} f_i(x_i^{k,j}, x_{q+1}^k)^T f_i(x_i^{k,0}, x_{q+1}^x) - \|f_{q+1}(X^k)\|^2,$$

it is clear that a sufficient condition for d^k given by (3.1) to be a descent direction on $\|f(X)\|_2$ is that for each $i = 1, \cdots, q$,

$$\sum_{j=0}^{I_{in}-1} \lambda_i^{k,j} f_i(x_i^{k,j}, x_{q+1}^k)^T f_i(x_i^{k,0}, x_{q+1}^k) > 0. \tag{3.3}$$

Note that (3.3) always holds for $I_{in} = 1$ (since each $\lambda_i^{k,1} > 0$), and that it is true for $I_{in} = 2$ if $\|f_i(x_i^{k,1}, x_{q+1}^k)\| < \|f_i(x_i^{k,0}, x_{q+1}^k)\|$ (which any line search will enforce) and $\lambda_i^{k,1} \leqq \lambda_i^{k,0}$. Thus we expect to get a descent direction most of the time. However, since (3.3) can be monitored independently for each i, the following parallel procedure could be used to guarantee that a descent direction is generated. For each j, the procedure calculates each $\Delta x_i^{k,j}$ by (3.2) using a standard line search as mentioned above, and then checks whether the corresponding partial sum of (3.3) is satisfied. If it is not, it sets $\Delta x_i^{k,l} = 0$ for $l = j, \cdots, I_{in} - 1$ and exits the inner iteration for x_i. The outer line search can be performed as in the explicit method.

Our approach for dealing with (nearly) singular Jacobians is based upon the Levenberg-Marquardt approach as described in Dennis and Schnabel [1983]. For a general system of nonlinear equations, if the current Jacobian matrix J is (nearly) singular, this approach modifies the search direction to be $-(J^TJ + \mu I)^{-1} J^T F$, where F is the current function value, and μ is a small positive number. This direction is a descent direction on $\|F(x)\|_2$ and is the solution to the trust region problem

$$\text{minimize}_d \ \|F + Jd\|_2 \quad \text{subject to } \|d\|_2 \leqq \Delta$$

for some $\Delta > 0$. In the limit as $\mu \to 0$, this direction equals $-J^+F$, where J^+ is the pseudoinverse of J.

In the explicit and implicit methods described in § 2, we need to solve systems of linear equations using the matrices $A_1, \cdots, A_q$ and the matrix $\hat{J} = P - \sum_{i=1}^q C_i A_i^{-1} B_i$. If any of these matrices, say, M, is nearly singular (i.e., either the factorization detects numerical singularity or the estimated condition number of M is greater than $macheps^{-1/2}$) we simply replace M^{-1} by $(M^T M + \mu I)^{-1} M^T$ in the formulas of § 2, where μ is chosen by the trust region strategy described by Dennis and Schnabel [1983], and is thus a function of M and the trust region size. These perturbations again have interpretations in terms of trust regions. Note also that the algorithms for deciding whether to perturb each $A_1, \cdots, A_q$, and for perturbing them if necessary, are totally independent so that they can be performed in parallel.

Combining these perturbation techniques with the inner line searches to assure descent at the outer iteration and global convergence is somewhat more complex, and will be addressed in a future paper. In our implementations, we have simply taken I_{in} inner iterations for each block i, $i = 1, \cdots, q$. We have used a standard line search to choose each $\lambda_i^{k,j}$ (requiring sufficient descent on f_i) but have not checked a condition like (3.3) that assures global descent, as this condition is more restrictive than necessary. To our knowledge, the algorithm has still always produced a descent direction.

The algorithm we implement is summarized below. If J or any A_i below is (nearly) singular, $\hat{J}^{-1}$ or A_i^{-1} is replaced by $(\hat{J}^T \hat{J} + \mu I)^{-1} J^T$ or $(A_i^T A_i + \mu_i I)^{-1} A_i^T$ for a small positive μ or μ_i, respectively. When f_{q+1} is linear, the explicit method is just a special case with $I_{in} = 1$ and each $\lambda_i^{k,1} = 1$.

