NASA
Technical
Paper
2892

1989

Parallel Gaussian Elimination of a Block Tridiagonal Matrix Using Multiple Microcomputers

Richard A. Blech
Lewis Research Center
Cleveland, Ohio

NASA

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

Summary

The solution of a block tridiagonal matrix using parallel processing is demonstrated in this report. The multiprocessor system which obtained the results and the software environment used to program that system are described. Theoretical partitioning and resource allocation for the Gaussian elimination method used to solve the matrix are discussed. The results obtained from running one-, two-, and three-processor versions of the block tridiagonal solver are presented. The PASCAL source code for these solvers is given in the appendix, and it may be transportable to other shared-memory parallel processors, provided that the synchronization routines are reproduced on the target system.

Introduction

Many computationally intensive problems can benefit from the use of parallel processing. One such problem, common to many fluid mechanics and structural dynamics applications, is the solution of large matrix equations. Because of the differencing techniques used in solving the partial differential equations that describe fluids and structures systems, the resulting matrices often exhibit a block tridiagonal structure. The block tridiagonal matrix requires much less computation to solve than a full N by N matrix. A full matrix requires approximately N^3 operations to solve; a block tridiagonal matrix requires approximately N operations.

Although the block tridiagonal structure significantly reduces computational effort, considerable time is still spent in the matrix solution. This is especially true in many iterative linearization techniques, such as Newton-Raphson, where a full matrix solution is required for every iteration. Because of this, other parallel processing techniques which can further reduce the amount of computation required to arrive at a solution should be investigated.

This paper presents the solution of a block tridiagonal matrix on a parallel processor. The block tridiagonal equations analyzed were taken from a transient rotor dynamics simulation program (ref. 1). In this program, Gaussian elimination is used to solve the matrix.

The real-time multiprocessor simulator (RTMPS) was used to solve these equations in parallel (refs. 2 to 5). The RTMPS is a parallel processor designed to do real-time simulation of dynamic systems. The hardware consists of dual busses with processors on each bus. A dual-port memory provides communication between the two busses by connecting processors on one bus to the processors on the other bus. Considerable software support is provided for one-dimensional scalar problems by a real-time multiprocessor language (RTMPL) and a real-time multiprocessor operating system (RTMPOS).

The potential of parallel processing for improving the performance of linear algebra routines has prompted a significant amount of research (refs. 6 to 8). Also, a significant amount of literature exists on the use of vector processors for linear algebra. Since vectorization of code involves the identification of the lowest level of parallelism (e.g., operation level parallelism), the principles behind both areas of research are very similar. Because of the high percentage of nested loops in linear algebra code, the ideal architecture for most linear algebra applications would consist of multiple vector processors.

This paper presents the application of parallel processing using one particular architecture (RTMPS) to one algorithm (Gaussian elimination). This combination, however, may not be the best approach to the problem. As mentioned previously, there are other architectures and algorithms that may be better suited for this application. The RTMPS system was used for this study because it was the only parallel processing hardware conveniently available. The intent of this study is to identify some practical aspects of implementing a commonly used algorithm on a parallel processor. The investigation of other architectures and algorithms will be the focus of future research.

Problem Description

The structure of the block tridiagonal matrix is shown in figure 1. Each block row, except the first and last, consists of three M by M blocks. There are N block rows total, including the first and last. If this matrix is called A, then the general problem is to find the solution to the system of equations

$$A\mathbf{x} = \mathbf{b}$$

where $\mathbf{x}$ and $\mathbf{b}$ are vectors, N elements in length.

A common method for solving this system is to perform a forward elimination of all coefficients below the diagonal and then a back substitution to solve for the vector $\mathbf{x}$. This procedure, called Gaussian elimination, is illustrated in the following example for a 3 by 3 matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & a'_{12} & a'_{13} \\ \phi & a'_{22} & a'_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} b'_1 \\ b'_2 \\ b_3 \end{bmatrix} \quad \begin{array}{l} a'_{11} = a_{11}/a_{11} = 1;\ a'_{12} = a_{12}/a_{11};\ a'_{13} = a_{13}/a_{11} \\ b'_1 = b_1/a_{11};\ a'_{22} = a_{22} - a_{21}a'_{12} \\ a'_{23} = a_{23} - a_{21}a'_{13};\ b'_2 = b_2 - a_{21}b'_1 \end{array}$$

$$\begin{bmatrix} 1 & a'_{12} & a'_{13} \\ \phi & a'_{22} & a'_{23} \\ \phi & a'_{32} & a'_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} \quad \begin{array}{l} a'_{32} = a_{32} - a_{31}a'_{12} \\ a'_{33} = a_{33} - a_{31}a'_{13} \\ b'_3 = b_3 - a_{31}b'_1 \end{array}$$

$$\begin{bmatrix} 1 & a'_{12} & a'_{13} \\ \phi & 1 & a''_{23} \\ \phi & \phi & a''_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} b'_1 \\ b''_2 \\ b''_3 \end{bmatrix} \quad \begin{array}{l} a''_{22} = a'_{22}/a'_{22} = 1;\ a''_{23} = a'_{23}/a'_{22} \\ b''_2 = b'_2/a'_{22};\ a''_{33} = a'_{33} - a'_{32}a''_{23} \\ b''_3 = b'_3 - a'_{32}b''_2 \end{array}$$

$$\begin{bmatrix} 1 & a'_{12} & a'_{13} \\ \phi & 1 & a''_{23} \\ \phi & \phi & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} b'_1 \\ b''_2 \\ b'''_3 \end{bmatrix} \quad \begin{array}{l} a'''_{33} = a''_{33}/a''_{33} = 1 \\ b'''_{33} = b''_{33}/a''_{33} \end{array}$$

Gaussian elimination is efficiently performed on a block tridiagonal matrix by applying a partial elimination process to four adjacent blocks at a time (fig. 2). The process begins with blocks 2 and 3 from the first block row and blocks 1 and 2 from the next block row. The Gaussian elimination procedure is applied to the matrix determined by these four blocks. However, the process stops once block 2 in the first block row is made upper right triangular. As a result of this process, block 1 in block row 2 is zero at this time.

This process is then repeated on the next group of blocks starting in the next block row and continuing for the whole matrix, moving the four-block template down through the matrix one block row at a time. Thus, by repeating a partial $2M$ by $2M$ Gaussian elimination N times, the tridiagonal matrix is transformed to upper right triangular form.

After the matrix has been transformed to an upper right triangular matrix, the result vector **x** can be solved by using back substitution starting from the bottom of the matrix. This is done by solving for the last element of the result vector **x**, and substituting that value into the equation for the second last element of **x** (next row up). Now, two values of the result vector are available for substitution into the equation for the third last element. The procedure is repeated one row at a time, proceeding upward through the matrix until all elements of **x** have been solved. The following equation illustrates the back-substitution process for the example problem.

$$x_3 = b'''_3$$

$$x_2 = b''_2 - a''_{23}x_3$$

$$x_1 = b'_1 - a'_{13}x_3 - a'_{12}x_2$$

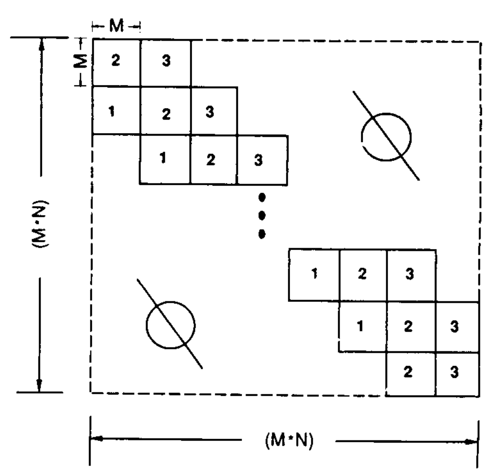
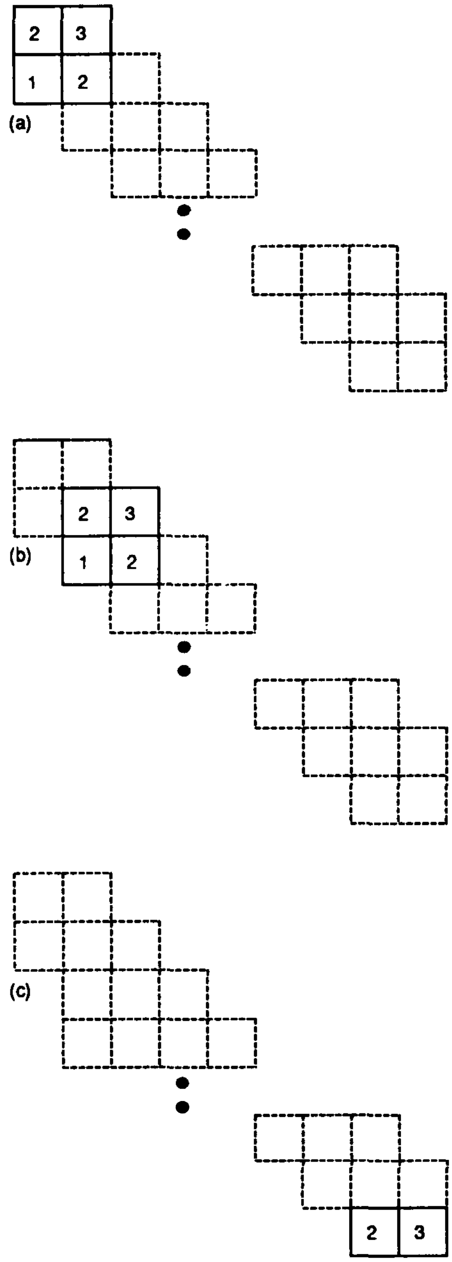


Figure 1.—Structure of block tridiagonal matrix.



(a) Step 1.
(b) Step 2.
(c) Step *N*.

Figure 2.—Gaussian elimination for block tridiagonal matrix.

The block tridiagonal matrix solver used in the rotor dynamics application consists of 30 block rows. Each block is 4 by 4. Thus, $N = 30$ and $M = 4$ for this application.

Partitioning Approach

An approach for parallelizing the Gaussian elimination procedure was developed by examining the data flow of the problem. A data flow diagram for the 3- by 3-matrix example is shown in figure 3. The circles represent mathematical operations, and the interconnections show the flow of data between calculations. For the 3- by 3-matrix example, 31 operations must be performed. A single computer can only execute these operations one at a time. The data flow diagram suggests that, if several computers are available, multiple operations could be done concurrently. The five stages of the computations are bracketed on the right side of figure 3. Within each stage, each vertical operation stream can be done in parallel. Stages 2 and 4 have streams of two operations each, while all other stages have streams of only one operation.

Most parallelism exists in the second stage, where eight operation streams can be done in parallel. If eight processors were available, the 16 operations of stage 2 could be done in a net count of two operations. Stage 1 would require four processors and could be done in a net count of one operation. Stages 3 and 4 would require three processors and could be done in net counts of one and two operations, respectively. Finally, stage 5 requires two processors and could be done in one operation. The minimum count for execution of the entire problem is the critical path. The critical path is the longest of the parallel operation streams in the data flow graph. In this example, the critical path is seven operations. Since each stage is done serially, only the maximum number of processors in any stage would be required (eight in this case).

The data flow diagram for the back-substitution process is shown in figure 4. There are two stages, and the critical path is four operations. The maximum number of processors required is two.

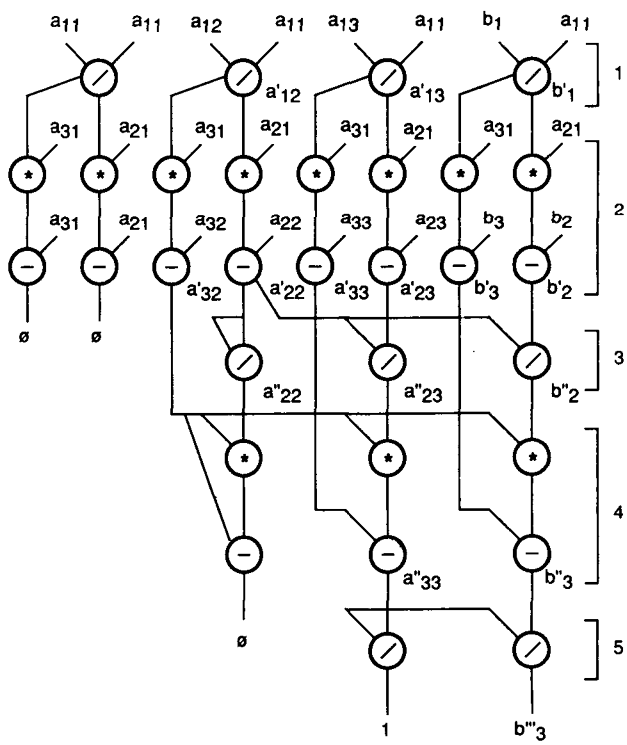


Figure 3.—Data flow diagram for Gaussian elimination (3 by 3 matrix).

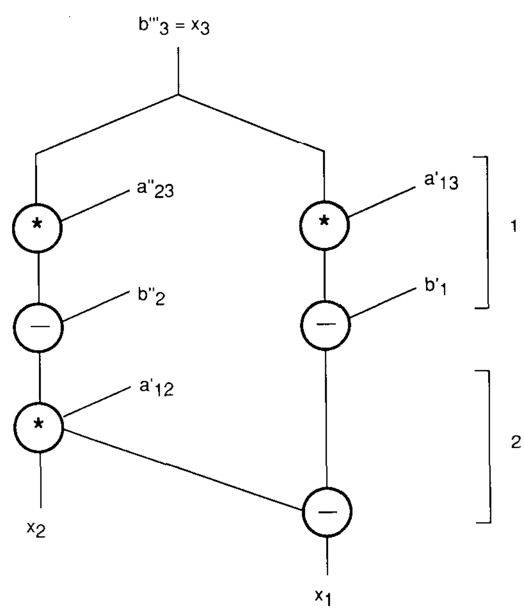


Figure 4.—Data flow diagram for back substitution (3 by 3 matrix).

