Board of Governors of the Federal Reserve System

International Finance Discussion Papers

Number 488

October 1994

A DISTRIBUTED BLOCK APPROACH TO SOLVING NEAR-BLOCK-DIAGONAL SYSTEMS WITH AN APPLICATION TO A LARGE MACROECONOMETRIC MODEL

Jon Faust and Ralph Tryon

Abstract

This paper demonstrates two advantages of well-known block variants of standard algorithms for solving nonlinear systems. First, if a problem is sufficiently close to block-diagonal, block algorithms may offer significant speed advantages on a single processor. Second, block Jacobi algorithms can easily and efficiently be distributed across multiple processors. We illustrate the use of a distributed block Jacobi algorithm to solve a large nonlinear macroeconometric model. For our application, on a four-processor Unix server, the algorithm achieves a speedup factor of more than 6 over the standard algorithm on a single processor. A speedup factor of about 2 is due to the added efficiency of the block algorithm on a single processor, and the remaining factor of 3 results from distributing the work over four processors.

A distributed block approach to solving near-block-diagonal systems with an application to a large macreconometric model

Jon Faust and Ralph Tryon[1]

A great many economists now have access to hardware that will support limited distributed processing. This hardware may be a local area network of DOS PCs or Unix workstations or possibly a workstation with multiple processors. From the standpoint of distributed processing, these systems may be limited by the relatively small number of processors that could reliably be used, say, between 3 and 15, and by slow communication among the processors. Given these constraints, algorithms that can be efficiently distributed over a small number of potentially diverse processors with slow inter-processor communication may have widespread applicability.[2]

In this paper, we demonstrate that in solving certain nonlinear systems, substantial benefits can be obtained by using a distributed algorithm that can trivially be implemented using widely available software and hardware. The application that motivated this is the solution of a large multi-country macroeconomic model with forward looking expectations at the Federal Reserve Board. We designed a distributed, block Jacobi variant of the Fair-Taylor algorithm to run on a network of Unix workstations or on a network of PCs. On a four-processor Solbourne server, the algorithm achieves a typical speedup factor of more than 6 over the standard algorithm on a single processor for our application. A speedup factor of about 2 is due to the added efficiency of the block algorithm on a single processor, and the remaining factor of 3 results from distributing the work over four processors.

[1] The authors are, respectively, a staff economist and a section chief at the International Finance Division of the Board of Governors of the Federal Reserve System. The authors thank Ray Board, Neil Ericsson, and Mico Loretan for helpful comments. The views in this paper are solely the responsibility of the authors and should not be interpreted as reflecting the views of the Board of Governors of the Federal Reserve System or of any other person associated with the Federal Reserve System.

[2] Coleman [1993] provides one example of such an algorithm for solving nonlinear dynamic models.

1 Distributed processing

The ideal problem for distributing across a small number of diverse processors would be divisible into a small number of totally independent blocks that could be solved simultaneously. Unfortunately, no solution algorithm for a simultaneous system can be divided into totally independent blocks, and one must consider breaking the algorithm into sub-problems that do have independent blocks.

For example, consider Jacobi's algorithm for solving the linear system,

$$0 = HY + h, \tag{1}$$

where $Y = (y_1, \dots, y_n)'$ and H is invertible. Jacobi's algorithm starts with the LDU decomposition, $H = L + D + U$, where D is a diagonal matrix and L and U are the upper and lower triangles of H. Rearranging (1) gives

$$Y = -D^{-1}(L + U)Y - D^{-1}h,$$

and suggests Jacobi iterations of the form,

$$\begin{aligned} Y^{i+1} &= -D^{-1}(L + U)Y^{i} - D^{-1}h, \\ &= \tilde{H}Y^{i} + \tilde{h}. \end{aligned} \tag{2}$$

Viewing each successive evaluation of (2) as a sub-problem, each equation of (2) is an independent block within the sub-problem. A distributed Jacobi algorithm could execute each iteration by simultaneously solving all the equations on different processors. The results of the iteration could then be collected from the various processors to form the inputs to the next iteration.

It is obviously important that the time benefits derived from simultaneously executing the blocks on different processors outweigh the overhead cost involved in communicating the problems to the various processors at the beginning of the sub-problem and collecting the answers upon conclusion. Given the constraint of relatively slow communication, one obviously wants to cast the problem in a way that is *coarse-grained*[3]—that requires many computations between each communication

[3] Wilson [1993] provides a useful introduction to the jargon of distributed computing.

event—and requires relatively small messages to be sent in each communication event. Such an algorithm will have a high *computation-to-communication* ratio.