IMPLICIT METHOD WITH GLOBAL MODIFICATION

1. For $j = 0, \cdots, I_{in} - 1$, calculate $x_i^{k,j+1} = x_i^{k,j} - \lambda_i^{k,j} A_i^{-1} f_i(x_i^{k,j}, x_{q+1}^k)$ where $\lambda_i^{k,j} \geqq 0$, $i = 1, \cdots, q$.
2. Form and factor $\hat{J} = P - \sum_{i=1}^q C_i A_i^{-1} B_i$.
3. Calculate $\Delta x_{q+1}^k = -\hat{J}^{-1} f_{q+1}(x_1^{k,I_{in}}, \cdots, x_q^{k,I_{in}}, x_{q+1}^k)$.
4. Calculate the corrections $\delta_i^k = -A_i^{-1} B_i \Delta x_{q+1}^k$ and set $\bar{x}_i^{k+1} = x_i^{k,I_{in}} + \delta_i^k$, $i = 1, \cdots, q$.
5. Calculate $X^{k+1} = X^k - \lambda^k d^k$ where $d^k = (\bar{x}_1^{k+1} - x_1^k, \cdots, \bar{x}_q^{k+1} - x_q^k, \Delta x_{q+1}^k)$.

4. Parallel explicit and implicit algorithms.

4.1. Motivation—LU factorization of block bordered linear equations. Note that the LU factorization of the block bordered Jacobian matrix

$$\begin{pmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_q & B_q \\ C_1 & C_2 & \cdots & C_q & P \end{pmatrix}$$

is

$$\begin{pmatrix} L_1 & & & & \\ & L_2 & & & \\ & & \ddots & & \vdots \\ & & & L_q & \\ \hat{C}_1 & \hat{C}_2 & \cdots & \hat{C}_q & L_{q+1} \end{pmatrix} \begin{pmatrix} U_1 & & & & \hat{B}_1 \\ & U_2 & & & \hat{B}_2 \\ & & \ddots & & \vdots \\ & & & U_q & \hat{B}_q \\ & & \cdots & & U_{q+1} \end{pmatrix},$$

where for $i = 1, \cdots, q$,

$$A_i = L_i U_i,$$
$$\hat{B}_i = L_i^{-1} B_i,$$
$$\hat{C}_i = C_i U_i^{-1},$$

and

$$L_{q+1}U_{q+1}=\hat{J}=P-\sum_{i=1}^{q} C_iA_i^{-1}B_i=P-\sum_{i=1}^{q} \hat{C}_i\hat{B}_i,$$

provided that the matrices A_i are nonsingular. (This is the same matrix $\hat{J}$ as in § 2.) The calculations of L_i, U_i, $\hat{B}_i$, and $\hat{C}_i$ for each i are independent, and thus can be parallelized very efficiently. The factorization of $\hat{J}$ must follow these calculations and will not parallelize as efficiently, especially on distributed memory multiprocessors, because it will require considerable communication.

A parallel version of the explicit method essentially consists of performing the above factorization in parallel at each iteration. The parallel version of the implicit method that we discuss next will be seen to perform closely related operations. The major difference will be that, by performing more than one inner iteration per outer iteration, it will spend a larger portion of its time on the calculations that parallelize very efficiently (those for blocks $1, \cdots, q$) and a smaller portion of its time on the calculations that parallelize less well—the formation and factorization of $\hat{J}$ and the outer line search. Thus the implicit method can be expected to parallelize more effectively than the explicit method, especially on distributed memory computers. If the two methods require similar amounts of time on sequential computers, as indicated in § 2, then the implicit method can be expected to be faster on parallel computers.