The solution of the block tridiagonal matrix contains the same parallelism described for the 3- by 3-matrix example. In the solution process, a partial elimination is performed on a $2M$ by $2M$ system N times. The maximum number of processors required would be a function of M. The critical path would be N multiplied by the critical path operation count for partial Gaussian elimination plus the critical path count for the back substitution. The data flow diagram would follow the same pattern as that of the 3 by 3 matrix, only the length and width would vary as the size of the matrix. A detailed analysis is given in the Theoretical Speedup Analysis section.

A PASCAL-coded version of the single-processor matrix solver used in the rotor dynamics simulation is given in the appendix. This is a direct PASCAL translation of the FORTRAN code used in the simulation. The procedures GETINF, IDATA, and IDATF are related to I/O on the unique hardware used for this study. The purpose of these procedures is described later in this report. The parallel structures discussed previously can be seen in the main body of the code. There are two main loops in the program. The outer loop (IB) cycles through the block rows of the matrix. The next loop (IP) does the partial Gaussian elimination on the 8 by 8 submatrix composed of blocks 2 and 3 in the current block row and blocks 1 and 2 in the next block row. Within the IP loop are six smaller loops which essentially perform the operations diagramed in the data flow graph in figure 3. The first two loops perform the divide operations, and the next four loops perform the multiply and subtract operations. As shown in the data flow graph, all divides can be done in parallel followed by all multiply and subtracts being done in parallel. This process is represented by the bracketed rows 1 and 2. The sequence is repeated for all four IP iterations.

The code for the back-substitution process is next in the program. Since the original code was not written with parallel processing in mind, there are no operations which can be done in parallel while using the code shown. Each result vector element is found by solving one row at a time. Each iteration of the outermost loop (IBI) depends on results from the previous iteration. The same is true for the next level loop (II). The innermost loops within the II loop are recursive in nature (the calculation of a variable depends on itself from a previous iteration) and, therefore, cannot be done in parallel.

The data flow graph for the back substitution, however, shows that parallel operations can be done. Figure 4 shows that once an element of the result vector has been calculated, it can be used to calculate parts of proceeding elements. Thus, partial sums of other result vector elements can be computed in parallel. This algorithm, called the column sweep, is described in reference 6. The column sweep algorithm requires a different coding approach than that used in the rotor dynamics version of the back-substitution process. A new version was coded and used for the two- and three-processor matrix solvers discussed later in this paper. The use of the column sweep algorithm exemplifies the type of analysis required for selecting an algorithm to run on a parallel processor.

Theoretical Speedup Analysis

The theoretical speedup for the parallel Gaussian elimination algorithm is computed by dividing the operation count for the serial version by the net operation count for the parallel version. An operation is one of the basic floating-point math operations: add, subtract, multiply, and divide.

Table I shows the determination of the operation count for the serial algorithm for one IB iteration of the forward elimination procedure and one I iteration of the back substitution. The table assumes a 4 by 4 block size. One IB iteration consists of four IP iterations, and the operation count for each IP iteration depends on the value of IP. For a matrix of 30 block rows, the operation count (OPS) would be

$$\begin{aligned}\text{OPS} &= 30(\text{number of operations per IB}) \\ &\qquad + 30(\text{number of operations per IP}) \\ &= 30(370) + 30(44) = 12\,420 \text{ operations}\end{aligned}$$

To simplify the analysis, it is assumed that the last block row is a full 8 by 8 matrix, although it is actually 4 by 8.

The operation count for an N block row, M- by M-block tridiagonal matrix would be

$$\text{OPS} = N\left\{\sum_{i=1}^{M}[(2M+2-i) + 2(2M+2-i)(2M-i) + 2(M+i-1)]\right\}$$

$$= \frac{N[M(4M+7)(7M-1)]}{6}$$

TABLE I.—DETERMINATION OF OPERATION COUNT

(a) Gaussian elimination

Loop, IP	Operations			Total number of operations
	Divide	Multiply	Subtract	
1	9	63	63	135
2	8	48	48	104
3	7	35	35	77
4	6	24	24	54
--	--	--	--	370

(b) Back substitution

Loop, I	Operations		Total number of operations
	Multiply	Subtract	
1	4	4	8
2	5	5	10
3	6	6	12
4	7	7	14
--	--	--	44

The data flow graphs in figures 3 and 4 suggest that a number of operations can be done in parallel. For the forward elimination process, the total number of operations which can be done in parallel is a function of the iteration index IP. Table II summarizes the maximum number of operations which can be performed in parallel as a function of IP. The last column shows the net operation count for each IP iteration (three) if there are enough processors available to match the number of operations that can be done in parallel. Each IP iteration consists of a parallel divide cycle, followed by a parallel multiply and subtract cycle. The net operation count is one for the divide cycle and two for the multiply and subtract cycle. Each IP iteration has three operations. As IP increases, the number of processors that can be used decreases.

Table II also shows the maximum number of parallel operations for each I iteration of the back-substitution process. Again, the net operation count is shown for the case where the number of processors matches the number of parallel operations. Based on the total operation count for the fully parallel forward elimination and back-substitution processes (assuming 30 block rows), the total operation count would be

$$\begin{aligned} \text{OPS} &= 30(4\times 3 \text{ operations}) + 30(4\times 2 \text{ operations}) \\ &= 600 \text{ operations} = 30\times 4\times 5 \end{aligned}$$

or, in general,

$$\text{OPS} = N\times M\times 5 \text{ operations}$$

For the matrix used in this study, the theoretical speedup (S) would be

$$S = \frac{12\ 420}{600} = 20.7$$

and, in general,

$$S = \frac{\sum_{i=1}^{M} (2M+2-i) + 2(2M+2-i)(2M-i) + 2(M+i-1)}{5M} = \frac{(4M+7)(7M-1)}{30}$$

for a N block row, M- by M-block matrix.

The theoretical speedup would be achieved if the maximum number of processors (63 as determined from table II) are available to perform the computations. Any overhead due to inefficient resource allocation (discussed in the next section) or communication between processors has been ignored. This simplification is made because of the difficulty in estimating the time required for such overhead. The theoretical speedup is useful only as an upper limit to determine if parallel processing can potentially benefit an application.

Determining the theoretical speedup is more complicated when less than the maximum number of processors is available. The speedup will also be a function of the way the parallel computations are allocated to the processors. For example, if there are four parallel operations and three processors, the net operation count would be two because the fourth operation must be done in serial on one of the three processors. The theoretical speedup for the three-processor matrix solver was determined to be 2.9 based on the best resource allocation possible.

TABLE II.—DETERMINATION OF PARALLEL OPERATION COUNT

(a) Forward elimination

Loop, II	Number of processors	Net operation count
1	63	3
2	48	3
3	35	3
4	24	3
--	--	12

(b) Back substitution

Loop, II	Number of processors	Net operation count
1	4	2
2	5	2
3	6	2
4	7	2
--	--	8

Resource Allocation

Allocating processor resources is a critical step in running any code on a parallel processor. If the processor resources (e.g., the number of processors) match the number of parallel tasks in a problem, then a one-to-one allocation can be done. This approach is not always efficient, however, as processors can spend much time in an idle state. In some cases this inefficiency is unavoidable. In others, a "packing" algorithm can be used to assign the parallel tasks to the minimum number of processors necessary. If the processor resources do not match the number of parallel tasks, then a packing algorithm is a necessity. Ideally, an automated procedure would assign the parallel computations to the available processors and generate the appropriate load modules (to execute on the processors). Such a procedure, unfortunately, was not available for this study.

A technique for allocating the parallel operations of the matrix solver to the appropriate processors was necessary. One, called the loop-unrolling technique, would require decomposing the loops into individual equations. For example, consider the following loop:

```
FOR I: = 1 to 5 DO
  FOR J: = I to 5 DO
    A(I,J): = B(I,J) * C(I,J);
```

The doubly nested loop can be decomposed into 15 equations, and each of these could be executed in parallel. Suppose that only three processors were available for the solution of these equations. One method of allocating the equations to the processors would be to write all 15 equations and allocate 5 equations to each processor. Although this apears easy for the given example, it can be tedious if there are hundreds of thousands of equations. Another, less tedious, method would be to use the following code segment on each of the processors:

```
FOR I: = 1 to 5 DO
  BEGIN
    J: = (I - 1) + PID;
    WHILE J <= 5 DO
      BEGIN
        A(I,J): = B(I,J) * C(I,J);
        J: = J + NPROC;
      END;
  END;
```

where PID is the processor identification number (in this case 1, 2, or 3) and NPROC is the number of processors (three for this example). In this method, called iteration allocation, each processor performs only the iterations which are assigned to it. The preceding example results in the allocation of computations as follows:

	P1	P2	P3
	A(1,1)	A(1,2)	A(1,3)
	A(1,4)	A(1,5)	A(2,4)
	A(2,2)	A(2,3)	A(3,5)
	A(2,5)	A(3,4)	-----
	A(3,3)	A(4,5)	-----
	A(4,4)	-----	-----
	A(5,5)	-----	-----
Total OPS	7	5	3

With 3 processors and this allocation method, the original 15 operations could be done in the equivalent of 7 operations. Although this method is less efficient than writing 15 separate equations, it is less tedious. In fact, by adding the following lines of code before the J: = (I − 1) + PID line in processors 1 and 3, the allocation can be improved:

P1	P2
IF I > 3 THEN PID = 3	IF I > 3 THEN PID = 1
ELSE PID = 1;	ELSE PID = 3;

This reallocates the A(4,4) and A(5,5) computations from processor 1 to processor 3. Now each processor solves five equations for a net count of five operations. However, this analysis ignores the overhead of the added control statements. Thus, iteration allocation is still less efficient than the loop-unrolling technique. But for large loops, iteration allocation is preferable since it is less tedious.

The number of processors available is a critical factor in considering which method to use. If the number of processors approaches the number of parallel tasks, then the iteration-allocation method essentially approaches the loop-unrolling technique (in the amount of work necessary to generate a parallel program). In general, if the number of parallel tasks is much greater than the number of processors, iteration allocation is preferable to loop-unrolling. This was the case for the parallel block tridiagonal solver described in this report, which made iteration allocation the method of choice.

Parallel Processing Hardware Description

The parallel processing hardware system used to run the block tridiagonal solver is a subset of the real-time multiprocessor simulator (RTMPS) described in reference 2. Figure 5 is a block diagram of the actual hardware used. The separate processors on the real-time bus are Motorola VM04 microcomputers, rather than the VM02 microcomputers used on the original RTMPS. The RTX channel linking the interactive and real-time busses still uses VM02 microcomputers. In the current configuration, a maximum of three VM04

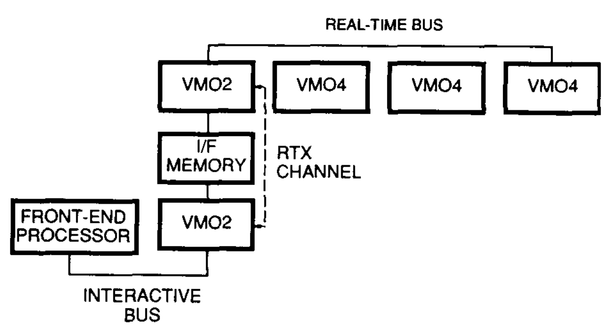


Figure 5.—Subset real-time multiprocessor system (RTMPS) architecture used for matrix solver study.

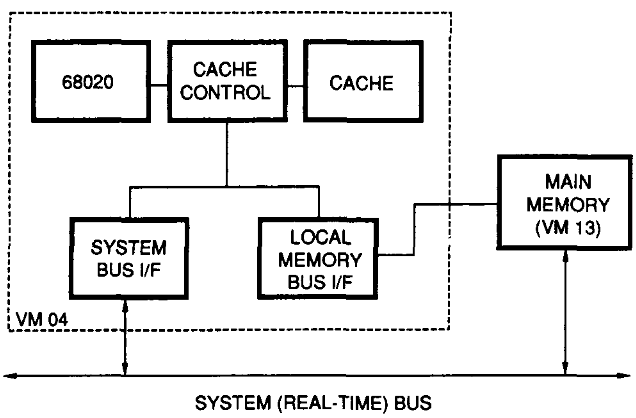


Figure 6.—VMO4 microcomputer architecture.

processors can be resident on the real-time bus. Expansion to two additional processors is possible with the existing card cage, but was not done for this study.

Figure 6 shows the architecture of the VM04 microcomputer. A dedicated memory bus connects a processor board to a main memory board. Both the processor board and the memory board have separate system bus interfaces. A high-speed cache memory on the processor board reduces memory access times for frequently referenced memory locations. The caching of main memory is handled by hardware and is transparent to the user when only a single processor is used.

Because of the cache memory, extra care must be taken when programming multiprocessor systems. A processor can have instances of "stale data" in its cache if another processor communicates with it through shared memory over the system bus. To avoid this problem, processors must disable their cache memories before accessing shared memory. One method, although time consuming, is to call a procedure to disable the cache each time a program requires access to shared memory. Another method would be to disable the cache entirely; however, there are many local-memory accesses which would lose the benefit of the faster cache memory. Fortunately, the VM04 contains a control register which allows the user to enable or disable caching of memory accesses that occur over the system bus. This register can be set once, and cache memory can be disabled for all shared-memory references (via the system bus), while local-memory references are still cached.

Software Environment

The existing RTMPL was designed to efficiently handle one-dimensional mathematical models. All arithmetic is done in scaled-fractions, and indexed variables (e.g., arrays) are not supported. An alternative language was needed to allow convenient programming of the block tridiagonal solver on the RTMPS hardware. To fill this need, a method was devised to allow PASCAL programs to be called from an RTMPL program. The the solver could be coded in the PASCAL language, a structured language with floating-point and indexed variable support.

Running the PASCAL program as a subroutine under RTMPL maintains compatibility with the RTMPOS. This is important because RTMPL generates a data base that RTMPOS uses to load and execute the parallel processing programs. Changes were made to RTMPOS to allow recognition of the floating-point data type. Thus, many interactive features provided by RTMPOS for scaled-fraction programs could also by applied to floating-point programs.