The distributed Jacobi algorithm is likely to score poorly on this criterion. In a system with a large number of equations, evaluating one equation may take very few computations, but if each processor must communicate the result for its equation to a large number of other processors before proceeding to the next iteration, the communication overhead may entirely wipe out the benefit from simultaneous evaluation. The simplest way to raise the computation-to-communication ratio in this case involves using a block variant of Jacobi's method.

2 Two benefits of block algorithms

Block variants of Jacobi, Gauss-Seidel, and other standard iterative algorithms are well established [e.g., Varga, 1962; and Ortega and Rheinboldt, 1970]. Klein [1983], for example, discusses an early use of block Gauss-Seidel to solve the Brookings macroeconometric model. This section highlights two potential benefits of block algorithms for nearly block-diagonal systems. First, the block algorithms may have a substantial efficiency advantage over standard algorithms when executed on a single processor. Second, if a small number of blocks is chosen, the distributed block algorithm will have a high computation-to-communication ratio, allowing efficient distribution over multiple processors.

2.1 Faster solution time on a single processor

We begin with the linear case introduced above to provide intuition for the nonlinear case. The basic results here are based on Varga [1962].

2.1.1 The linear case

To generalize the *point Jacobi* algorithm introduced above to the *block Jacobi* method, begin by selecting an ordering of the elements of Y and partitioning the vector into

b blocks:

$$Y = (Y_1', \ldots, Y_b')'$$

Form the block LDU decomposition of H, $H = L_B + D_B + U_B$, where subscript B indicates the partitioning of H conformably with Y such that D_B is block diagonal with b square blocks. Assuming that each diagonal block is invertible, we can proceed as before:

$$\begin{aligned} Y^{i+1} &= -D_B^{-1}(L_B + U_B)Y^i - D_B^{-1}h \\ &= \tilde{H}_B Y^i + \tilde{h}_B \end{aligned} \tag{3}$$

Iterations on (3) will converge starting from any Y^0 if the spectral radius (the modulus of the largest eigenvalue) of $\tilde{H}_B$, $\rho(\tilde{H}_B)$, is less than one. The asymptotic rate of convergence is $-\log(\rho(\tilde{H}_B))$ [e.g., Varga 1962].

Block variants of Jacobi's method provide a range of solution options with direct inversion and point Jacobi as polar cases. With $b = 1$, $D_B = H$ and block Jacobi corresponds to direct inversion; with $b = n$, $D_B = D$ and the algorithm reduces to point Jacobi. The value of b that minimizes solution time depends on the trade-off between the computational resources required to invert D_B and those required for iterating on (3). Since the number of computations required to invert D_B is inversely related to the number of blocks,[4] setting $b = 1$ maximizes the number of computations required to invert D_B but minimizes the number of iterations required at zero. Selecting a larger number of blocks will be warranted if the gain in speed in inverting D_B exceeds any slowdown from raising the required number of iterations.

While raising the number of blocks from 1 to 2 must weakly raise the number of iterations required, little else can be said about the relation between the way a problem is blocked and rate of convergence.[5] Clearly, however, if H is block diagonal for some blocking, then for that blocking $L_B + U_B = 0$ and no iterations are required. More generally, as the largest element of $L_B + U_B$ goes to zero, the asymptotic rate of

[4] For b diagonal blocks of roughly size n/b, inverting D_B is of order n^3/b^2 when treating inversion of an $(n \times n)$ matrix as order n^3.

[5] Varga [1962] gives some results.

convergence rises without bound for any fixed D_B.[6] Thus, if a problem is sufficiently close to block diagonal, a block algorithm with an intermediate b may be quicker than either direct inversion ($b = 1$) or point Jacobi ($b = n$).

2.1.2 The nonlinear case

Now consider solving the $(n \times 1)$ nonlinear equation system,

$$0 = G(Y), \tag{4}$$

for the $(n \times 1)$ vector Y. Point Jacobi iterations now take the form

$$0 = g_j(y_1^i, \ldots, y_{j-1}^i, \mathbf{y}_\mathbf{j}^{\mathbf{i+1}}, y_{j+1}^i, \ldots, y_n^i) \qquad j = 1, \ldots, n,$$

where g_j is the j^{th} equation of G. If g_j is nonlinear, an iterative method for solving g_j will generally be required. In the neighborhood of a solution, Y^*, the system is approximately of the form (1) with $H = \partial G(Y^*)/\partial Y$. Thus, the reasoning of the linear case regarding convergence and optimal blocking will apply locally in the nonlinear case.