4.2. Parallel algorithms. Below we give a general description of a parallel corrected implicit method that is based upon the sequential method presented in §§ 2 and 3. The parallelism comes mainly from executing all the operations on blocks 1 through q, which have been designed to be independent, concurrently. The parallel explicit method is just the special case with $I_{in}=1$ and no inner line search.

```
INNER ITERATIONS
1. For $i=1, q$, Do in parallel:
   1.1. Factor $A_i$ and estimate its condition number Cond $(A_i)$.
   1.2. If Cond $(A_i)\leqq Tol$ then set $M_i=A_i$, $N_i=I$.
        Else choose $\mu_i>0$, form and factor $M_i=A_i^TA_i+\mu_iI$, set $N_i=A_i^T$.
   1.3. For $j=0, I_{in}-1$, Do:
        Solve $M_i\Delta x_i^{k,j}=-N_if_i(x_i^{k,j}, x_{q+1}^k)$ for $\Delta x_i^{k,j}$.
        Inner line search: $x_i^{k,j+1}=x_i^{k,j}+\lambda_i^{k,j}\Delta x_i^{k,j}$ for some $\lambda_i^{k,j}>0$.
   1.4. Solve $M_iW_i=N_iB_i$ for $W_i$.
   1.5. Calculate $T_i=C_iW_i$.

OUTER ITERATION
*2. Form $\hat{J}=P-\sum_{i=1}^q T_i$.
*3. Factor $\hat{J}$ and estimate its condition number Cond $(\hat{J})$.
*4. If Cond $(\hat{J})\leqq Tol$ then set $M=\hat{J}$, $N=I$.
    Else choose $\mu>0$, form and factor $M=\hat{J}^T\hat{J}+\mu I$, set $N=\hat{J}^T$.
*5. Solve $M\Delta x_{q+1}^k=-Nf_{q+1}(x_1^{k,I_{in}}, \cdots, x_q^{k,I_{in}}, x_{q+1}^k)$ for $\Delta x_{q+1}^k$.
 6. For $i=1, q$, Do in parallel:
    Calculate corrections $\delta_i^k=-W_i\Delta x_{q+1}^k$ and set $\bar{x}_i^{k+1}=x_i^{k,I_{in}}+\delta_i^k$.
*7. Outer line search: $X^{k+1}=X^k+\lambda^k(\bar{x}_1^{k+1}-x_1^k, \cdots, \bar{x}_q^{k+1}-x_q^k, \Delta x_{q+1}^k)$ for some
    $\lambda^k>0$.
```

The steps marked with stars require synchronization (on a shared memory multiprocessor) or communication (on a distributed memory multiprocessor). Step 2 requires synchronization if the matrices T_i are full. In the VLSI problems, however, the nonzero columns of B_i, and hence T_i, are disjoint (see § 1.2) and hence step 2 can be performed in parallel.

On shared memory machines, steps 3-5 can be performed in parallel using standard parallel methods for solving linear equations. On a distributed memory machine, it will only be efficient to perform steps 3-5 in parallel if the dimension of $\hat{J}$ is rather large. In our test problems, $\hat{J}$ was fairly small, so we performed steps 3-5 on one processor, on which we kept P, $\hat{J}$, and x_{q+1}. The remaining data was distributed in the obvious way: A_i, B_i, C_i, and x_i were stored together on one processor that handled block i. Step 7 includes two main operations, the calculation of trial points $\bar{x}^{k+1}$ and the evaluations of F at the points, which are performed in parallel on a shared memory machine, and may be performed in parallel on a distributed memory machine depending on their costs relative to the cost of communication.

5. Experimental results on a hypercube multiprocessor.

5.1. The test problem: A nonlinear block bordered circuit equation. The nonlinear block bordered application we consider for testing is the VLSI circuit simulation problem. Standard circuit simulation methods consist of stiffly stable implicit integration formulae to discretize the differential equations, Newton's method to solve the resulting nonlinear algebraic equations

$$F(X)=0, \tag{5.1}$$

and sparse LU decomposition to solve the linear equations that arise at each iteration

$$J\Delta X=-F(X), \tag{5.2}$$

where $J\in R^{n\times n}$ is the Jacobian matrix of (5.1). Typically, less than 2 percent of the entries of J are nonzero for $n>500$ (see, e.g., Sangiovanni-Vincentelli and Webber [1986]). The Newton iteration is repeated until the solution converges or the upper bound on the number of iterations is reached. The program then decides whether to accept the solution, based on its estimate of local truncation error and the number of iterations required.