An RTMPL macro was written to transfer control from RTMPL to PASCAL. A new PASCAL initialization routine (ref. 9) was written to save any necessary RTMPL registers, execute the PASCAL program, restore the RTMPL registers, and return to the RTMPL program. RTMPL variables were used as buffers to transfer information from the PASCAL program to the RTMPL program. Special procedures were written to do the transfers. This represents one of the disadvantages of the RTMPL-PASCAL approach: Neither program recognizes the variables of the other. In order to output any results from the PASCAL program to RTMPOS, data must explicitly be transferred from a PASCAL variable to an RTMPL variable. This inconvenience can translate into high overhead if data is output frequently from the PASCAL program. Fortunately, for the block tridiagonal solver, the only output required was at the end of the program.

The automated data-transfer setup feature of the RTMPL cannot be used with the RTMPL-PASCAL approach. All data transfers must be done from within the PASCAL program. One method for transferring data from PASCAL is to call a procedure to do the transfer. However, if there is frequent data transfer in the program, the overhead of the procedure call can significantly reduce the transfer speed. A better method is to exploit the way that PASCAL handles variables. Variables declared in the main PASCAL program are global variables; variables declared from within a procedure are local to that procedure. Global variables are shared by the main program and all procedures. This suggests that a shared-memory multiprocessor environment can be implementaed by using the global variable area as the shared memory. The advantage of a shared-memory approach is that data can be transferred implicitly between processors by a simple memory reference instruction. The need for a procedure call to transfer data is eliminated, thus, reducing overhead.

Figure 7 shows how the PASCAL shared-memory approach is implemented for two processors connected by a bus. The PASCAL compiler maintains two registers for variable storage. The first (A5) points to the base of the global variables. The other register (A6) points to the base of the local variable area. If both processors (P1,P2) have dual-ported memory, part of the memory of P1 can be shared with P2. The PASCAL programs for P1 and P2 would have the shared-memory variables declared first. The program code body would be contained in a procedure call, with any local variables declared within the procedure. Then the main program would merely call this procedure. The structure of the PASCAL program for both processors would be as follows:

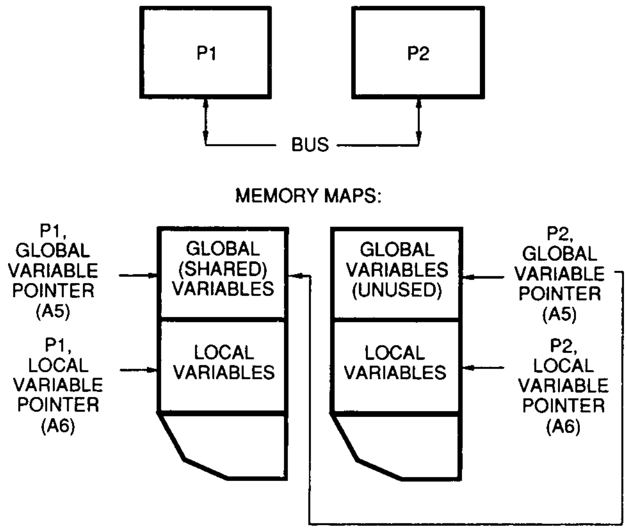


Figure 7.—PASCAL shared-memory approach.

```
PROGRAM SOLVE;
  VAR
    (This is the declaration of shared-memory variables)
PROCEDURE SOLVE__CODE;
  VAR
    (This is the declaration of variables local to the processor)
  BEGIN
    (The main code body goes here)
  END; (Of SOLVE__CODE)
BEGIN (Of the main program)
  SOLVE__CODE;
END. (Of program)
```

Since P1 has the shared-memory area in its own memory, its global variable register A5 can be left as set by the PASCAL compiler. For P2 to reference the shared-memory area, its register A5 must have the base address of P1's memory (from the bus) added to it. When this is done, all global variable accesses set up by the compiler will automatically go to shared memory. This approach can be used for as many processors as required.

All PASCAL language statements except those dealing with I/O, files, and pointers can be used. All I/O is done through the facilities provided by RTMPL and RTMPOS. These facilities include on-line examination of program variables and read advisories. The read advisory provides a method for recording large arrays of data from a program onto a disk file (RTMPL user's manual). This method was used for the block tridiagonal solver to record the value of the result vector.

Discussion of Results

The block tridiagonal solver was run on the RTMPS system with one, two, and three processors. The PASCAL code for each of the cases is contained in the appendix of this report. The matrix notation used for the rotor dynamics problem is retained in this code. Array B in the PASCAL code is the matrix of coefficients, the array C is the right-side vector, and the array DU is the result vector. The first VAR declaration is the global, or shared-memory area. A multiply indexed array is used for the block tridiagonal matrix. The first two indices (from left to right) are the row and column indices within a block. The next index is the block row index, and the last index is the block index (1, left; 2, middle; 3, right). The vectors DU and C are doubly indexed arrays: The first index indicates element within the current block row, and the second index is the block row index. Although the use of multiple indices simplifies the programming procedure, it is very costly in computation time.

The code for the single-processor solver is a direct PASCAL translation of the FORTRAN code used in the rotor dynamics problem. Procedure GETINF is used to send information about the variables (in this case, the result vector) to the RTMPS control processor. Calling procedure GETINF triggers a read advisory on the control processor which saves results in a disk file. Procedures IDATA and IDATF initialize the matrix and right-side vector to values that were generated by the rotor dynamics simulation. The use of actual data from the rotor dynamics simulation was important since the existence and accuracy of a matrix solution depends heavily on the matrix values. The results generated by the single-processor solver, as well as those for the two- and three-processor solvers, were compared to results generated by the rotor dynamics simulation on a mainframe computer. In all cases, the results matched exactly.

There are two versions of the two-processor solver given in the appendix: The first contains the original serial back-substitution algorithm; the other does the back substitution by using the column sweep approach. In both versions, the forward elimination process is done in parallel, and iterations within the IP loop are allocated to each processor. This is done with the WHILE-DO construct, as described in the Resource Allocation section of this report. Before each IP iteration begins, both processors synchronize to insure that the previous IP iteration was completed. This is critical since results from the previous iteration are needed to calculate the next iteration. Two boolean flags, one for each processor, are used to synchronize the processors. The flags are located in the global, or shared-memory, area. Both processors set their respective flags true after they have finished an IP iteration.Before starting the next iteration, each processor checks the other's flag to make sure they are synchronized. Then both flags are cleared, and the iteration can begin. If one processor is not done, the other will wait for it. A counter is tested to exit the wait loop if the other processor does not respond.

The version of the two-processor solver with the column-sweep back-substitution algorithm differs from the serial back substitution version in two ways: (1) The synchronization is done with an assembly language procedure to decrease its execution time. The assembly procedure performs exactly the same function as the original PASCAL version of the procedure (which is commented out in the listing); and (2) the back-substitution process, previously done on one processor, is now done on two processors. After an element of the result vector is computed, both processors work on computing partial results of other vector elements. Both processors then synchronize, compute the next full result vector element, and repeat the process until the entire result vector is obtained.

The code for the three-processor version of the solver uses a synchronization method which is more efficient than that used in the two-processor case. When each processor is done with its iteration, it sends a flag to each of the other processors in the system. Before starting the next iteration, each processor tests for the flags sent to it by the other processors. Since these flags are now in local memory (not global memory as in the two-processor case), the processor does not have to continually access the bus to test a flag. This reduces bus traffic for those processors which may still be accessing shared memory to complete their computations.

Another technique used in the three-processor solver to reduce bus traffic is the copying of frequently accessed variables from shared memory to local memory. In the three-processor code, arrays BI2 and BI3 are local-memory variables which contain current matrix row information used frequently throughout the program. These arrays are loaded with appropriate values from shared memory at the beginning of an IP iteration. All future references to these values are made from local memory, and the number of bus accesses required is reduced.

Table III summarizes the running time for each of the three cases. The speedup for each of the multiprocessor runs is also shown. A 30-block row matrix, with 4 by 4 blocks, was solved in each case. For the two-processor case, results are given for the serial back substitution and for the column-sweep back-substitution algorithms. As expected, the column sweep algorithm gives a faster solution. The two-processor case shows a speedup for 1.96, very close to the ideal value of 2. The three-processor case is less efficient with a speedup of 2.7. The reduction in efficiency can be attributed to a number of factors: Resource allocation, loss of cache variables, and increased access time for the shared memory because of increased bus traffic.

TABLE III.—TIMING INFORMATION FOR MULTIPROCESSOR

Number of processors	Back substitution	Time, sec	Speedup
1	Column sweep	0.9502	-----
2	Serial Column sweep	0.5168 .4834	1.838 1.965
3	Column sweep	0.3500	2.715

Several important notes are given here regarding cache memory. All multiprocessor runs were made with the cache memory enabled on all processors which did not contain the shared memory. The processor which did contain the shared memory had its cache disabled. This processor could not take advantage of the control register cache disabling for bus accesses (described in the hardware section) since all variables are physically within its own memory. The single-processor case used as the reference for speedup calculations was run with cache memory enabled. Variables which are in shared memory for the multiprocessor cases (and, hence, not cached) can be cached in the single-processor case. Thus, a certain percentage of the speedup achieved through parallel processing can be offset by the loss of cache variables. Although it appears that this is not a factor in the two-processor case, it may account for some of the overhead in the three-processor case.

A synchronization problem was encountered during the development of the three-processor solver which highlighted one of the difficulties with transporting existing algorithms (written for serial processors) to parallel processors. In the Gaussian elimination process, before elements below the diagonal are eliminated the original values are needed to compute other elements of the matrix. Thus, the sequence of the computations is critical. All processors would have to be synchronized (in addition to the synchronization that must be done for each IP iteration) to insure that the original value of the element being eliminated has been used by the other processors needing it. For example, consider the following elimination step for a 3 by 3 matrix:

(1) BR: = A(2,1);

(2) A(2,1): = A(2,1) − BR * A(1,1)/A(1,1);

(3) A(2,2): = A(2,2) − BR * A(1,2)/A(1,1);

(4) A(2,3): = A(2,3) − BR * A(1,3)/A(1,1);

(5) F(2): = F(2) − BR * F(1)/A(1,1);

where A is the array of matrix elements and F is the right-side vector. In statement (1), BR is assigned the value of A(2,1), and the computation of A(2,1) in statement (2) will result in zero. Statement (1) is antidependent on statement (2) (ref. 10). Assume that four processors are available to do statements (2) through (5) with all data resident in a shared memory (except for BR which is in each processor's local memory). Each processor must execute the assignment statement which copies the value for A(2,1) from shared memory into local variable BR. It would appear, since each processor performs the same number of operations, that each processor could safely read A(2,1) before it is changed by processor 1. This also assumes that all processors begin their operations at the same time. However, timing differences between processors, communication delays between processors and shared memory, and load imbalances make this assumption dangerous. This was the case for the three-processor version of the block tridiagonal solver as it was derived from the original FORTRAN code used in the rotor dynamics simulation. Synchronization routines were necessary which added overhead and resulted in slower execution.

The addition of synchornization routines for this part of the code can be avoided, however, by examining the Gaussian elimination process closer. The back-substitution process only requires elements above the diagonal to compute the result vector. The zeros below the diagonal are not needed. In fact, the computations which create the zeros are not necessary if the resulting upper right triangular matrix is only needed to compute the result vector. If the A(2,1) calculation was eliminated in the previous example, each processor would read the correct value of A(2,1) without synchronization problems. This approach was taken for the three-processor solver to achieve the speedup of 2.7. The single processor time used in the speedup calculation includes the computation of zero elements below the diagonal. If these computations are removed from the single-processor code also, then the relative speedup is reduced to 2.49. This is because the single-processor solver has fewer computations to do and, thus, runs faster.

Concluding Remarks

An approach to implementing a block tridiagonal matrix solver on a shared-memory parallel processor has been demonstrated. It should be possible to run the PASCAL programs for the one-, two-, and three-processor solvers on other shared-memory parallel processors, if the I/O and synchronization procedures are reproduced on the target system. The same approach can also be extended to more processors if they are available.

The results presented here are only a small part of the potential research that can be done on parallel processing of matrix solvers and solution of partial differential equations in general. Alternative architectures exist which have the potential for providing extremely fast matrix solutions. Architectures incorporating multiple array or vector processors, such as the ALLIANT FX/8 or CRAY X-MP, are examples. A pipelined math unit can perform operations much faster than the nonpipelined units found in typical microcomputers and mainframes. The key to tapping the potential of these architectures is the identification of at least two levels of parallelism in a given problem. The first is the operation level, which corresponds to the vectorization process done for single vector processors. The second is the vector operation level. Parallelism here consists of multiple vector operations which can be done concurrently.

Another high-potential research area is the investigation of alternative algorithms, given an architecture which can exploit the natural parallelism in the algorithm. There are many highly parallel iterative algorithms for solving systems of equations. Among these are successive overrelaxation methods (SOR) and conjugate gradient methods. Given an appropriate architecture, these methods could potentially yield higher performance than the Gaussian elimination method.

The selection of an appropriate algorithm for solving any problem on a parallel processor is a function of many parameters. NASA Lewis Research Center is currently constructing a hypercluster system to provide a test bed for investigating architecture and algorithm interactions (ref. 10). The combination of multiple vector and scalar processors in a flexible interconnection scheme will allow a wide variety of architectural concepts to be studied. It is hoped that future work using the hypercluster will answer some of the questions regarding appropriate architecture and algorithm combinations for both computational fluid mechanics and computational structural mechanics problems.