The block Jacobi algorithm involves repeatedly evaluating blocks of the form,

$$0 = G_j(Y_1^i, \ldots, Y_{j-1}^i, \mathbf{Y}_\mathbf{j}^{\mathbf{i+1}}, Y_{j+1}^i, \ldots, Y_b^i) \quad j = 1, \ldots, b, \tag{5}$$

where G_j is the j^{th} block of equations in G. When (5) is nonlinear, evaluation will generally require an iterative method. The iterations required to solve (5) on each Jacobi iteration are the nonlinear analog of inverting D_B in the linear case. In the nonlinear case, there is no explicit inverse, and the inverse must be taken implicitly on each iteration. For concreteness, consider solving the blocks by Newton's method. (In our application, the blocks are solved by Fair-Taylor; most standard iterative methods could be substituted for Newton in this discussion.) Given the problem

[6] The asymptotic speed of convergence is governed by $-\log(\rho(\tilde{H}_B))$. The spectral radius $\rho(\tilde{H}_B)$, is bounded above by the largest absolute row sum: $\max_j \sum_{i=1}^n |[\tilde{H}_B]_{j,i}|$. For any D_B, the maximum row sum must go to zero with the largest absolute element of $L_B + U_B$, implying that the asymptotic rate of convergence rises without bound.

$F(X) = 0$, Newton iterations are of the form

$$X^{j+1} = X^j - \left(\frac{\partial F(X^j)}{\partial X}\right)^{-1} F(X^j) \tag{6}$$

Iterations over (5) are called *outer loop* iterations. The Newton iterations required to solve (5) are called *inner loop* iterations.

Substituting Newton's method for direct inversion of D_B, we can repeat the optimal blocking arguments of the linear case. The block method provides a range of options between Newton's method (with $b = 1$) and point Jacobi ($b = n$). Raising b will make each inner loop iteration quicker, since each Newton iteration across all the blocks requires inverting b Jabobian matrices of size $(n/b \times n/b)$. More generally, raising b will speed the inner loop iterations for any inner loop algorithm that has some step the computational cost of which rises faster than one-for-one with block size.

If $G(Y)$ is block diagonal, for some blocking, in the sense that $\frac{\partial G}{\partial Y}$ is block diagonal, then the block algorithm for this blocking gives precisely the same answer as Newton's method.[7] When $\frac{\partial G}{\partial Y}$ is sufficiently close to block diagonal (globally or local to the solution), then the logic of the linear case can be repeated to show that the computations of the block algorithm will nearly correspond to those of straight Newton (globally or locally). In this case substantial time savings can result from using a block Jacobi algorithm ($b > 1$) and Newton on the inner loops in place of straight Newton ($b = 1$).

The basic intuition for this result is that the number of calculations required to complete an iteration may be reduced by judiciously setting certain partial derivatives to zero. If these partial derivatives are sufficiently near zero, setting them to zero for certain calculations will not substantially degrade the progress made toward a solution on each iteration.

[7] The block algorithm may be faster in this case than Newton, but the same speed advantage could be obtained using a sparse matrix inversion routine in the Newton algorithm that recognized the block structure of the Jacobian.

2.2 Good computation-to-communication ratio

The second potential benefit of the block Jacobi algorithm is that it can be simply and efficiently distributed. The inner loop of the block algorithm involves iterative solution of b independent blocks of equations. These blocks clearly could be solved simultaneously on b separate processors.[8] If the total number of equations is large relative to the number of blocks, each block will require many computations to solve and the computation-to-communication ratio is likely to be quite favorable.

For example, suppose that we hold the number of blocks fixed at b, assume that the blocks are approximately equal in size, and raise the number of equations, n. The amount of information each block must read before and write after each inner loop iteration rises at most linearly with n: the block must write its solution to at most n/b equations and read the solution to at most $(b-1)n/b$ equations solved by the other blocks.[9] As noted above, the number of computations required to solve each block will rise faster than linearly with block size. With the number of blocks fixed, as the system grows, the cost of solving the blocks rises faster than the communication cost, and the computation-to-communication ratio becomes more favorable. Thus, the block Jacobi method is likely to be useful on large systems being evaluated using a small number of processors.