As mentioned in § 1.2, partitioning the circuit leads to a block bordered system of nonlinear equations of the form (1.1) (see, e.g., Rabbat, Sangiovanni-Vincentelli, and Hsieh [1979]). Given a circuit network Γ, a group of partitioned subnetworks γ_i, $i=1,\cdots,q$, and the connecting current and voltage equations, the block bordered nonlinear system of equations is defined as follows. Currents between two subnetworks and voltages at the boundary are each represented by two variables, one in each subnetwork, which are set equal to each other by equations of f_{q+1}. Variables x_i ($i=1,\cdots,q$) are used to represent internal voltages and current variables in each of the q independent subnetworks. Some of these are the current connecting variables among the q subnetworks. The variables x_{q+1} are used to represent the voltage connecting variables among the q subnetworks. Here the equations for voltages and currents are standard current equations involving resistors, transistors, diodes, voltage sources, and other elements. Since the connecting equation f_{q+1} is linear, the coefficient matrices C_i, $i=1,\cdots,q$ for the current connecting functions are constant, and the coefficient matrix P for the voltage connecting function is also constant.

For a very large circuit, the network Γ may be divided into subnetworks recursively, which leads to a multilevel block bordered system of nonlinear equations. In such a case, the diagonal blocks A_i ($i=1,\cdots,q$) are themselves block bordered matrices. The border elements of the multilevel system represent the connections of the highest level.

We applied our algorithm to a simulation of the 741 op-amp circuit (see, e.g., Sedra and Smith [1982]), which was introduced in 1966 and is currently produced by almost every analog semiconductor manufacturer. The circuit is partitioned into four parts with roughly equal nodes in each subcircuit. A transistor is viewed as a nonlinear three-terminal device in the circuit. Thus, applying the Ebers–Moll transistor model (see Ebers and Moll [1954]), 24, 27, 23, and 27 KCL functions are defined in the first, second, third, and fourth block, respectively. The seven connections among the four blocks result in 14 linear current and voltage connecting functions. The total number of variables is $24+27+23+27+14=115$.

We also used a large analog filter composed of three 741 op-amp circuits (see, e.g., Smith [1971] and Valkenburg [1982]) to construct a two-level block bordered nonlinear system. The analog filter is first partitioned into three parts, each of which contains one 741 op-amp circuit. The first-level block bordered structure is thus formed with three diagonal blocks and one connecting block. Each of the diagonal blocks is a 741 op-amp circuit that is partitioned into the second-level block bordered structure.

5.2. The 741 op-amp circuit simulation on the Intel Hypercube. The nonlinear block bordered equations of the 741 op-amp circuit were solved in parallel on an Intel iPSC1 hypercube using the algorithm of § 4.2. The four blocks of the circuit were distributed among four nodes of the hypercube. For convenience, the steps involving the connection function f_{q+1} (steps 3–5 of the parallel algorithm) were performed on a different node which plays the control role. They could just as well have been done on one of the four nodes. Identical initial values were used as the inputs for all the above experiments, and the convergence tolerances were also the same for those experiments. The solutions of the experiments were verified by comparing them to the solutions computed by the program SPICE, which is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analysis (see Newton, Pederson, and Sangiovanni-Vincentelli [1988]).

Tables 5.1 and 5.2 show the experimental results for the explicit method. Tables 5.3 and 5.4 list the experimental results for the corrected implicit method with one or more than one inner iterations per outer iteration and with inner line searches. Note that as long as the inner line search is applied, the corrected implicit method even with one inner iteration per outer iteration is not the same as the explicit method.

In Tables 5.1 and 5.3, T_i $(i=1,\cdots,4)$ is the total computing time for all computations for solving the ith diagonal block on node i, T_b is the total computing time for all computations for solving the bottom block on the control node, T_c is the total

TABLE 5.1
The explicit method times for the op-amp 741 circuit.

T_1	T_2	T_3	T_4	T_b	T_c	N_{out}
12.81	14.20	13.45	14.58	3.42	0.43	20

TABLE 5.2
The explicit method parallel performance for the op-amp 741 circuit.