Lewis Research Center
National Aeronautics and Space Administration
Cleveland, Ohio, November 17, 1988

Appendix - PASCAL Program Listings

Single-Processor Block Tridiagonal Solver

```
       0) 0)-- PROGRAM SOLVE;
       0) 0)--
       0) 0)-- TYPE
       0) 0)--
       0) 0)-- RVECT=ARRAY [1..4,1..32] OF REAL;
       0) 0)-- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
       0) 0)--
       0) 0)-- VAR
       0) 0)--
  -12288) 0)--   A,B             :   AMAT;
  -13824) 0)--   F,C,DU          :   RVECT;
  -13832) 0)--   BP,BR           :   REAL;
  -13856) 0)--   N,I,J,IB,IP,I1  :   INTEGER;
  -13876) 0)--   I2,I3,II1,K,J1  :   INTEGER;
  -13892) 0)--   II,JB,IB1,IBI   :   INTEGER;
  -13892) 0)--
       0) 1)-- PROCEDURE GETINF( VAR ADDR : RVECT; NUMEL : INTEGER );FORWARD;
       0) 1)--
       0) 1)-- PROCEDURE IDATA(  VAR MATRXA : AMAT; VNUM : INTEGER );FORWARD;
       0) 1)-- PROCEDURE IDATF(  VAR MATRXF : RVECT; VNUM2 : INTEGER );FORWARD;
       0) 1)--
**** IDATF     ASSUMED EXTERNAL
**** IDATA     ASSUMED EXTERNAL
**** GETINF    ASSUMED EXTERNAL
         1  0)A- BEGIN
         2  0)--   II1:= 1;
         3  0)--   I2:=2;
         4  0)--   I3:=3;
         5  0)--   N:=30;
            0)--
         6  0)--   IDATA( B,1536 );
         7  0)--   IDATF( C,128 );
            0)--
         8  0)--   FOR IB:= 1 TO N DO
         9  0)--     FOR IP:= 1 TO 4 DO
            0)B-       BEGIN
        10  0)--         BP:= B[ IP,IP,IB,I2 ];
        11  0)--         FOR J:= IP TO 4 DO
        12  0)--           B[ IP,J,IB,I2 ]:= B[ IP,J,IB,I2 ] / BP;
        13  0)--         FOR J:= 1 TO 4 DO
        14  0)--           B[ IP,J,IB,I3 ]:= B[ IP,J,IB,I3 ] / BP;
        15  0)--         C[ IP,IB ]:= C[ IP,IB ] /BP;
        16  0)--         IF IP <> 4 THEN
            0)C-           BEGIN
        17  0)--             I1:= IP + 1;
        18  0)--             FOR I:= I1 TO 4 DO
            0)D-               BEGIN
        19  0)--                 BR:= B[ I,IP,IB,I2 ];
        20  0)--                 FOR J:= IP TO 4 DO
        21  0)--                   B[ I,J,IB,I2 ]:= B[ I,J,IB,I2 ] - BR * B[ IP,J,IB,I2 ];
        22  0)--                 FOR J:= 1 TO 4 DO
        23  0)--                   B[ I,J,IB,I3 ]:= B[ I,J,IB,I3 ] - BR * B[ IP,J,IB,I3 ];
        24  0)--                 C[ I,IB ]:= C[ I,IB ] - BR * C[ IP,IB ];
            0)-D               END; [ FOR I]
            0)-C           END; [IF IP]
        25  0)--         IF IB <> N THEN
            0)C-           BEGIN
        26  0)--             FOR I:= 1 TO 4 DO
            0)D-               BEGIN
        27  0)--                 IB1:= IB + 1;
        28  0)--                 BR:= B[ I,IP,IB1,II1];
        29  0)--                 FOR J:= IP TO 4 DO
        30  0)--                   B[ I,J,IB1,II1 ]:= B[ I,J,IB1,II1 ] - BR * B[ IP,J,IB,I2 ];
        31  0)--                 FOR J:= 1 TO 4 DO
        32  0)--                   B[ I,J,IB1,I2 ]:= B[ I,J,IB1,I2 ] - BR * B[ IP,J,IB,I3 ];
        33  0)--                 C[ I,IB1 ]:= C[ I,IB1 ] - BR * C[ IP,IB ];
            0)-D               END; [FOR I]
            0)-C           END; [ IF IB]
            0)-B       END; [ FOR IP]
```

```
     34  0)---    FOR IBI:= 1 TO N DO
         0)B--      BEGIN
     35  0)---        IB:= N + 1 - IBI;
     36  0)---        IB1:= IB + 1;
     37  0)---        FOR II:= 1 TO 4 DO
         0)C--          BEGIN
     38  0)---            I:= 5 - II;
     39  0)---            DU[ I,IB ]:= -1.0 * C[ I,IB ];
     40  0)---            IF I <> 4 THEN
         0)D--              BEGIN
     41  0)---                J1:= I + 1;
     42  0)---                FOR J:= J1 TO 4 DO
     43  0)---                  DU[ I,IB ]:= DU[ I,IB ] - B[ I,J,IB,I2 ] * DU[ J,IB ];
         0)-D               END; [ IF I <> 4 ]
     44  0)---            IF IB <> N THEN
         0)D--              BEGIN
     45  0)---                FOR J:= 1 TO 4 DO
     46  0)---                  DU[ I,IB ]:= DU[ I,IB ] - B[ I,J,IB,I3 ] * DU[ J,IB1 ];
         0)-D               END; [ IF IB <> N ]
         0)-C           END; [ FOR II ]
         0)-B       END; [ FOR IBI ]
     47  0)---    K:=128;
     48  0)---    GETINF( DU,K );
         0)-A END.
```

```
**** NO ERROR(S) AND NO WARNING(S) DETECTED

**** 90 LINES 3 PROCEDURES

**** 944 PCODE INSTRUCTIONS
```

Dual-Processor Block Tridiagonal Solver; Serial Back Substitution, Processor 1

```
       0)  0)--- PROGRAM SOLVE;
       0)  0)---
       0)  0)--- TYPE
       0)  0)---
       0)  0)--- RVECT=ARRAY [1..4,1..32] OF REAL;
       0)  0)--- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
       0)  0)---
       0)  0)--- VAR
       0)  0)---
   -6144)  0)---    B                 :    AMAT;
   -7168)  0)---    C,DU              :    RVECT;
   -7170)  0)---    SYNC1,SYNC2       :    BOOLEAN;
   -7170)  0)---
       0)  1)--- PROCEDURE GETINF( VAR ADDR:RVECT; NUMEL:INTEGER );FORWARD;
       0)  1)---
       0)  1)--- PROCEDURE PUTINT( IVAL:INTEGER; VAR IPTR:INTEGER );FORWARD;
       0)  1)---
       0)  1)--- PROCEDURE IDATA( VAR MATRIXA : AMAT; VCNT : INTEGER );FORWARD;
       0)  1)---
       0)  1)--- PROCEDURE IDATF( VAR MATRIXF : RVECT; VCNT1 : INTEGER ); FORWARD;
       0)  1)---
       0)  1)--- PROCEDURE COPROC ;
       0)  1)---
       0)  1)--- CONST
       0)  1)---
       0)  1)---    CMAX=1000000;
       0)  1)--- VAR
       0)  1)---
   -6144)  1)---    A                   :    AMAT;
   -6656)  1)---    F                   :    RVECT;
   -6664)  1)---    BP,BR               :    REAL;
   -6688)  1)---    N,I,J,IB,IP,I1      :    INTEGER;
   -6708)  1)---    I2,I3,II1,K,J1      :    INTEGER;
   -6724)  1)---    II,JB,IB1,IBI       :    INTEGER;
   -6736)  1)---    ERR,SCNT1,IPTR      :    INTEGER;
   -6736)  1)---
   -6736)  1)---
       1   1)A-- BEGIN
       2   1)---    SYNC1:=FALSE;
```

```
 3  1)---  IPTR:=0;
 4  1)---  SCNT1:=0;
 5  1)---  ERR:=0;
 6  1)---  II1:= 1;
 7  1)---  I2:=2;
 8  1)---  I3:=3;
 9  1)---  N:=30;
    1)---
10  1)---  IDATA( B,1536 );
11  1)---  IDATF( C,128 );
    1)---
12  1)---  SYNC1:=TRUE;
    1)---
13  1)---  FOR IB:= 1 TO N DO
14  1)---    FOR IP:= 1 TO 4 DO
    1)B--      BEGIN
    1)---
15  1)---        SCNT1:=0;
    1)C--        REPEAT                        [ SYNCHRONIZE WITH PROCESSOR 2 ]
16  1)---          SCNT1:=SCNT1+1;
17  1)---          IF SCNT1 > CMAX THEN
    1)D--            BEGIN
18  1)---              SYNC2:=TRUE;
19  1)---              ERR:=ERR + 1;
    1)--D            END;
20  1)--C        UNTIL SYNC2;
21  1)---        SYNC2:= FALSE;
    1)---
22  1)---        BP:= B[ IP,IP,IB,I2 ];
23  1)---        J:= IP;
24  1)---        WHILE J <= 4 DO
    1)C--          BEGIN
25  1)---            B[ IP,J,IB,I2 ]:= B[ IP,J,IB,I2 ] / BP;
26  1)---            J:=J+2;
    1)--C          END;
27  1)---        J:= 1;
28  1)---        WHILE J <= 4 DO
    1)C--          BEGIN
29  1)---            B[ IP,J,IB,I3 ]:= B[ IP,J,IB,I3 ] / BP;
30  1)---            J:= J + 2;
    1)--C          END;
31  1)---        C[ IP,IB ]:= C[ IP,IB ] /BP;
    1)---
    1)---
32  1)---        IF IP <> 4 THEN
    1)C--          BEGIN
33  1)---            I1:= IP + 1;
34  1)---            FOR I:= I1 TO 4 DO
    1)D--              BEGIN
35  1)---                BR:= B[ I,IP,IB,I2 ];
36  1)---                J:= IP;
37  1)---                WHILE J <= 4 DO
    1)E--                  BEGIN
38  1)---                    B[ I,J,IB,I2 ]:= B[ I,J,IB,I2 ] - BR * B[ IP,J,IB,I2 ];
39  1)---                    J:= J + 2;
    1)--E                  END;
40  1)---                J:= 1;
41  1)---                WHILE J <= 4 DO
    1)E--                  BEGIN
42  1)---                    B[ I,J,IB,I3 ]:= B[ I,J,IB,I3 ] - BR * B[ IP,J,IB,I3 ];
43  1)---                    J:= J + 2;
    1)--E                  END;
44  1)---                C[ I,IB ]:= C[ I,IB ] - BR * C[ IP,IB ];
    1)--D              END; [ FOR I]
    1)--C          END; [IF IP]
45  1)---        IF IB <> N THEN
    1)C--          BEGIN
46  1)---            FOR I:= 1 TO 4 DO
    1)D--              BEGIN
47  1)---                IB1:= IB + 1;
48  1)---                BR:= B[ I,IP,IB1,II1];
49  1)---                J:= IP;
50  1)---                WHILE J <= 4 DO
    1)E--                  BEGIN
51  1)---                    B[ I,J,IB1,II1 ]:= B[ I,J,IB1,II1 ] - BR * B[ IP,J,IB,I2 ];
52  1)---                    J:= J + 2;
```

```
        1)-E                     END;
  53    1)--                   J:= 1;
  54    1)--                   WHILE J <= 4 DO
        1)E-                     BEGIN
  55    1)--                       B[ I,J,IB1,I2 ]:= B[ I,J,IB1,I2 ] - BR * B[ IP,J,IB,I3 ];
  56    1)--                       J:= J + 2;
        1)-E                     END;
  57    1)--                   C[ I,IB1 ]:= C[ I,IB1 ] - BR * C[ IP,IB ];
        1)-D                 END; [FOR I]
        1)-C             END; [ IF IB]
  58    1)--         SYNC1:=TRUE;
        1)-B       END; [ FOR IP]
        1)--
  59    1)--       SCNT1:=0;
        1)B-       REPEAT                     [ SYNCHRONIZE WITH PROCESSOR 2 ]
  60    1)--         SCNT1:=SCNT1+1;
  61    1)--         IF SCNT1 > CMAX THEN
        1)C--          BEGIN
  62    1)--             SYNC2:=TRUE;
  63    1)--             ERR:= ERR + 1;
        1)-C           END;
  64    1)-B       UNTIL SYNC2;
        1)--
  65    1)--     FOR IBI:= 1 TO N DO
        1)B-       BEGIN
  66    1)--         IB:= N + 1 - IBI;
  67    1)--         IB1:= IB + 1;
  68    1)--         FOR II:= 1 TO 4 DO
        1)C-           BEGIN
  69    1)--             I:= 5 - II;
  70    1)--             DU[ I,IB ]:= -1.0 * C[ I,IB ];
  71    1)--             IF I <> 4 THEN
        1)D-               BEGIN
  72    1)--                 J:= I + 1;
  73    1)--                 WHILE J <= 4 DO
        1)E-                   BEGIN
  74    1)--                     DU[ I,IB ]:= DU[ I,IB ] - B[ I,J,IB,I2 ] * DU[ J,IB ];
  75    1)--                     J:= J + 1;
        1)-E                   END;
        1)-D               END; [ IF I <> 4 ]
  76    1)--             IF IB <> N THEN
        1)D-               BEGIN
  77    1)--                 J:= 1;
  78    1)--                 WHILE J <= 4 DO
        1)E-                   BEGIN
  79    1)--                     DU[ I,IB ]:= DU[ I,IB ] - B[ I,J,IB,I3 ] * DU[ J,IB1 ];
  80    1)--                     J:=J + 1;
        1)-E                   END;
        1)-D               END; [ IF IB <> N ]
        1)-C           END; [ FOR II ]
        1)-B       END; [ FOR IBI ]
  81    1)--     PUTINT( ERR,IPTR );
  82    1)--     PUTINT( SCNT1,IPTR );
  83    1)--     K:=128;
  84    1)--     GETINF( DU,K );
        1)-A END;  [ COPROC ]
        1)--
**** IDATF     ASSUMED EXTERNAL
**** IDATA     ASSUMED EXTERNAL
**** PUTINT    ASSUMED EXTERNAL
**** GETINF    ASSUMED EXTERNAL
  85    0)A-- BEGIN
  86    0)--     COPROC;
        0)-A END.

**** NO ERROR(S) AND NO WARNING(S) DETECTED

**** 175 LINES 5 PROCEDURES

**** 1076 PCODE INSTRUCTIONS
```