3 The distributed algorithm

This section provides a stylized description of the way we implemented the block algorithm—further detail is provided below in the discussion of the application. The only hardware required for the algorithm are multiple processors (perhaps separate computers) that share access to some file storage device. The software requirements are similarly minimal. We assume that the user has a solution algorithm for solving

[8] This distributability explains the preference for block Jacobi over block Gauss-Seidel, for example. In block Gauss-Seidel, the updated answers from blocks 1 through $j-1$ are used in solving the j^{th} block, which requires that the blocks be run sequentially.

[9] The use of *at most* is required because, for example, if the system is somewhat sparse—that is, if some variables do not occur in blocks—then those variables need not be read by all blocks.

the blocks in the inner loop. The primary software requirement is that the inner loop algorithm be implemented in software that provides minimal programmability: one must be able to wrap a programming loop around the inner loop solution algorithm, write and read disk files on the shared server, and branch within the program depending on the contents of the disk files.

All communication is done through disk files on the shared file server; thus, no special software for handling interprocess communication is required. This method of communication may be quite slow, but was chosen in the interest of creating the most portable and simplest possible illustration of distributed processing. Below we verify that for our application, the cost of choosing this inelegant communication approach is small.

The algorithm is programmed in a simple master-worker setup. There is one worker process per block running the software with the inner loop algorithm. The master controls the operation of the workers, and involves a simple program:

Master program

```
1.  Initialize all workers.
2.  Loop:
3.          Start each worker.
4.          Wait until all workers stop.
5.          Read status of each worker.
6.  Continue loop until all workers report convergence.
7.  Stop all workers.
```

The master-worker communication is conducted by writing flags to disk files. When the workers are running, the master periodically reads the flags to see if any workers are still running. When the workers have stopped, they periodically read the flags to see if the master has told them to re-start.

In the worker program, X_j is defined as those elements of Y_j that enter some equation in any block other than the j^{th} block. These are the variables that are endogenous to the j^{th} block and also enter some other block; thus, they are the only variables that must be communicated among the workers.[10]

[10] worker i need only read in those elements of X_j, $j \neq i$, that are used in block i. The workers in our implementation exploit this efficiency, which is not shown in the summary here.

Program for worker j

```
1.  Outer loop:
2.      Wait until master says start.
3.      Read $X_k, k \neq j$, from common space.
4.      Set $Y_j^0 = Y_j^i$ from previous call or initialization.
5.      Store $X_j^0$.
6.      $i = 0$.
7.      Inner loop:
8.          Compute $Y_j^{i+1}$ from $Y_j^i$ and $X_k$, $k \neq j$.
9.          Increment $i$.
10.     Continue loop until $d(Y_j^i, Y_j^{i-1}) < \epsilon$ or $i > \bar{\imath}$.
11.     Write $X_j^i$ to common space.
12.     If $i \leq \bar{\imath}$ and $d(X_j^i, X_j^0) < \epsilon$, report convergence;
            otherwise report nonconvergence.
13. Continue loop until master says stop.
```

In steps (10) and (12), d is a metric on the distance between two successive answers. In our implementation,

$$d(v,w) = \max_{k \in 1,\ldots,n} \frac{|v_k - w_k|}{|w_k| + \gamma} \tag{7}$$

for any $(n \times 1)$ vectors v and w.[11] In our case, $\gamma = 10$.

As implemented in worker step (10), the inner loop iterations stop in two cases: if the convergence criterion for the problem is met, and if a maximum iteration count, $\bar{\imath}$, is reached. The workers stop after $\bar{\imath}$ iterations even if they have not converged in order to allow the workers to share whatever progress they have made. If one sets $\bar{\imath}$ too high, time may be wasted on early outer loop iterations needlessly refining solutions for the blocks. With $\bar{\imath}$ too low, too much time may be spent sharing results that have not changed much from the previous outer loop iteration. In our application, $\bar{\imath} = 5$; the results were not very sensitive to setting $\bar{\imath}$ between 2 and 10, but were sensitive to setting $\bar{\imath}$ too high.

A worker reports overall convergence to the master in (12) if the block has converged in (10), and if X_j did not change by more than ϵ from the initial value, X_j^0, for the current outer loop iteration. The value X_j^0 is what all the other workers

[11] This is the standard form of convergence criteria used in solving large macroeconometric models. Other criteria might well be of interest, e.g., [Wolfe, 1978].

are taking as exogenous on the current outer loop iteration. Thus, the worker reports convergence if the updated value of X_j reported in (11) is no more than ϵ away from the previous report.