T_s	T_p	sp	eff
58.46	18.43	3.14	78.5%

TABLE 5.3
The corrected implicit method times for the op-amp 741 *circuit.*

I_{in}	T_1	T_2	T_3	T_4	T_b	T_c	N_{out}
1	11.81	13.06	11.96	13.28	2.84	0.38	18
2	12.60	14.01	12.83	14.45	1.65	0.32	15
3	11.28	12.84	11.46	13.01	1.38	0.25	12
4	18.48	20.57	18.81	21.34	1.32	0.25	11
5	29.04	32.12	29.92	32.89	1.34	0.24	11

TABLE 5.4
The corrected implicit method parallel performance for the op-amp 741 *circuit.*

I_{in}	T_s	T_p	sp	eff
1	52.95	16.5	3.21	80.25%
2	55.54	16.42	3.38	84.50%
3	49.97	14.64	3.43	85.75%
4	80.52	22.91	3.50	87.50%
5	125.31	34.47	3.60	90.0%

communication time for the computation, N_{out} is the total number of outer iterations required to converge to the solution, and I_{in} in Table 5.3 is the number of inner iterations used in the corrected implicit method. In the performance Tables 5.2 and 5.4, T_s is the computing time for solving the same problem on one node:

$$T_s = \sum_{i=1}^{4} T_i + T_b,$$

T_p is the parallel computing time;

$$T_p = \max(T_i, \cdots, T_4) + T_b + T_c,$$

sp is the speedup of the parallel computation:

$$sp = \frac{T_s}{T_p},$$

and eff is the parallel efficiency defined by

$$eff = \frac{sp}{\text{number of processors}}.$$

Our experiments show that the inner line search and inner iterations indeed speed up the convergence to the solution. For example, the experiment with one inner iteration with inner line search used 18 iterations to converge to the solution. The same experiment without inner line search (explicit method) used 20 iterations. As the number of inner iterations I_{in} is increased, the total number of outer iterations N_{out} decreases from 18 to 11, and the speedup increases because the bottom block computations constitute a smaller percentage of the overall computation. However, the sequential computing time only decreases by a small amount for $I_{in} = 3$, and then increases dramatically because the cost of the extra inner iterations swamps the small savings from the further decrease in the number of outer iterations. Both the sequential and parallel computing times are minimized when the number of inner iterations is $I_{in} = 3$, and the speedup in this case, 3.43, is good.

Our experiments also show that the bottleneck computing time T_b of the corrected implicit method is more than 50 percent lower than in the explicit method if more than one inner iteration is applied. The communication time is also lower since the total number of iterations, and hence the time to send the updated variables among the nodes, is less than for the explicit method. Consequently, the advantage of the implicit method over the explicit method should be greater in larger problems where the amount of communication and cost of solving the bottom block are larger.

5.3. Experiments for two-level block bordered circuit equations. We also solved in parallel the two-level system of nonlinear block bordered equations for an analog filter formed by connecting together three blocks of the 741 op-amp circuit. We again used the Intel iPSC1 hypercube multiprocessor. The linearization of these equations at each iteration has the form

$$J\Delta X = -F, \tag{5.3}$$

where J is the two-level block bordered matrix

$$\begin{pmatrix}
A_1^1 & & & B_1^1 & & & & & & & & & \\
& \ddots & & \vdots & & & & & & & & & \hat{B}_1 \\
& & A_q^1 & B_q^1 & & & & & & & & & \\
C_1^1 & \cdots & C_q^1 & P^1 & & & & & & & & & \\
& & & & A_1^2 & & & B_1^2 & & & & & \\
& & & & & \ddots & & \vdots & & & & & \hat{B}_2 \\
& & & & & & A_q^2 & B_q^2 & & & & & \\
& & & & C_1^2 & \cdots & C_q^2 & P^2 & & & & & \\
& & & & & & & & A_1^3 & & & B_1^3 & \\
& & & & & & & & & \ddots & & \vdots & \hat{B}_3 \\
& & & & & & & & & & A_q^3 & B_q^3 & \\
& & & & & & & & C_1^3 & \cdots & C_q^3 & P^3 & \\
& \hat{C}_1 & & & & & \hat{C}_2 & & & & \hat{C}_3 & & \hat{P}
\end{pmatrix},$$

$$\Delta X = (\Delta x_1^1, \cdots, \Delta x_q^1, \Delta x_{q+1}^1, \Delta x_1^2, \cdots, \Delta x_q^2, \Delta x_{q+1}^2, \Delta x_1^3, \cdots, \Delta x_q^3, \Delta x_{q+1}^3, \Delta \hat{x}_{q+1})^T,$$

and

$$F = (f_1^1, \cdots, f_q^1, f_{q+1}^1, f_1^2, \cdots, f_q^2, f_{q+1}^2, f_1^3, \cdots, f_q^3, f_{q\times 1}^3, \hat{f}_{q+1})^T.$$