Dual-Processor Block Tridiagonal Solver; Serial Back Substitution, Processor 2

```
    0)  0)-- PROGRAM SOLVE;
    0)  0)--
    0)  0)-- TYPE
    0)  0)--
    0)  0)-- RVECT=ARRAY [1..4,1..32] OF REAL;
    0)  0)-- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
    0)  0)--
    0)  0)-- VAR
    0)  0)--
-6144)  0)--   B                  :   AMAT;
-7168)  0)--   C,DU               :   RVECT;
-7170)  0)--   SYNC1,SYNC2        :   BOOLEAN;
-7170)  0)--
    0)  1)-- PROCEDURE SETA5( IOFFST : INTEGER );FORWARD;
    0)  1)--
    0)  1)-- PROCEDURE PUTINT( IVAR:INTEGER; VAR IPTR:INTEGER );FORWARD;
    0)  1)--
    0)  1)-- PROCEDURE COPROC ;
    0)  1)--
    0)  1)-- CONST
    0)  1)--
    0)  1)--   CMAX=1000000;
    0)  1)--
    0)  1)-- VAR
    0)  1)--
-6144)  1)--   A                  :   AMAT;
-6656)  1)--   F                  :   RVECT;
-6664)  1)--   BP,BR              :   REAL;
-6688)  1)--   N,I,J,IB,IP,I1     :   INTEGER;
-6708)  1)--   I2,I3,II1,K,J1     :   INTEGER;
-6724)  1)--   II,JB,IB1,IBI      :   INTEGER;
-6736)  1)--   SCNT2,ERR,IPTR     :   INTEGER;
-6736)  1)--
    1   1)A- BEGIN
    2   1)--   SETA5( 16#300000 );
    3   1)--   SYNC2:=TRUE;
    4   1)--   IPTR:=0;
    5   1)--   SCNT2:=0;
    6   1)--   ERR:=0;
    7   1)--   II:=2;
    8   1)--   II1:=1;
    9   1)--   I2:=2;
   10   1)--   I3:=3;
   11   1)--   N:=30;
   12   1)--   K:=128;
        1)--
        1)--
   13   1)--   FOR IB:= 1 TO N DO
   14   1)--     FOR IP:= 1 TO 4 DO
        1)B-       BEGIN
        1)--
   15   1)--         SCNT2:=0;
        1)C-         REPEAT                     [ SYNCHRONIZE WITH PROCESSOR 1 ]
   16   1)--           SCNT2:=SCNT2+1;
   17   1)--           IF SCNT2 > CMAX THEN
        1)D-             BEGIN
   18   1)--               SYNC1:=TRUE;
   19   1)--               ERR:= ERR + 1;
        1)-D             END;
   20   1)-C         UNTIL SYNC1;
   21   1)--         SYNC1:= FALSE;
        1)--
   22   1)--         BP:= B[ IP,IP,IB,I2 ];
   23   1)--         J:= IP + 1;
   24   1)--         WHILE J <= 4 DO
        1)C-           BEGIN
   25   1)--             B[ IP,J,IB,I2 ]:= B[ IP,J,IB,I2 ] / BP;
   26   1)--             J:=J+2;
```

```
                 1)--C            END;
        27       1)---          J:= 2;
        28       1)---          WHILE J <= 4 DO
                 1)C--            BEGIN
        29       1)---              BC IP,J,IB,I3 ]:= BC IP,J,IB,I3 ] / BP;
        30       1)---              J:= J + 2;
                 1)--C            END;
                 1)---
                 1)---
        31       1)---          IF IP <> 4 THEN
                 1)C--            BEGIN
        32       1)---              I1:= IP + 1;
        33       1)---              FOR I:= I1 TO 4 DO
                 1)D--                BEGIN
        34       1)---                  BR:= BC I,IP,IB,I2 ];
        35       1)---                  J:= IP + 1;
        36       1)---                  WHILE J <= 4 DO
                 1)E--                    BEGIN
        37       1)---                      BC I,J,IB,I2 ]:= BC I,J,IB,I2 ] - BR * BC IP,J,IB,I2 ];
        38       1)---                      J:= J + 2;
                 1)--E                    END;
        39       1)---                  J:= 2;
        40       1)---                  WHILE J <= 4 DO
                 1)E--                    BEGIN
        41       1)---                      BC I,J,IB,I3 ]:= BC I,J,IB,I3 ] - BR * BC IP,J,IB,I3 ];
        42       1)---                      J:= J + 2;
                 1)--E                    END;
                 1)--D                END; [ FOR I]
                 1)--C            END; [IF IP]
        43       1)---          IF IB <> N THEN
                 1)C--            BEGIN
        44       1)---              FOR I:= 1 TO 4 DO
                 1)D--                BEGIN
        45       1)---                  IB1:= IB + 1;
        46       1)---                  BR:= BC I,IP,IB1,II1];
        47       1)---                  J:= IP + 1;
        48       1)---                  WHILE J <= 4 DO
                 1)E--                    BEGIN
        49       1)---                      BC I,J,IB1,II1 ]:= BC I,J,IB1,II1 ] - BR * BC IP,J,IB,I2 ];
        50       1)---                      J:= J + 2;
                 1)--E                    END;
        51       1)---                  J:= 2;
        52       1)---                  WHILE J <= 4 DO
                 1)E--                    BEGIN
        53       1)---                      BC I,J,IB1,I2 ]:= BC I,J,IB1,I2 ] - BR * BC IP,J,IB,I3 ];
        54       1)---                      J:= J + 2;
                 1)--E                    END;
                 1)--D                END; [FOR I]
                 1)--C            END; [ IF IB]
        55       1)---        SYNC2:=TRUE;
                 1)--B      END; [ FOR IP]
                 1)---
        56       1)---  PUTINT( ERR,IPTR );
        57       1)---  PUTINT( SCNT2,IPTR );
                 1)--A END; [ COPROC ]
                 1)---
**** PUTINT    ASSUMED EXTERNAL
**** SETA5     ASSUMED EXTERNAL
        58       0)A-- BEGIN
        59       0)---    COPROC;
                 0)--A END.

     **** NO ERROR(S) AND NO WARNING(S) DETECTED

     **** 127 LINES 3 PROCEDURES

     **** 717 PCODE INSTRUCTIONS
```

Dual-Processor Block Tridiagonal Solver; Column Sweep Serial Back Substitution, Processor 1

```
 1(        0) 0)--- PROGRAM SOLVE;
 2(        0) 0)---
 3(        0) 0)--- TYPE
```

ORIGINAL PAGE IS OF POOR QUALITY

```
        0)  0)---
        0)  0)--- RVECT=ARRAY [1..4,1..32] OF REAL;
        0)  0)--- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
        0)  0)---
        0)  0)--- VAR
        0)  0)---
    -6144)  0)---   B                  :   AMAT;
    -7168)  0)---   C,DU               :   RVECT;
    -7170)  0)---   SYNC1,SYNC2        :   BOOLEAN;
    -7170)  0)---
        0)  1)--- PROCEDURE GETINF( VAR ADDR:RVECT; NUMEL:INTEGER );FORWARD;
        0)  1)---
        0)  1)--- PROCEDURE PUTINT( IVAL:INTEGER; VAR IPTR:INTEGER );FORWARD;
        0)  1)---
        0)  1)--- PROCEDURE SYNCRO( VAR EFLG,CNT:INTEGER; MAXCNT:INTEGER; VAR SFLG:BOOLEAN );
        0)  1)--- FORWARD;
        0)  1)---
        0)  1)--- PROCEDURE IDATA( VAR MATRIXA : AMAT; VCNT : INTEGER );FORWARD;
        0)  1)---
        0)  1)--- PROCEDURE IDATF( VAR MATRIXF : RVECT; VCNT1 : INTEGER ); FORWARD;
        0)  1)---
        0)  1)--- PROCEDURE COPROC ;
        0)  1)---
        0)  1)--- CONST
        0)  1)---
        0)  1)---    CMAX=1000000;
        0)  1)--- VAR
        0)  1)---
    -6144)  1)---    A                   :   AMAT;
    -6656)  1)---    F                   :   RVECT;
    -6664)  1)---    BP,BR               :   REAL;
    -6688)  1)---    N,I,J,IB,IP,I1      :   INTEGER;
    -6708)  1)---    I2,I3,II1,K,J1      :   INTEGER;
    -6724)  1)---    II,JB,IB1,IBI       :   INTEGER;
    -6736)  1)---    ERR,SCNT1,IPTR      :   INTEGER;
    -6736)  1)---
    -6736)  1)---
        1   1)A-- BEGIN
        2   1)---    SYNC1:=FALSE;
        3   1)---    IPTR:=0;
        4   1)---    SCNT1:=0;
        5   1)---    ERR:=0;
        6   1)---    II1:= 1;
        7   1)---    I2:=2;
        8   1)---    I3:=3;
        9   1)---    N:=30;
            1)---
       10   1)---    IDATA( B,1536 );
       11   1)---    IDATF( C,128 );
            1)---
       12   1)---    SYNC1:=TRUE;
            1)---
       13   1)---    FOR IB:= 1 TO N DO
       14   1)---      FOR IP:= 1 TO 4 DO
            1)B--        BEGIN
            1)---
       15   1)---          SYNCRO( ERR,SCNT1,CMAX,SYNC2 );
            1)--- [        SCNT1:=0;                  ]
            1)--- [        REPEAT         ]           [ SYNCHRONIZE WITH PROCESSOR 2 ]
            1)--- [          SCNT1:=SCNT1+1;      ]
            1)--- [          IF SCNT1 > CMAX THEN ]
            1)--- [            BEGIN               ]
            1)--- [              SYNC2:=TRUE;       ]
            1)--- [              ERR:=ERR + 1;      ]
            1)--- [            END;                  ]
            1)--- [        UNTIL SYNC2;               ]
       16   1)---          SYNC2:= FALSE;
            1)---
       17   1)---          BP:= B[ IP,IP,IB,I2 ];
       18   1)---          J:= IP;
       19   1)---          WHILE J <= 4 DO
            1)C--            BEGIN
       20   1)---              B[ IP,J,IB,I2 ]:= B[ IP,J,IB,I2 ] / BP;
       21   1)---              J:=J+2;
            1)--C            END;
       22   1)---          J:= 1;
       23   1)---          WHILE J <= 4 DO
```

```
        1)C--          BEGIN
    24  1)---            B[ IP,J,IB,I3 ]:= B[ IP,J,IB,I3 ] / BP;
    25  1)---            J:= J + 2;
        1)-C           END;
    26  1)---        C[ IP,IB ]:= C[ IP,IB ] /BP;
        1)---
        1)---
    27  1)---        IF IP <> 4 THEN
        1)C--          BEGIN
    28  1)---            I1:= IP + 1;
    29  1)---            FOR I:= I1 TO 4 DO
        1)D--              BEGIN
    30  1)---                BR:= B[ I,IP,IB,I2 ];
    31  1)---                J:= IP;
    32  1)---                WHILE J <= 4 DO
        1)E--                  BEGIN
    33  1)---                    B[ I,J,IB,I2 ]:= B[ I,J,IB,I2 ] - BR * B[ IP,J,IB,I2 ];
    34  1)---                    J:= J + 2;
        1)-E                   END;
    35  1)---                J:= 1;
    36  1)---                WHILE J <= 4 DO
        1)E--                  BEGIN
    37  1)---                    B[ I,J,IB,I3 ]:= B[ I,J,IB,I3 ] - BR * B[ IP,J,IB,I3 ];
    38  1)---                    J:= J + 2;
        1)-E                   END;
    39  1)---                C[ I,IB ]:= C[ I,IB ] - BR * C[ IP,IB ];
        1)-D               END; [ FOR I]
        1)-C           END; [IF IP]
    40  1)---        IF IB <> N THEN
        1)C--          BEGIN
    41  1)---            FOR I:= 1 TO 4 DO
        1)D--              BEGIN
    42  1)---                IB1:= IB + 1;
    43  1)---                BR:= B[ I,IP,IB1,II1];
    44  1)---                J:= IP;
    45  1)---                WHILE J <= 4 DO
        1)E--                  BEGIN
    46  1)---                    B[ I,J,IB1,II1 ]:= B[ I,J,IB1,II1 ] - BR * B[ IP,J,IB,I2 ];
    47  1)---                    J:= J + 2;
        1)-E                   END;
    48  1)---                J:= 1;
    49  1)---                WHILE J <= 4 DO
        1)E--                  BEGIN
    50  1)---                    B[ I,J,IB1,I2 ]:= B[ I,J,IB1,I2 ] - BR * B[ IP,J,IB,I3 ];
    51  1)---                    J:= J + 2;
        1)-E                   END;
    52  1)---                C[ I,IB1 ]:= C[ I,IB1 ] - BR * C[ IP,IB ];
        1)-D               END; [FOR I]
        1)-C           END; [ IF IB]
    53  1)---      SYNC1:=TRUE;
        1)-B     END; [ FOR IP]
        1)---
    54  1)---    SYNCRO( ERR,SCNT1,CMAX,SYNC2 );
    55  1)---    SYNC2:= FALSE;
        1)---  [ SCNT1:=0;                     ]
        1)--- [  REPEAT          ]    [ SYNCHRONIZE WITH PROCESSOR 2 ]
        1)--- [    SCNT1:=SCNT1+1;  ]
        1)--- [    IF SCNT1 > CMAX THEN ]
        1)--- [      BEGIN              ]
        1)--- [        SYNC2:=TRUE;     ]
        1)--- [        ERR:= ERR + 1;   ]
        1)--- [      END;               ]
        1)--- [  UNTIL SYNC2;           ]
        1)---
        1)--- [COLUMN SWEEP BACKSUBSTITUTION ALGORITHM]
    56  1)---  FOR IB:= 1 TO N DO
        1)B--    BEGIN
    57  1)---      I:=1;
    58  1)---      WHILE I <= 4 DO
        1)C--        BEGIN
    59  1)---          C[I,IB]:= -1.0 * C[I,IB];
    60  1)---          I:=I + 2;
        1)-C         END;
        1)-B     END;
    61  1)---  SYNC1:=TRUE;
    62  1)---  SYNCRO( ERR,SCNT1,CMAX,SYNC2 );
    63  1)---  SYNC2:=FALSE;
```

```
            1)--
     64     1)--    FOR IBI:= 1 TO N DO
            1)B--     BEGIN
     65     1)--        IB:= N + 1 - IBI;
     66     1)--        IB1:= IB - 1;
     67     1)--        DU[4,IB]:= C[4,IB];
     68     1)--        FOR I:= 4 DOWNTO 1 DO
            1)C--         BEGIN
     69     1)--            I1:=I-1;
     70     1)--            IF ( NOT((IB = 1) AND (I = 1)) AND (I <> 1)) THEN
            1)D--             BEGIN
     71     1)--                J:=I1;
     72     1)--                WHILE J >= 1 DO
            1)E--                 BEGIN
     73     1)--                    C[J,IB]:= C[J,IB] - B[J,I,IB,2] * DU[I,IB];
     74     1)--                    J:=J-2;
            1)-E                  END;
            1)-D              END;
     75     1)--            IF IB <> 1 THEN
            1)D--             BEGIN
     76     1)--                J:=1;
     77     1)--                WHILE J <= 4 DO
            1)E--                 BEGIN
     78     1)--                    C[J,IB1]:= C[J,IB1] - B[J,I,IB1,3] * DU[I,IB];
     79     1)--                    J:=J + 2;
            1)-E                  END;
            1)-D              END;
     80     1)--            SYNCRO( ERR,SCNT1,CMAX,SYNC2 );
     81     1)--            SYNC2:=FALSE;
     82     1)--            DU[I1,IB]:=C[I1,IB];
     83     1)--            SYNC1:=TRUE;
            1)-C          END;
            1)-B      END;
            1)--
            1)-- [OLD BACKSUBSTITUTION]
            1)-- [ FOR IBI:= 1 TO N DO
            1)--      BEGIN
            1)--        IB:= N + 1 - IBI;
            1)--        IB1:= IB + 1;
            1)--        FOR II:= 1 TO 4 DO
            1)--          BEGIN
            1)--            I:= 5 - II;
            1)--            DU[ I,IB ]:= -1.0 * C[ I,IB ];
            1)--            IF I <> 4 THEN
            1)--              BEGIN
            1)--                J:= I + 1;
            1)--                WHILE J <= 4 DO
            1)--                  BEGIN
            1)--                    DU[ I,IB ]:= DU[ I,IB ] - B[ I,J,IB,I2 ] * DU[ J,IB ];
            1)--                    J:= J + 1;
            1)--                  END;
            1)--              END;
            1)--            IF IB <> N THEN
            1)--              BEGIN
            1)--                J:= 1;
            1)--                WHILE J <= 4 DO
            1)--                  BEGIN
            1)--                    DU[ I,IB ]:= DU[ I,IB ] - B[ I,J,IB,I3 ] * DU[ J,IB1 ];
            1)--                    J:=J + 1;
            1)--                  END;
            1)--              END;
            1)--          END;
            1)--      END;  ]
     84     1)--    PUTINT( ERR,IPTR );
     85     1)--    PUTINT( SCNT1,IPTR );
     86     1)--    K:=128;
     87     1)--    GETINF( DU,K );
            1)-A END;  [ COPROC ]
            1)--
**** IDATF    ASSUMED EXTERNAL
**** IDATA    ASSUMED EXTERNAL
**** SYNCRO   ASSUMED EXTERNAL
**** PUTINT   ASSUMED EXTERNAL
**** GETINF   ASSUMED EXTERNAL
     88     0)A- BEGIN
     89     0)--    COPROC;
            0)-A END.
```

```
**** NO ERROR(S) AND NO WARNING(S) DETECTED

**** 229 LINES 6 PROCEDURES

**** 1199 PCODE INSTRUCTIONS
```