In order to divide the gains from the distributed block algorithm between the gains from blocking alone and the gains from distributing the blocks, we also implemented a single processor (SP) version of the block algorithm. This required few changes to the distributed processor (DP) version. Basically, in step 3 of the master, each of the blocks is run in sequence, rather than running all of them simultaneously. The SP and DP versions of the algorithm execute the same computations and arrive at the same solution. While the maximum theoretical speedup factor for the DP algorithm over the SP algorithm is equal to the number of processors applied, two factors will limit the speedup obtained. First, the SP algorithm avoids inter-processor communication overhead of the DP algorithm. Second, the DP algorithm has a *synchronization barrier*: at the end of each outer loop iteration, each worker must wait until all others have finished. Thus, if the load on the workers is not balanced, some processors may lay idle for a time.

4 Application to a multi-country model

We applied the algorithm to solving a multi-country macroeconometric model with conventional rational expectations. The model is nonlinear, involving, for example, log and log-linear specifications. The rational expectations involve equations with forward looking variables of the form[12]

$$y_{j,t} = f(y_{j,t+1}, y_{j,t-1}, y_{j,t}). \tag{8}$$

While much work is underway on alternative solution methods for these models [e.g., Fisher, 1992; Boucekkine, 1994; Brauset, 1994], the standard approach to solving such models is the Fair-Taylor algorithm [Fair and Taylor, 1983].[13] We

[12] It is not the dating conventions, but the roots of the system that ultimately determine what variables are forward looking and must be solved forward. See, e.g., Blanchard and Kahn [1980].

[13] Fisher [1992] has a detailed discussion of Fair-Taylor and other approaches to the solution of models with forward-looking expectations.

chose to explore a block Jacobi variant of Fair-Taylor.

The Fair-Taylor algorithm is a multi-step procedure. The innermost step uses some standard algorithm to solve the model conditional on paths for the forward-looking variables ($y_{j,t+1}$ in (8))—in our implementation, Newton's method is used. The algorithm then repeatedly substitutes the actual solutions for the forward looking variables for the assumed paths on the previous iteration and re-solves the model until convergence is reached.[14] Terminal conditions for each forward-looking variable must be specified—the model is simulated sufficiently far into the future that the solution over the period of interest is invariant to the terminal conditions chosen.

4.1 The RE7 multi-country model

We apply the distributed-processing block algorithm to solve RE7, an experimental macroeconometric model of the major industrial countries developed in the Division of International Finance at the Federal Reserve Board.[15] RE7 has about 1050 equations that make up seven country models. The individual country models are similar in structure but are not identical: the largest has almost 200 equations, and the smallest about 140 equations.

A crucial decision in implementing a block variant of Fair-Taylor is how to block the system. The goal, of course, is to order the equations such that the Jacobian (at the solution) is nearly block-diagonal. We made each of the seven country models a block.[16] While there are many important interactions among the country blocks, these are considerably less important than the interactions within a country block. Further, these international interactions operate through relatively few channels: the

[14] In the standard terminology, we have just described two of three Fair-Taylor steps: Fair-Taylor type 2 iterations involve repeatedly solving the model with updated values for the forward looking variables. Type 2 iterations nest type 1 iterations—the iterations required to solve the model conditional on a given path for the forward-looking variables. Type 3 iterations, used to test sensitivity to terminal conditions, are not considered here.

[15] Ralph Tryon and Joseph Gagnon developed RE7, which is a rational expectations version of the Multi-Country Model developed at the Board. See Edison, Marquez, and Tryon [1987] and Stevens et al. [1984].

[16] All exchange rates in the model are bilateral exchange rates with the U.S., and the exchange rate equations were included in the U.S. block.

typical country block has six X variables—variables that are treated as endogenous in the country block, but treated as exogenous in some other country block. This gives the system a strong element of block-diagonality given our blocking along country borders.[17] We did not explore other approaches to blocking.[18]

The model is typically used to simulate alternative policy scenarios. First, a baseline solution is obtained for the time period of interest, either historical or forecast. Then an alternative path for an exogenous variable—such as the price of oil, or the growth rate of money—is specified, and the model is solved again. For a model as large as RE7, obtaining a Fair-Taylor solution to such problems can require several hours on a fast UNIX workstation.