The system (5.3) could be solved by applying the block bordered solver to each of the three block bordered submatrices, and then solving the whole system by applying the block bordered solver again. Alternatively, the block bordered Jacobian matrix may be reordered to

$$\begin{bmatrix}
A_1^1 & & & & & & & & & B_1^1 & & & \\
 & \ddots & & & & & & & & \vdots & & & \hat{B}_1 \\
 & & A_q^1 & & & & & & & B_q^1 & & & \\
 & & & A_1^2 & & & & & & & B_1^2 & & \\
 & & & & \ddots & & & & & & \vdots & & \hat{B}_2 \\
 & & & & & A_q^2 & & & & & B_q^2 & & \\
 & & & & & & A_1^3 & & & & & B_1^3 & \\
 & & & & & & & \ddots & & & & \vdots & \hat{B}_3 \\
 & & & & & & & & A_q^3 & & & B_q^3 & \\
C_1^1 & \cdot & C_q^1 & & & & & & & P^1 & & & \\
 & & & C_1^2 & \cdot & C_q^2 & & & & & P^2 & & \\
 & & & & & & C_1^3 & \cdot & C_q^3 & & & P^3 & \\
 & \hat{C}_1 & & & \hat{C}_2 & & & \hat{C}_3 & & & & & \hat{P}
\end{bmatrix},$$

with ΔX reordered to

$$\Delta X = (\Delta x_1^1, \cdots, \Delta x_q^1, \Delta x_1^2, \cdots, \Delta x_q^2, \Delta x_1^3, \cdots, \Delta x_q^3, \Delta x_{q+1}^1, \Delta x_{q+1}^2, \Delta x_{q+1}^3, \Delta \hat{x}_{q+1})^T,$$

and F reordered to

$$F = (f_1^1, \cdots, f_q^1, f_1^2, \cdots, f_q^2, f_1^3, \cdots, f_q^3, f_{q+1}^1, f_{q+1}^2, f_{q+1}^3, \hat{f}_{q+1})^T.$$

In solving this block bordered system, two levels of parallelism can be exploited. Let m be the number of amplifiers in the analog filter and q the number of subcircuits inside each amplifier; here $m=3$ and $q=4$. First the $m \times q$ independent operations for solving the diagonal blocks can be performed in parallel. Second, the m independent operations for transforming the matrices P^j, $j=1, \cdots, m$, and solving the resultant systems of equations can be performed in parallel. Finally, the very bottom block, with the matrix $\hat{P}$, must be transformed and solved.

In our test program, the 12 diagonal block equations of the analog filter were distributed among 12 nodes of the Intel hypercube. The first level of internal connection functions in each amplifier, f_{q+1}^j ($j=1, \cdots, 3$), was distributed to three of the 12 nodes, and the second level connection function among the three amplifiers in the analog filter, $\hat{f}_{q+1}$, was handled sequentially by one of the 12 nodes.

Tables 5.5 and 5.6 give the experimental results and the performance of the explicit method for solving the two-level analog filter equation. Tables 5.7-5.11 show the

TABLE 5.5
The explicit method times for the analog filter.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
13.49	14.93	14.21	15.34	13.44	14.85	14.16	15.46	13.36
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
14.76	14.25	15.63	3.01	3.12	3.17	0.58	1.31	21

TABLE 5.6
The explicit method parallel performance for the analog filter.

T_s	T_p	sp	eff
183.76	20.69	8.88	74.00%

TABLE 5.7
Corrected implicit method times for the analog filter, $I_{in} = 1$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
11.54	12.78	12.13	13.13	11.52	12.69	12.10	13.21	11.42
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
12.61	12.20	13.40	2.56	2.67	2.73	0.5	1.12	18

TABLE 5.8
Corrected implicit method times for the analog filter, $I_{in} = 2$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
11.83	11.93	11.84	12.09	11.91	12.01	11.89	12.12	11.84
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
12.05	11.94	12.13	1.65	1.65	1.64	0.36	0.73	12

TABLE 5.9
Corrected implicit method times for the analog filter, $I_{in} = 3$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
21.25	23.67	23.25	24.51	21.67	23.29	24.21	24.35	21.22
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
23.56	23.21	24.75	1.52	1.51	1.53	0.34	0.69	11

TABLE 5.10
Corrected implicit method times for the analog filter, $I_{in} = 4$.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
29.03	32.24	29.98	32.75	29.11	32.04	29.87	32.71	29.40
T_{10}	T_{11}	T_{12}	T^1	T^2	T^3	T_b	T_c	N_{out}
32.21	31.05	32.34	1.50	1.51	1.50	0.35	0.69	11

TABLE 5.11
Parallel performance of the corrected implicit method for the analog filter.