Dual-Processor Block Tridiagonal Solver; Column Sweep Back Substitution, Processor 2

```
      0) 0)--- PROGRAM SOLVE;
      0) 0)---
      0) 0)--- TYPE
      0) 0)---
      0) 0)--- RVECT=ARRAY [1..4,1..32] OF REAL;
      0) 0)--- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
      0) 0)---
      0) 0)--- VAR
      0) 0)---
  -6144) 0)---    B               :    AMAT;
  -7168) 0)---    C,DU            :    RVECT;
  -7170) 0)---    SYNC1,SYNC2     :    BOOLEAN;
  -7170) 0)---
      0) 1)--- PROCEDURE SETA5( IOFFST : INTEGER );FORWARD;
      0) 1)---
      0) 1)--- PROCEDURE PUTINT( IVAR:INTEGER; VAR IPTR:INTEGER );FORWARD;
      0) 1)---
      0) 1)--- PROCEDURE SYNCRO( VAR EFLG,CNT:INTEGER; MAXCNT:INTEGER; VAR SFLG:BOOLEAN );
      0) 1)--- FORWARD;
      0) 1)---
      0) 1)--- PROCEDURE COPROC ;
      0) 1)---
      0) 1)--- CONST
      0) 1)---
      0) 1)---    CMAX=1000000;
      0) 1)---
      0) 1)--- VAR
      0) 1)---
  -6144) 1)---    A                   :    AMAT;
  -6656) 1)---    F                   :    RVECT;
  -6664) 1)---    BP,BR               :    REAL;
  -6688) 1)---    N,I,J,IB,IP,I1      :    INTEGER;
  -6708) 1)---    I2,I3,II1,K,J1      :    INTEGER;
  -6724) 1)---    II,JB,IB1,IBI       :    INTEGER;
  -6736) 1)---    SCNT2,ERR,IPTR      :    INTEGER;
  -6736) 1)---
      1  1)A-- BEGIN
      2  1)---    SETA5( 16#300000 );
      3  1)---    SYNC2:=TRUE;
      4  1)---    IPTR:=0;
      5  1)---    SCNT2:=0;
      6  1)---    ERR:=0;
      7  1)---    II:=2;
      8  1)---    II1:=1;
      9  1)---    I2:=2;
     10  1)---    I3:=3;
     11  1)---    N:=30;
     12  1)---    K:=128;
         1)---
         1)---
     13  1)---    FOR IB:= 1 TO N DO
     14  1)---       FOR IP:= 1 TO 4 DO
         1)B--          BEGIN
         1)---
     15  1)---             SYNCRO( ERR,SCNT2,CMAX,SYNC1 );
         1)---  [          SCNT2:=0;                    ]
         1)---  [          REPEAT        ]         [ SYNCHRONIZE WITH PROCESSOR 1 ]
         1)---  [            SCNT2:=SCNT2+1;     ]
         1)---  [            IF SCNT2 > CMAX THEN ]
         1)--- [               BEGIN             ]
         1)--- [                 SYNC1:=TRUE;    ]
         1)--- [                 ERR:= ERR + 1;  ]
         1)--- [               END;    ]
         1)--- [           UNTIL SYNC1; ]
     16  1)---             SYNC1:= FALSE;
```

```
        1)--
17      1)--          BP:= B[ IP,IP,IB,I2 ];
18      1)--          J:= IP + 1;
19      1)--          WHILE J <= 4 DO
        1)C-            BEGIN
20      1)--              B[ IP,J,IB,I2 ]:= B[ IP,J,IB,I2 ] / BP;
21      1)--              J:=J+2;
        1)-C            END;
22      1)--          J:= 2;
23      1)--          WHILE J <= 4 DO
        1)C-            BEGIN
24      1)--              B[ IP,J,IB,I3 ]:= B[ IP,J,IB,I3 ] / BP;
25      1)--              J:= J + 2;
        1)-C            END;
        1)--
        1)--
26      1)--          IF IP <> 4 THEN
        1)C-            BEGIN
27      1)--              I1:= IP + 1;
28      1)--              FOR I:= I1 TO 4 DO
        1)D-                BEGIN
29      1)--                  BR:= B[ I,IP,IB,I2 ];
30      1)--                  J:= IP + 1;
31      1)--                  WHILE J <= 4 DO
        1)E-                    BEGIN
32      1)--                      B[ I,J,IB,I2 ]:= B[ I,J,IB,I2 ] - BR * B[ IP,J,IB,I2 ];
33      1)--                      J:= J + 2;
        1)-E                    END;
34      1)--                  J:= 2;
35      1)--                  WHILE J <= 4 DO
        1)E-                    BEGIN
36      1)--                      B[ I,J,IB,I3 ]:= B[ I,J,IB,I3 ] - BR * B[ IP,J,IB,I3 ];
37      1)--                      J:= J + 2;
        1)-E                    END;
        1)-D                END; [ FOR I]
        1)-C            END; [IF IP]
38      1)--          IF IB <> N THEN
        1)C-            BEGIN
39      1)--              FOR I:= 1 TO 4 DO
        1)D-                BEGIN
40      1)--                  IB1:= IB + 1;
41      1)--                  BR:= B[ I,IP,IB1,II1];
42      1)--                  J:= IP + 1;
43      1)--                  WHILE J <= 4 DO
        1)E-                    BEGIN
44      1)--                      B[ I,J,IB1,II1 ]:= B[ I,J,IB1,II1 ] - BR * B[ IP,J,IB,I2 ];
45      1)--                      J:= J + 2;
        1)-E                    END;
46      1)--                  J:= 2;
47      1)--                  WHILE J <= 4 DO
        1)E-                    BEGIN
48      1)--                      B[ I,J,IB1,I2 ]:= B[ I,J,IB1,I2 ] - BR * B[ IP,J,IB,I3 ];
49      1)--                      J:= J + 2;
        1)-E                    END;
        1)-D                END; [FOR I]
        1)-C            END; [ IF IB]
50      1)--        SYNC2:=TRUE;
        1)-B      END; [ FOR IP]
        1)--
51      1)--    SYNCRO( ERR,SCNT2,CMAX,SYNC1 );
52      1)--    SYNC1:=FALSE;
        1)--
        1)-- [COLUMN SWEEP BACKSUBSTITUTION ALGORITHM]
53      1)--    FOR IB:= 1 TO N DO
        1)B-      BEGIN
54      1)--        I:=2;
55      1)--        WHILE I <= 4 DO
        1)C-          BEGIN
56      1)--            C[I,IB]:= -1.0 * C[I,IB];
57      1)--            I:=I + 2;
        1)-C          END;
        1)-B      END;
58      1)--    SYNC2:=TRUE;
59      1)--    SYNCRO( ERR,SCNT2,CMAX,SYNC1 );
60      1)--    SYNC1:=FALSE;
        1)--
61      1)--    FOR IBI:= 1 TO N DO
```

```
          1)B-      BEGIN
     62   1)---       IB:= N + 1 - IBI;
     63   1)---       IB1:= IB - 1;
     64   1)---       DU[4,IB]:= C[4,IB];
     65   1)---       FOR I:= 4 DOWNTO 1 DO
          1)C-          BEGIN
     66   1)---           I1:=I-1;
     67   1)---           IF ( NOT((IB = 1) AND (I = 1)) AND (I <> 1)) THEN
          1)D-              BEGIN
     68   1)---               J:=I1 - 1;
     69   1)---               WHILE J >= 1 DO
          1)E-                  BEGIN
     70   1)---                   C[J,IB]:= C[J,IB] - B[J,I,IB,2] * DU[I,IB];
     71   1)---                   J:=J-2;
          1)-E                  END;
          1)-D              END;
     72   1)---           IF IB <> 1 THEN
          1)D-              BEGIN
     73   1)---               J:=2;
     74   1)---               WHILE J <= 4 DO
          1)E-                  BEGIN
     75   1)---                   C[J,IB1]:= C[J,IB1] - B[J,I,IB1,3] * DU[I,IB];
     76   1)---                   J:=J + 2;
          1)-E                  END;
          1)-D              END;
     77   1)---           SYNC2:=TRUE;
     78   1)---           SYNCRO( ERR,SCNT2,CMAX,SYNC1 );
     79   1)---           SYNC1:=FALSE;
          1)-C          END;
          1)-B      END;
          1)---
     80   1)---  PUTINT( ERR,IPTR );
     81   1)---  PUTINT( SCNT2,IPTR );
          1)-A END; [ COPROC ]
          1)---
**** SYNCRO   ASSUMED EXTERNAL
**** PUTINT   ASSUMED EXTERNAL
**** SETA5    ASSUMED EXTERNAL
     82   0)A- BEGIN
     83   0)---   COPROC;
          0)-A END.

     **** NO ERROR(S) AND NO WARNING(S) DETECTED

     **** 180 LINES 4 PROCEDURES

     **** 1040 PCODE INSTRUCTIONS
```