4.2 Software and hardware implementation

The RE7 model is implemented in Portable TROLL running under the SunOS (UNIX) operating system. Portable TROLL is a software package designed for developing and simulating large time-series econometric models.[19] The basic Fair-Taylor algorithm and block algorithms used for this paper were implemented by the authors using Portable TROLL's macro programming language. The outcome of this experiment could be distorted if the TROLL implementation of Fair-Taylor were poorly designed to handle large systems. Brillet [1994] presents some evidence that TROLL's Fair-Taylor algorithm is very efficient, at least relative to other available packages.[20]

The block algorithm closely follows the program outlined in the previous section. The Fair-Taylor algorithm is used for the inner loop iterations. Each worker is a

[17] This procedure is very similar to that used in Project LINK, which involves a large number of country models [Klein, 1983].

[18] Gilli [1992] and Gilli and Pauletto [1994] systematically explore the block structure of the Jacobian of macroeconomic models, not heavily exploiting particularities of the model. Board and Tinsley [1994] explore search algorithms for finding block structure of an economic model.

[19] Intex Solutions, Inc., Needham, Massachusetts. Portable TROLL was written by Peter Hollinger and Leonid Spivakovsky, drawing on the mainframe TROLL program developed at the Massachusetts Institute of Technology and the National Bureau of Economic Research.

[20] On limited range of experiments on a 501 equation model, TROLL's Fair-Taylor algorithm was five times faster in simulating than the nearest of the three other packages tested.

separate process in UNIX running TROLL to solve one of the country model blocks. The worker processes share data directories with each other and with the master. The master communicates with the workers using status flags written to disk files. To prevent read/write conflicts, the master and the worker each wait for exclusive write access to the status flag while either reading or writing.

At the beginning of each outer loop step, each worker reads the required X variables from a subdirectory shared with all the other worker processes. At the end of each outer loop iteration, each worker writes its X variables to the shared subdirectory. Each worker builds the lists of variables to read and write during initialization, so the user need not specify them.

The primary hardware platform is a Solbourne model 704/6E UNIX server with four processors which can run up to four processes simultaneously with only a slight overhead cost. (If more than four processes are running simultaneously, they share the four processors.) We also tested the algorithm on seven 33-MHz 80486-based PCs running under DOS and communicating across a Token Ring. The results for the DOS-based test are reported in footnote 22.

4.3 The results

The experiments involve solving for the change from baseline caused by shocks to some exogenous variables. We report results for three different shocks with RE7. Shock A is a change in monetary policy in one country that has little real impact on other countries. Shock B is a single-country fiscal policy scenario, which given the structure of RE7 has relatively strong feedbacks among all the countries. Shock C is a world fiscal policy change that is balanced across the countries.

Table 1 reports the simulation time for simulating these three scenarios in RE7 using one block (i.e., conventional Fair-Taylor) and with seven blocks, running on a single processor.[21] Solving the model in seven blocks yields a speedup factor of

[21] These experiments use $\bar{\imath} = 5$, $\epsilon = 0.00001$, $\gamma = 10$. Throughout, the ϵ and γ parameters are the same for the Fair-Taylor and block algorithms; thus, each stops when two successive answers are sufficiently close as measured by (7). Convergence of the Fair-Taylor algorithm in this sense does not imply convergence of the block algorithm, and vice versa. Thus, some care must be taken in

Shock	1 Block	7 Blocks	Speedup factor
Shock A	3646	1425	2.6
Shock B	6726	3539	1.9
Shock C	8749	4088	2.1

Single processor, time in seconds.

Table 1: Solution times with and without blocking

Shock	1 Block		7 Blocks		Speedup factor
	#	seconds	#	seconds	
Shock A	78	46.7	69	20.7	2.3
Shock B	146	46.1	164	21.6	2.1
Shock C	210	41.7	230	18.4	2.3

Single processor; speedup factor is per iteration.

Table 2: Number of inner loop iterations and time per iteration

between 1.9 and 2.6 over the time for solving it as one block. This speedup can be decomposed into the speedup on each inner loop iteration and any rise or fall in the required number of inner loop iterations. The number of inner loop iterations required is roughly the same for the blocked and unblocked solutions (Table 2). Thus, the speedup from blocking essentially comes from the fact that the blocked inner-loop iterations are quicker to execute by a factor of more than 2.

Table 3 compares the results from running the 7-block algorithm on one processor and on four processors simultaneously. The speedup factor shown here is purely the result of distributing the problem onto multiple processors, since the algorithm follows exactly the same iterations in either case. The distributed processing speedup factor is between 3.1 and 3.3. This is less than the theoretical limit of 4 by about 20 percent.

Two major factors can account for this shortfall. The first is communication

interpreting these speedup numbers. We did very limited work tightening the convergence criteria (shrinking ϵ) and verifying that the algorithms converge to the same point and that the speedup was relatively stable for a range of ϵ.