I_{in}	T_s	T_p	sp	eff
1	157.19	17.19	8.86	73.83%
2	148.68	14.86	10.01	83.40%
3	283.84	27.31	10.39	86.58%
4	373.08	34.89	10.69	89.08%

experimental results and performance of the corrected implicit method for solving the two-level block bordered analog filter equations with one to four inner iterations. The symbols in the tables have the same meanings as in Tables 5.1–5.4, with the following exception: T_i, $i=1,\cdots,12$, is the total time for the 12 first-level blocks, while T^j, $j=1,2,3$, is the time for the three second-level blocks.

Our experimental results show that the corrected implicit method is also more efficient than the explicit method on this larger block bordered system of equations. The total number of iterations N_{out} decreases from 18 to 11 as the number of inner iterations I_{in} is increased from 1 to 4, but the sequential computing time T_b only decreases from 157.19 to 148.68 for $I_{in}=2$, then increases again. The high speedups for $I_{in}=3$ and 4 in comparison to the same sequential method are not significant since the large number of inner iterations makes the algorithm inefficient, and the sequential time is suboptimal. For the optimal number of inner iterations, $I_{in}=2$, the speedup is 10.01 out of 12 processors and the efficiency is 83.40 percent. The computation time improvement over the parallel explicit method is 28 percent, as compared to 19 percent in the sequential case. Our parallel analog filter simulation experiment indicates that applying the implicit method to solving large block bordered circuit equations on a distributed memory multiprocessor can result in high efficiency.

6. Summary and future research.

We have introduced a corrected implicit method for solving block bordered systems of nonlinear equations. It allows multiple "inner" iterations, iterations on the variables, and equations of the q diagonal blocks, to be performed per each "outer" iteration, which involves all the variables and equations including the connecting bottom block. If only one inner iteration is performed per outer iteration, no line search is used, and the bottom connecting equations are linear, then the corrected implicit method is identical to the explicit method (Newton's method). When more than one inner iteration is performed per outer iteration, however, the methods are different, and in our experiments the corrected implicit method solves problems in somewhat less time than the explicit method on sequential computers. On parallel computers, the corrected implicit method has a larger advantage over the explicit method because it parallelizes more effectively, since the inner iterations constitute a larger percentage of the total computation and parallelize far better than the outer iterations. On one- and two-level block bordered problems from VLSI circuit design that we tested, the parallel efficiency of the fastest (sequential and parallel) corrected implicit method on an Intel iPSC1 hypercube was about 85 percent.

The methods presented in this paper all assume that the Jacobian matrix is available at each iteration, either analytically or by finite differences, and that it is not too expensive to evaluate. In some applications, however, the nonlinear equations are given by an expensive computational procedure, and analytic or finite difference Jacobians are very expensive to obtain. In such cases, for general systems of nonlinear equations, secant approximations to the Jacobian are used that are based entirely on function values at the iterates (see, e.g., Dennis and Schnabel [1983]). The development of related secant approximations to the Jacobian for block bordered nonlinear equations seems to be an attractive research topic, since it appears possible to construct approximations that retain the block bordered sparsity pattern of the Jacobian, and also allow the factorization of the Jacobian approximation to be updated efficiently.

REFERENCES

C. CHRISTARA [1988], *Spline collocation methods, software and architectures for linear elliptic boundary value problems*, Ph.D. thesis, Computer Science Department, Purdue University, West Lafayette, IN, August, 1988.

C. CHRISTARA AND E. HOUSTIS [1989], *A domain decomposition spline collocation method for elliptic partial differential equations*, in Proc. 4th Conf. Hypercube Concurrent Computers and Applications, Monterey, CA, March 6-8, 1989.