Three-Processor Block Tridiagonal Solver; Processor 1

```
       0) 0)--- PROGRAM SOLVE;
       0) 0)---
       0) 0)--- TYPE
       0) 0)---
       0) 0)--- INT5=ARRAY [1..5] OF INTEGER;
       0) 0)--- REAL4=ARRAY [1..4] OF REAL;
       0) 0)--- RVECT=ARRAY [1..4,1..32] OF REAL;
       0) 0)--- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
       0) 0)---
       0) 0)--- VAR
       0) 0)---
   -6144) 0)---   B                    :    AMAT;
   -7168) 0)---   C,DU                 :    RVECT;
   -7168) 0)---
   -7168) 0)---
       0) 1)--- PROCEDURE GETINF( VAR ADDR:RVECT; NUMEL:INTEGER );FORWARD;
       0) 1)---
       0) 1)--- PROCEDURE PUTINT( IVAL:INTEGER; VAR IPTR:INTEGER );FORWARD;
       0) 1)---
       0) 1)--- PROCEDURE SYNCRO2( VAR SYNCINF : INT5 );FORWARD;
       0) 1)---
       0) 1)--- PROCEDURE IDATA( VAR MATRIXA : AMAT; VCNT : INTEGER );FORWARD;
```

```
         0) 1)----
         0) 1)---- PROCEDURE IDATF( VAR MATRIXF : RVECT; VCNT1 : INTEGER ); FORWARD;
         0) 1)----
         0) 1)---- PROCEDURE COPROC ;
         0) 1)----
         0) 1)---- CONST
         0) 1)----
         0) 1)----   CMAX=1000000;
         0) 1)---- VAR
         0) 1)----
       -64) 1)----   AI2,AII1,BI2,BI3 :    REAL4;
      -576) 1)----   F                :    RVECT;
      -584) 1)----   BP,BR            :    REAL;
      -608) 1)----   N,I,J,IB,IP,I1   :    INTEGER;
      -628) 1)----   I2,I3,II1,K,J1   :    INTEGER;
      -644) 1)----   II,JB,IB1,IBI    :    INTEGER;
      -656) 1)----   ERR,SCNT1,IPTR   :    INTEGER;
      -676) 1)----   SYNCTAB          :    INT5;
      -676) 1)----
         1  1)A-- BEGIN
         2  1)----   IPTR:=0;
         3  1)----   SYNCTAB[3]:=CMAX;
         4  1)----   SYNCTAB[4]:=0;
         5  1)----   SYNCTAB[1]:=1;
         6  1)----   SYNCTAB[5]:=3;
         7  1)----   II1:= 1;
         8  1)----   I2:=2;
         9  1)----   I3:=3;
        10  1)----   N:=30;
            1)----
        11  1)----   IDATA( B,1536 );
        12  1)----   IDATF( C,128 );
            1)----
            1)----
        13  1)----   FOR IB:= 1 TO N DO
        14  1)----     FOR IP:= 1 TO 4 DO
            1)B--        BEGIN
            1)----
        15  1)----         SYNCRO2( SYNCTAB );
        16  1)----         BP:= B[ IP,IP,IB,I2 ];
        17  1)----         I1:= IP + 1;
        18  1)----         IB1:=IB + 1;
            1)----
            1)----
        19  1)----         J:=IP;
        20  1)----         WHILE J <= 4 DO
            1)C--            BEGIN
        21  1)----             BI2[J]:=B[ IP,J,IB,I2 ];
        22  1)----             J:= J + 3;
            1)--C            END;
            1)----
        23  1)----         J:=1;
        24  1)----         IF ( (IP = 1) OR (IP = 4 ) )
        25  1)----           THEN K:= 4
        26  1)----           ELSE K:= 3;
            1)----
        27  1)----         WHILE J <= 4 DO
            1)C--            BEGIN
        28  1)----             BI3[ J ]:= B[ IP,J,IB,I3 ];
        29  1)----             J:= J + K;
            1)--C            END;
            1)----
        30  1)----         J:= IP;
        31  1)----         WHILE J <= 4 DO
            1)C--            BEGIN
        32  1)----             BI2[ J ]:= BI2[ J ] / BP;
        33  1)----             B[ IP,J,IB,I2 ]:= BI2[ J ];
        34  1)----             J:=J+3;
            1)--C            END;
            1)----
        35  1)----         J:= 1;
        36  1)----         WHILE J <= 4 DO
            1)C--            BEGIN
        37  1)----             BI3[ J ]:= BI3[ J ] / BP;
        38  1)----             B[ IP,J,IB,I3 ]:= BI3[ J ];
        39  1)----             J:= J + K;
            1)--C            END;
```

```
        1)----
   40   1)----          IF IP <> 4 THEN
        1)C--             BEGIN
   41   1)----              FOR I:= I1 TO 4 DO
        1)D--                 BEGIN
   42   1)----                  BR:= B[ I,IP,IB,I2 ];
        1)----
   43   1)----                  J:= IP + 3;
   44   1)----                  WHILE J <= 4 DO
        1)E--                     BEGIN
   45   1)----                      B[ I,J,IB,I2 ]:= B[ I,J,IB,I2 ] - BR * BI2[ J ];
   46   1)----                      J:= J + 3;
        1)--E                     END;
        1)----
   47   1)----                  J:= 1;
   48   1)----                  WHILE J <= 4 DO
        1)E--                     BEGIN
   49   1)----                      B[ I,J,IB,I3 ]:= B[ I,J,IB,I3 ] - BR * BI3[ J ];
   50   1)----                      J:= J + K;
        1)--E                     END;
        1)----
        1)--D                 END; [ FOR I]
        1)--C             END; [IF IP]
        1)----
   51   1)----          IF IB <> N THEN
        1)C--             BEGIN
   52   1)----              FOR I:= 1 TO 4 DO
        1)D--                 BEGIN
   53   1)----                  BR:= B[ I,IP,IB1,II1 ];
        1)----
   54   1)----                  J:= IP + 3;
   55   1)----                  WHILE J <= 4 DO
        1)E--                     BEGIN
   56   1)----                      B[ I,J,IB1,II1 ]:= B[ I,J,IB1,II1 ] - BR * BI2[ J ];
   57   1)----                      J:= J + 3;
        1)--E                     END;
        1)----
   58   1)----                  J:= 1;
   59   1)----                  WHILE J <= 4 DO
        1)E--                     BEGIN
   60   1)----                      B[ I,J,IB1,I2 ]:= B[ I,J,IB1,I2 ] - BR * BI3[ J ];
   61   1)----                      J:= J + K;
        1)--E                     END;
        1)----
   62   1)----                  IF ( (IP = 1) OR (IP = 4 ) ) THEN
   63   1)----                  C[ I,IB1 ]:= C[ I,IB1 ] - BR * C[ IP,IB ];
        1)----
        1)--D                 END; [FOR I]
        1)--C             END; [ IF IB]
        1)--B       END; [ FOR IP]
        1)----
   64   1)----    SYNCRO2( SYNCTAB );
        1)----
        1)---- [COLUMN SWEEP BACKSUBSTITUTION ALGORITHM]
   65   1)----    FOR IB:= 1 TO N DO
        1)B--       BEGIN
   66   1)----        I:=1;
   67   1)----        WHILE I <= 4 DO
        1)C--           BEGIN
   68   1)----            C[I,IB]:= -1.0 * C[I,IB];
   69   1)----            I:=I + 3;
        1)--C           END;
        1)--B       END;
        1)----
   70   1)----    SYNCRO2( SYNCTAB );
        1)----
   71   1)----    FOR IBI:= 1 TO N DO
        1)B--       BEGIN
   72   1)----        IB:= N + 1 - IBI;
   73   1)----        IB1:= IB - 1;
   74   1)----        DU[4,IB]:= C[4,IB];
   75   1)----        FOR I:= 4 DOWNTO 1 DO
        1)C--           BEGIN
   76   1)----            I1:=I-1;
   77   1)----            IF ( NOT((IB = 1) AND (I = 1)) AND (I <> 1)) THEN
        1)D--               BEGIN
   78   1)----                J:=I1;
```


ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

```
   79  1)---             WHILE J >= 1 DO
       1)E-                BEGIN
   80  1)---                 C[J,IB]:= C[J,IB] - B[J,I,IB,2] * DU[I,IB];
   81  1)---                 J:=J-3;
       1)-E                END;
       1)-D            END;
   82  1)---         IF IB <> 1 THEN
       1)D-            BEGIN
   83  1)---             J:=2;
   84  1)---             WHILE J <= 4 DO
       1)E-                BEGIN
   85  1)---                 C[J,IB1]:= C[J,IB1] - B[J,I,IB1,3] * DU[I,IB];
   86  1)---                 J:=J + 3;
       1)-E                END;
       1)-D            END;
   87  1)---         SYNCRO2( SYNCTAB );
   88  1)---         DU[1,IB]:=C[1,IB];
   89  1)---         SYNCRO2( SYNCTAB );
       1)-C        END;
       1)-B    END;
       1)---
   90  1)---   ERR:=SYNCTAB[4];
   91  1)---   PUTINT( ERR,IPTR );
   92  1)---   SCNT1:=SYNCTAB[2];
   93  1)---   PUTINT( SCNT1,IPTR );
   94  1)---   K:=128;
   95  1)---   GETINF( DU,K );
       1)-A END;  [ COPROC ]
       1)---
**** IDATF     ASSUMED EXTERNAL
**** IDATA     ASSUMED EXTERNAL
**** SYNCRO2   ASSUMED EXTERNAL
**** PUTINT    ASSUMED EXTERNAL
**** GETINF    ASSUMED EXTERNAL
   96  0)A- BEGIN
   97  0)---   COPROC;
       0)-A END.

     **** NO ERROR(S) AND NO WARNING(S) DETECTED

     **** 208 LINES 6 PROCEDURES

     **** 1150 PCODE INSTRUCTIONS
```

Three-Processor Block Tridiagonal Solver; Processor 2

```
(      0) 0)---
(      0) 0)--- PROGRAM SOLVE;
(      0) 0)---
(      0) 0)--- TYPE
(      0) 0)---
(      0) 0)--- INT5=ARRAY [1..5] OF INTEGER;
(      0) 0)--- REAL4=ARRAY [1..4] OF REAL;
(      0) 0)--- RVECT=ARRAY [1..4,1..32] OF REAL;
(      0) 0)--- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
(      0) 0)---
(      0) 0)--- VAR
(      0) 0)---
(  -6144) 0)---   B                          :   AMAT;
(  -7168) 0)---   C,DU                       :   RVECT;
(  -7168) 0)---
(  -7168) 0)---
(      0) 1)--- PROCEDURE SETA5( IOFFST : INTEGER );FORWARD;
(      0) 1)---
(      0) 1)--- PROCEDURE PUTINT( IVAL:INTEGER; VAR IPTR:INTEGER );FORWARD;
(      0) 1)---
(      0) 1)--- PROCEDURE SYNCRO2C( VAR SYNCINF : INT5 );FORWARD;
(      0) 1)---
(      0) 1)--- PROCEDURE COPROC ;
(      0) 1)---
(      0) 1)--- CONST
(      0) 1)---
```

```
   0) 1)---   CMAX=1000000;
   0) 1)--- VAR
   0) 1)---
 -64) 1)---   AI2,AII1,BI2,BI3 :   REAL4;
-576) 1)---   F                :   RVECT;
-584) 1)---   BP,BR            :   REAL;
-608) 1)---   N,I,J,IB,IP,I1   :   INTEGER;
-628) 1)---   I2,I3,II1,K,J1   :   INTEGER;
-644) 1)---   II,JB,IB1,IBI    :   INTEGER;
-656) 1)---   ERR,SCNT1,IPTR   :   INTEGER;
-676) 1)---   SYNCTAB          :   INT5;
-676) 1)---
-676) 1)---
   1  1)A- BEGIN
   2  1)---   SETA5( 16#300000 );
   3  1)---   IPTR:=0;
   4  1)---   SYNCTAB[3]:=CMAX;
   5  1)---   SYNCTAB[4]:=0;
   6  1)---   SYNCTAB[1]:=2;
   7  1)---   SYNCTAB[5]:=3;
   8  1)---   II1:= 1;
   9  1)---   I2:=2;
  10  1)---   I3:=3;
  11  1)---   N:=30;
      1)---
      1)---
  12  1)---   FOR IB:= 1 TO N DO
  13  1)---     FOR IP:= 1 TO 4 DO
      1)B-        BEGIN
  14  1)---         SYNCRO2C( SYNCTAB );
  15  1)---         I1:= IP + 1;
  16  1)---         IB1:=IB + 1;
  17  1)---         BP:= B[ IP,IP,IB,I2 ];
      1)---
      1)--- [       FOR I:= I1 TO 4 DO
      1)---           AI2[I]:=B[I,IP,IB,I2];
      1)---         FOR I:= 1 TO 4 DO
      1)---           AII1[I]:=B[I,IP,IB1,II1];      ]
      1)---
  18  1)---         J:=IP + 1;
  19  1)---         WHILE J <= 4 DO
      1)C-            BEGIN
  20  1)---             BI2[J]:=B[ IP,J,IB,I2 ];
  21  1)---             J:= J + 3;
      1)-C            END;
      1)---
  22  1)---         J:=2;
  23  1)---         IF ( (IP = 1) OR (IP = 4) )
  24  1)---           THEN K:= 2
  25  1)---           ELSE K:= 3;
  26  1)---         WHILE J <= 4 DO
      1)C-            BEGIN
  27  1)---             BI3[ J ]:= B[ IP,J,IB,I3 ];
  28  1)---             J:= J + K;
      1)-C            END;
      1)---
  29  1)---         J:= IP + 1;
  30  1)---         WHILE J <= 4 DO
      1)C-            BEGIN
  31  1)---             BI2[ J ]:= BI2[ J ] / BP;
  32  1)---             B[ IP,J,IB,I2]:= BI2[ J ];
  33  1)---             J:=J+3;
      1)-C            END;
      1)---
  34  1)---         J:= 2;
  35  1)---         WHILE J <= 4 DO
      1)C-            BEGIN
  36  1)---             BI3[ J ]:= BI3[ J ] / BP;
  37  1)---             B[ IP,J,IB,I3 ]:= BI3[ J ];
  38  1)---             J:= J + K;
      1)-C            END;
      1)---
  39  1)---         IF IP <> 4 THEN
      1)C-            BEGIN
  40  1)---             FOR I:= I1 TO 4 DO
      1)D-                BEGIN
  41  1)---                 BR:= B[ I,IP,IB,I2 ];
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        1)---
  42    1)---             J:= IP + 1;
  43    1)---             WHILE J <= 4 DO
        1)E--               BEGIN
  44    1)---                 B[ I,J,IB,I2 ]:= B[ I,J,IB,I2 ] - BR * BI2[ J ];
  45    1)---                 J:= J + 3;
        1)--E               END;
        1)---
  46    1)---             J:= 2;
  47    1)---             WHILE J <= 4 DO
        1)E--               BEGIN
  48    1)---                 B[ I,J,IB,I3 ]:= B[ I,J,IB,I3 ] - BR * BI3[ J ];
  49    1)---                 J:= J + K;
        1)--E               END;
        1)---
        1)--D           END; [ FOR I]
        1)--C       END; [IF IP]
  50    1)---     IF IB <> N THEN
        1)C--       BEGIN
  51    1)---         FOR I:= 1 TO 4 DO
        1)D--           BEGIN
  52    1)---             BR:= B[ I,IP,IB1,II1 ];
        1)---
  53    1)---             J:= IP + 1;
  54    1)---             WHILE J <= 4 DO
        1)E--               BEGIN
  55    1)---                 B[ I,J,IB1,II1 ]:= B[ I,J,IB1,II1 ] - BR * BI2[ J ];
  56    1)---                 J:= J + 3;
        1)--E               END;
        1)---
  57    1)---             J:= 2;
  58    1)---             WHILE J <= 4 DO
        1)E--               BEGIN
  59    1)---                 B[ I,J,IB1,I2 ]:= B[ I,J,IB1,I2 ] - BR * BI3[ J ];
  60    1)---                 J:= J + K;
        1)--E               END;
        1)---
  61    1)---             IF IP = 2 THEN
  62    1)---             C[ I,IB1 ]:= C[ I,IB1 ] - BR * C[ IP,IB ];
        1)---
        1)--D           END; [FOR I]
        1)--C       END; [ IF IB]
        1)--B   END; [ FOR IP]
        1)---
  63    1)---   SYNCRO2C( SYNCTAB );
        1)---
        1)--- [COLUMN SWEEP BACKSUBSTITUTION ALGORITHM]
  64    1)---   FOR IB:= 1 TO N DO
        1)B--     BEGIN
  65    1)---       I:=2;
  66    1)---       WHILE I <= 4 DO
        1)C--         BEGIN
  67    1)---           C[I,IB]:= -1.0 * C[I,IB];
  68    1)---           I:=I + 3;
        1)--C         END;
        1)--B     END;
        1)---
  69    1)---   SYNCRO2C( SYNCTAB );
        1)---
  70    1)---   FOR IBI:= 1 TO N DO
        1)B--     BEGIN
  71    1)---       IB:= N + 1 - IBI;
  72    1)---       IB1:= IB - 1;
  73    1)---       DU[4,IB]:= C[4,IB];
  74    1)---       FOR I:= 4 DOWNTO 1 DO
        1)C--         BEGIN
  75    1)---           I1:=I-1;
  76    1)---           IF ( NOT((IB = 1) AND (I = 1)) AND (I <> 1)) THEN
        1)D--             BEGIN
  77    1)---               J:=I1 - 1;
  78    1)---               WHILE J >= 1 DO
        1)E--                 BEGIN
  79    1)---                   C[J,IB]:= C[J,IB] - B[J,I,IB,2] * DU[I,IB];
  80    1)---                   J:=J-3;
        1)--E                 END;
        1)--D             END;
  81    1)---           IF IB <> 1 THEN
```

```
                1)D--              BEGIN
         82     1)---                J:=3;
         83     1)---                WHILE J <= 4 DO
                1)E--                  BEGIN
         84     1)---                    C[J,IB1]:= C[J,IB1] - B[J,I,IB1,3] * DU[I,IB];
         85     1)---                    J:=J + 3;
                1)-E                   END;
                1)-D               END;
         86     1)---            SYNCRO2C( SYNCTAB );
         87     1)---            SYNCRO2C( SYNCTAB );
                1)-C           END;
                1)-B       END;
                1)---
                1)---
         88     1)---    ERR:=SYNCTAB[4];
         89     1)---    PUTINT( ERR,IPTR );
         90     1)---    SCNT1:=SYNCTAB[2];
         91     1)---    PUTINT( SCNT1,IPTR );
                1)-A END; [ COPROC ]
                1)---
**** SYNCRO2C ASSUMED EXTERNAL
**** PUTINT   ASSUMED EXTERNAL
**** SETA5    ASSUMED EXTERNAL
         92     0)A- BEGIN
         93     0)---   COPROC;
                0)-A END.

      **** NO ERROR(S) AND NO WARNING(S) DETECTED

      **** 203 LINES 4 PROCEDURES

      **** 1107 PCODE INSTRUCTIONS
```