Shock	One Processor	Four Processors	Speedup factor
Shock A	1425	441	3.2
Shock B	3539	1088	3.3
Shock C	4088	1315	3.1

Seven blocks.

Table 3: Solution times with one processor and four processors

overhead—the time spent writing messages to disk, waiting for the messages to be read, and writing and reading the X variables. Since we have chosen the simple approach of communicating through disk files, it is important to know what penalty we are paying for not using a more elegant approach. Our data indicate that at most 5 percent of the solution time could be attributed to this communication overhead. Sending messages instantaneously, therefore could have raised the typical speedup factor to about 3.4.

The vast majority of the shortfall relative to the potential speedup of factor of 4 is due to the synchronization barrier. The load on the processors is not balanced, and some processors lay idle at the end of some outer loop iterations. While it would be impossible to select a blocking that distributed the work evenly for all shocks, we might achieve better average performance by breaking up the U.S. block, which is about one-third larger than the smallest country block. Setting a lower $\bar{\imath}$ or looser convergence criteria for the U.S. block on early iterations might also help.

The total speedup factor for the distributed, block algorithm is the product of the gain from blocking and the gain from distributing. This factor is 8.3 for Shock A, 6.3 for Shock B, and 6.5 for Shock C.[22]

[22] As noted above, we also ran shock A on a network of seven DOS-based 33-MHz PCs. The times were 5819 seconds for one processor and 1373 for seven, giving a speedup of 4.2. Almost all of the short-fall from the theoretical bound of a seven times speedup was due to the unbalanced load.

5 Conclusions

We set out to explore the gains that could be obtained by taking the simplest steps toward distributed processing to solve a large macroeconometric model. For our model, blocking and distributing over 4 processors yielded a speedup factor of over 6. Gains of this magnitude were required to make regular use of the model feasible, and the algorithm is now used in production work at the Federal Reserve Board. A variant of the algorithm using a different inner-loop algorithm is being investigated for use in solving the International Monetary Fund's Multimod model.

Block Jacobi methods can easily be applied to a wide range of problems. The method will perform best in nearly block-diagonal problems. While we did not explore the required degree of block-diagonality, our application illustrated that the substantial speedup per iteration from solving a problem in small blocks—a factor of about two in our case—could pay for a substantial penalty in terms of number of iterations to convergence. Thus, the approach may be useful even in problems that are not as close to block-diagonal as our model.

A large number of economic and econometric problems have a degree of block diagonality. For example, any *sectoral* model with limited channels for feedback has this form—whether the sectors are countries, sectors of a single economy, or nearly separable portions of individual decision problems. In econometrics, the information matrix associated with many maximum likelihood problems will be near block diagonal. For example, the Gaussian information matrix for vector autoregressions is block diagonal, and will be nearly so given some restrictions on the autoregression.

References

Blanchard, O.J., and C.M. Kahn. 1980. "The solution of linear difference model under rational expectations," *Econometrica*, 48, 1305–1311.

Board, Raymond, and Peter Tinsley. 1994. "Smart systems and simple agents: industry pricing by parallel and genetic strategies" manuscript, Federal Reserve Board, June.

Boucekkine, R. 1994. "An alternative methodology for solving nonlinear forward-looking models," *Journal of Economic Dynamics and Control*, forthcoming.

Brillet, Jean Louis. 1994. "Solving large models on micro-computers: a review of available packages," manuscript, INSEE.

Bruaset, Are. 1994. "Efficient solution of linear equations arising in a nonlinear economic model," manuscript, SINTEF.

Coleman, John. 1993. "Solving nonlinear dynamic models on parallel computers," *Journal of Business and Economic Statistics*, 11(3), 325–330.

Edison, Hali J., Jaime R. Marquez and Ralph W. Tryon. 1987. "The structure and properties of the Federal Reserve Board Multicountry Model," *Economic Modelling*, 4 (2), 115–315.

Fair, Ray C. and John B. Taylor. 1983. "Solution and maximum likelihood estimation of dynamic nonlinear rational expectations models," *Econometrica*, 51, 1169–1186.

Fisher, Paul. 1992. *Rational expectations in macroeconometric models.* Kluwer: Boston.

Gilli, Manfred. 1992. "Causal ordering and beyond," *International Economic Review,* 33(4), 957–971.

Gilli, Manfred and Giorgio Pauletto. 1994. "Parallel algorithms for solving rational expectations models," manuscript, University of Geneva, June.