L. CHUA AND P. LIN [1975], *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, Prentice-Hall, Englewood Cliffs, NJ.

T. COLEMAN AND G. LI [1990], *Solving systems of nonlinear equations on a message-passing multiprocessor*, SIAM J. Sci. Statist. Comput., 11, pp. 1116–1135.

J. DENNIS AND R. SCHNABEL [1983], *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

J. EBERS AND J. MOLL [1954], *Large-signal behavior of junction transistors*, Proc. IRE, 42, pp. 1761–1772.

C. FARHAT AND E. WILSON [1986], *Concurrent iterative solution of large finite element systems*, Tech. Report, Civil Engineering Department, University of California, Berkeley, CA.

R. FONTECILLA [1987], *A parallel nonlinear Jacobi algorithm for solving nonlinear equations*, Tech. Report, Computer Science Department, University of Maryland, College Park, MD, May.

M. R. GAREY AND D. S. JOHNSON [1979], *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, CA.

P. GILL, W. MURRAY, AND M. WRIGHT [1981], *Practical Optimization*, Academic Press, New York.

M. MU AND J. RICE [1989], *Solving linear systems with sparse matrices on hypercubes*, Tech. Report CSD-TR-870, Computer Science Department, Purdue University, West Lafayette, IN, February.

A. NEWTON, D. PEDERSON, AND A. SANGIOVANNI-VINCENTELLI [1988], SPICE 3B1 *user's guide*, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA.

B. NOUR-OMID AND K. C. PARK [1986], *Solving structural mechanics problems on a Caltech hypercube machine*, Tech. Report, Mechanical Engineering Department, University of Colorado, Boulder, CO.

J. M. ORTEGA AND R. G. VOIGT [1985], *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27, pp. 149–240.

D. P. O'LEARY AND R. E. WHITE [1985], *Multi-splittings of matrices and parallel solution of linear systems*, SIAM J. Algebraic Discrete Meth., 4, pp. 137–149.

N. RABBAT AND H. HSIEH [1976], *A latent macromodular approach to large-scale sparse networks*, IEEE Trans. Circuits and Systems, CAS-23, pp. 745–752.

N. RABBAT, A. SANGIOVANNI-VINCENTELLI AND H. HSIEH [1979], *A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain*, IEEE Trans. Circuits and Systems, CAS-26, pp. 733–741.

N. RABBAT AND A. SANGIOVANNI-VINCENTELLI [1980], *Techniques of time-domain analysis of* LSI *circuits*, Tech. Report RC 8351 (#36320), IBM T. J. Watson Research Center, Yorktown Heights, NY, July.

C. ROMINE AND J. ORTEGA [1986], *Parallel solution of triangular systems of equations*, ICASE Tech. Report, National Aeronautics and Space Administration, Hampton, VA.

A. SANGIOVANNI-VINCENTELLI, L. CHEN, AND L. CHUA [1977], *An efficient heuristic cluster algorithm for tearing large-scale networks*, IEEE Trans. Circuits and Systems, CAS-24, pp. 709–717.

A. SANGIOVANNI-VINCENTELLI AND D. WEBBER [1986], *Computer architecture issues in circuit simulation*, in High Speed Computing, R. Wilhelmson, ed., University of Illinois Press, Chicago, IL.

A. SEDRA AND K. SMITH [1982], *Microelectronic Circuits*, CBS College Publishing, New York.

J. SMITH [1971], *Modern Operational Circuit Design*, John Wiley, New York.

M. VALKENBURG [1982], *Analog Filter Design*, CBS College Publishing, New York.

R. S. VARGA [1973], *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ.

R. E. White [1986], *Parallel algorithms for nonlinear problems*, SIAM J. Algebraic Discrete Meth., 7, pp. 137–149.

X. Zhang [1989], *Parallel computation for the solution of nonlinear block bordered equations and their applications*, Ph.D. thesis, Department of Computer Science, University of Colorado, Boulder, CO, July.

X. Zhang, R. Byrd, and R. Schnabel [1989], *Solving nonlinear block bordered circuit equations on hypercube multiprocessors*, in Proc. 4th Conf. Hypercube Concurrent Computers and Applications, Monterey, CA, March 6–8.