Klein, Lawrence, 1983. *Lectures in Econometrics.* Elsevier: Amsterdam.

Ortega, J.M. and W.C. Rheinboldt. 1970. *Iterative solution of nonlinear equations in several variables.* Academic Press: New York.

Stevens, Guy, Richard Berner, Peter Clark, Ernesto Hernandez-Cata, Howard Howe and Sung Kwack, 1984. *The U.S. Economy in an Interdependent World: A multicountry Model*, Board of Governors of the Federal Reserve System, Washington, D.C.

Varga, Richard S. 1962. *Matrix iterative analysis.* Prentice-Hall: Englewood Cliffs, New Jersey.

Wilson, Gregory. 1993. "A glossary of parallel computing terminology," IEEE Parallel and Distributed Technology, Feb., 52–67.

Wolfe, M.A. 1978. *Numerical methods for unconstrained optimization.* Van Nostrand Reinhold: Wokingham, Berkshire, England.

International Finance Discussion Papers

IFDP Number	Titles	Author(s)
	1994	
488	A distributed block approach to solving near-block-diagonal systems with an application to a large macroeconometric model	John Faust Ralph Tryon
487	Conditional and Structural Error Correction Models	Neil R. Ericsson
486	Bank Positions and Forecasts of Exchange Rate Movements	Michael P. Leahy
485	Technological Progress and Endogenous Capital Depreciation: Evidence from the U.S. and Japan	Robert Dekle
484	Are Banks Market Timers or Market Makers? Explaining Foreign Exchange Trading Profits	John Ammer Allan D. Brunner
483	Constant Returns and Small Markups in U.S. Manufacturing	Susanto Basu John G. Fernald
482	The Real Exchange Rate and Fiscal Policy During the Gold Standard Period: Evidence from the United States and Great Britain	Graciela L. Kaminsky Michael Klein
481	The Debt Crisis: Lessons of the 1980s for the 1990s	Graciela L. Kaminsky Alfredo Pereira
480	Who Will Join EMU? Impact of the Maastricht Convergence Criteria on Economic Policy Choice and Performance	R. Sean Craig
479	Determinants of the 1991-93 Japanese Recession: Evidence from a Structural Model of the Japanese Economy	Allan D. Brunner Steven B. Kamin
478	On Risk, Rational Expectations, and Efficient Asset Markets	Guy V.G. Stevens Dara Akbarian
477	Finance and Growth: A Synthesis and Interpretation of the Evidence	Alexander Galetovic
476	Trade Barriers and Trade Flows Across Countries and Industries	Jong-Wha Lee Phillip Swagel

Please address requests for copies to International Finance Discussion Papers, Division of International Finance, Stop 24, Board of Governors of the Federal Reserve System, Washington, D.C. 20551.

International Finance Discussion Papers

IFDP Number	Titles	Author(s)
	1994	
475	The Constancy of Illusions or the Illusion of Constancies: Income and Price Elasticities for U.S. Imports, 1890-1992	Jaime Marquez
474	The Dollar as an Official Reserve Currency under EMU	Michael P. Leahy
473	Inflation Targeting in the 1990s: The Experiences of New Zealand, Canada, and the United Kingdom	John Ammer Richard T. Freeman
472	International Capital Mobility in the 1990s	Maurice Obstfeld
471	The Effect of Changes in Reserve Requirements on Investment and GNP	Prakash Loungani Mark Rush
470	International Economic Implications of the End of the Soviet Union	William L. Helkie David H. Howard Jaime Marquez
469	International Dimension of European Monetary Union: Implications For The Dollar	Karen H. Johnson
468	European Monetary Arrangements: Implications for the Dollar, Exchange Rate Variability and Credibility	Hali J. Edison Linda S. Kole
467	Fiscal Policy Coordination and Flexibility Under European Monetary Union: Implications for Macroeconomic Stabilization	Jay H. Bryson
466	The Federal Funds Rate and the Implementation of Monetary Policy: Estimating the Federal Reserve's Reaction Function	Allan D. Brunner
465	Understanding the Empirical Literature on Purchasing Power Parity: The Post-Bretton Woods Era	Hali J. Edison Joseph E. Gagnon William R. Melick
464	Inflation, Inflation Risk, and Stock Returns	John Ammer
463	Are Apparent Productive Spillovers a Figment of Specification Error?	Susanto Basu John S. Fernald
462	When do long-run identifying restrictions give reliable results?	Jon Faust Eric M. Leeper