Three-Processor Block Tridiagonal Solver; Processor 3

```
        0) 0)--- PROGRAM SOLVE;
        0) 0)---
        0) 0)--- TYPE
        0) 0)---
        0) 0)--- INT5=ARRAY [1..5] OF INTEGER;
        0) 0)--- REAL4=ARRAY [1..4] OF REAL;
        0) 0)--- RVECT=ARRAY [1..4,1..32] OF REAL;
        0) 0)--- AMAT=ARRAY [1..4,1..4,1..32,1..3] OF REAL;
        0) 0)---
        0) 0)--- VAR
        0) 0)---
    -6144) 0)---   B                      :   AMAT;
    -7168) 0)---   C,DU                   :   RVECT;
    -7168) 0)---
    -7168) 0)---
        0) 1)--- PROCEDURE SETA5( IOFFST : INTEGER );FORWARD;
        0) 1)---
        0) 1)--- PROCEDURE PUTINT( IVAL:INTEGER; VAR IPTR:INTEGER );FORWARD;
        0) 1)---
        0) 1)--- PROCEDURE SYNCRO2C( VAR SYNCINF : INT5 );FORWARD;
        0) 1)---
        0) 1)--- PROCEDURE COPROC ;
        0) 1)---
        0) 1)--- CONST
        0) 1)---
        0) 1)---   CMAX=1000000;
        0) 1)--- VAR
        0) 1)---
      -64) 1)---   AI2,AII1,BI2,BI3  :  REAL4;
     -576) 1)---   F                 :  RVECT;
     -584) 1)---   BP,BR             :  REAL;
     -608) 1)---   N,I,J,IB,IP,I1    :  INTEGER;
     -628) 1)---   I2,I3,II1,K,J1    :  INTEGER;
     -644) 1)---   II,JB,IB1,IBI     :  INTEGER;
     -656) 1)---   ERR,SCNT1,IPTR    :  INTEGER;
     -676) 1)---   SYNCTAB           :  INT5;
     -676) 1)---
```

ORIGINAL PAGE IS OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

```
   1   1)A-- BEGIN
   2   1)---    SETAS( 16#300000 );
   3   1)---    IPTR:=0;
   4   1)---    SYNCTAB[3]:=CMAX;
   5   1)---    SYNCTAB[4]:=0;
   6   1)---    SYNCTAB[1]:=3;
   7   1)---    SYNCTAB[5]:=3;
   8   1)---    II1:= 1;
   9   1)---    I2:=2;
  10   1)---    I3:=3;
  11   1)---    N:=30;
       1)---
  12   1)---    FOR IB:= 1 TO N DO
  13   1)---      FOR IP:= 1 TO 4 DO
       1)B--        BEGIN
  14   1)---          SYNCRO2C( SYNCTAB );
  15   1)---          I1:= IP + 1;
  16   1)---          IB1:=IB + 1;
  17   1)---          BP:= B[ IP,IP,IB,I2 ];
       1)---
       1)--- [        FOR I:= I1 TO 4 DO
       1)---            AI2[I]:=B[I,IP,IB,I2];
       1)---          FOR I:=1 TO 4 DO
       1)---            AII1[I]:=B[I,IP,IB1,II1];      ]
       1)---
  18   1)---          J:=IP + 2;
  19   1)---          WHILE J <= 4 DO
       1)C--            BEGIN
  20   1)---              BI2[J]:=B[ IP,J,IB,I2 ];
  21   1)---              J:= J + 3;
       1)--C            END;
       1)---
  22   1)---          J:=3;
  23   1)---          WHILE J <= 4 DO
       1)C--            BEGIN
  24   1)---              BI3[ J ]:= B[ IP,J,IB,I3 ];
  25   1)---              J:= J + 3;
       1)--C            END;
       1)---
  26   1)---          J:= IP + 2;
  27   1)---          WHILE J <= 4 DO
       1)C--            BEGIN
  28   1)---              BI2[ J ]:= BI2[ J ] / BP;
  29   1)---              B[ IP,J,IB,I2]:= BI2[ J ];
  30   1)---              J:=J+3;
       1)--C            END;
       1)---
  31   1)---          J:= 3 ;
  32   1)---          WHILE J <= 4 DO
       1)C--            BEGIN
  33   1)---              BI3[ J ]:= BI3[ J ] / BP;
  34   1)---              B[ IP,J,IB,I3 ]:= BI3[ J ];
  35   1)---              J:= J + 3;
       1)--C            END;
  36   1)---          C[ IP,IB ]:= C[ IP,IB ] / BP;
       1)---
  37   1)---          IF IP <> 4 THEN
       1)C--            BEGIN
  38   1)---              FOR I:= I1 TO 4 DO
       1)D--                BEGIN
  39   1)---                  BR:= B[ I,IP,IB,I2 ];
       1)---
  40   1)---                  J:= IP + 2;
  41   1)---                  WHILE J <= 4 DO
       1)E--                    BEGIN
  42   1)---                      B[ I,J,IB,I2 ]:= B[ I,J,IB,I2 ] - BR * BI2[ J ];
  43   1)---                      J:= J + 3;
       1)--E                    END;
       1)---
  44   1)---                  J:= 3;
  45   1)---                  WHILE J <= 4 DO
       1)E--                    BEGIN
  46   1)---                      B[ I,J,IB,I3 ]:= B[ I,J,IB,I3 ] - BR * BI3[ J ];
  47   1)---                      J:= J + 3;
       1)--E                    END;
       1)---
  48   1)---                  C[ I,IB ]:= C[ I,IB ] - BR * C[ IP, IB ];
```

```
        1)---
        1)-D                END; [ FOR I]
        1)-C            END; [IF IP]
49      1)---         IF IB <> N THEN
        1)C--           BEGIN
50      1)---             FOR I:= 1 TO 4 DO
        1)D--               BEGIN
51      1)---                 BR:= B[ I,IP,IB1,II1 ];
        1)---
52      1)---                 J:= IP + 2;
53      1)---                 WHILE J <= 4 DO
        1)E--                   BEGIN
54      1)---                     B[ I,J,IB1,II1 ]:= B[ I,J,IB1,II1 ] - BR * BI2[ J ];
55      1)---                     J:= J + 3;
        1)-E                    END;
        1)---
56      1)---                 J:= 3;
57      1)---                 WHILE J <= 4 DO
        1)E--                   BEGIN
58      1)---                     B[ I,J,IB1,I2 ]:= B[ I,J,IB1,I2 ] - BR * BI3[ J ];
59      1)---                     J:= J + 3;
        1)-E                    END;
        1)---
60      1)---                 IF IP = 3 THEN
61      1)---                 C[ I,IB1 ]:= C[ I,IB1 ] - BR * C[ IP,IB ];
        1)---
        1)-D                END; [FOR I]
        1)-C            END; [ IF IB]
        1)-B      END; [ FOR IP]
        1)---
62      1)---     SYNCRO2C( SYNCTAB );
        1)---
        1)---
        1)--- [COLUMN SWEEP BACKSUBSTITUTION ALGORITHM]
63      1)---   FOR IB:= 1 TO N DO
        1)B--     BEGIN
64      1)---       I:=3;
65      1)---       WHILE I <= 4 DO
        1)C--         BEGIN
66      1)---           C[I,IB]:= -1.0 * C[I,IB];
67      1)---           I:=I + 3;
        1)-C          END;
        1)-B      END;
        1)---
68      1)---   SYNCRO2C( SYNCTAB );
        1)---
69      1)---   FOR IBI:= 1 TO N DO
        1)B--     BEGIN
70      1)---       IB:= N + 1 - IBI;
71      1)---       IB1:= IB - 1;
72      1)---       DU[4,IB]:= C[4,IB];
73      1)---       FOR I:= 4 DOWNTO 1 DO
        1)C--         BEGIN
74      1)---           I1:=I-1;
75      1)---           IF ( NOT((IB = 1) AND (I = 1)) AND (I <> 1)) THEN
        1)D--             BEGIN
76      1)---               J:=I1 - 2;
77      1)---               WHILE J >= 1 DO
        1)E--                 BEGIN
78      1)---                   C[J,IB]:= C[J,IB] - B[J,I,IB,2] * DU[I,IB];
79      1)---                   J:=J-3;
        1)-E                  END;
        1)-D              END;
80      1)---           IF IB <> 1 THEN
        1)D--             BEGIN
81      1)---               J:=1;
82      1)---               WHILE J <= 4 DO
        1)E--                 BEGIN
83      1)---                   C[J,IB1]:= C[J,IB1] - B[J,I,IB1,3] * DU[I,IB];
84      1)---                   J:=J + 3;
        1)-E                  END;
        1)-D              END;
85      1)---           SYNCRO2C( SYNCTAB );
86      1)---           SYNCRO2C( SYNCTAB );
        1)-C          END;
        1)-B      END;
        1)---
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
          87  1)---     ERR:=SYNCTAB[4];
          88  1)---     PUTINT( ERR,IPTR );
          89  1)---     SCNT1:=SYNCTAB[2];
          90  1)---     PUTINT( SCNT1,IPTR );
              1)-A END;  [ COPROC ]
              1)---
**** SYNCRO2C ASSUMED EXTERNAL
**** PUTINT   ASSUMED EXTERNAL
**** SETAS    ASSUMED EXTERNAL
          91  0)A-- BEGIN
          92  0)---     COPROC;
              0)-A END.

     **** NO ERROR(S) AND NO WARNING(S) DETECTED

     **** 200 LINES 4 PROCEDURES

     **** 1134 PCODE INSTRUCTIONS
```

References

1. Kascak, A.F.: Direct Integration of Transient Rotor Dynamics. NASA TP–1597, 1980.
2. Blech, R.A.; and Arpasi, D.J.: Hardware for a Real-Time Multiprocessor Simulator. NASA TM–83805, 1985.
3. Arpasi, D.J.: Real-Time Multiprocessor Programming Language (RTMPL) User's Manual. NASA TP–2422, 1985.
4. Cole, G.L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator, User's Manual. NASA TP–2426, 1985.
5. Arpasi, D.J.; and Milner, E.J.: Partitioning and Packing Mathematical Simulation Models for Calculation on Parallel Computers. NASA TM–87170, 1986.
6. Ortega, J.M.; and Voigt, R.G.: Solution of Partial Differential Equations on Vector and Parallel Computers. SIAM Rev., vol. 27, no. 2, June 1985, pp. 149–240.
7. Lord, R.E.; Kowalik, J.S.; and Kumar, S.P.: Solving Linear Algebraic Equations on a MIMD Computer. Proceedings of the 1980 International Conference on Parallel Processing, IEEE, 1980, pp. 205–210.
8. Hockney, R.W.; and Jesshope, C.R.: Parallel Computers. Adam Hilger Ltd., Bristol, 1981.
9. Resident PASCAL User's Manual. Motorola Inc., M68KPASC (D4), 1982.
10. Padua, D.A.; and Wolfe, M.J.: Advanced Compiler Optimizations for Supercomputers. Commun. ACM, vol. 29, no. 12, Dec. 1986, pp. 1184–1201.
11. Blech, R.A.: The Hypercluster: A Parallel Processing Test-Bed for Computational Mechanics Applications. NASA TM–89823, 1987.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TP-2892		

4. Title and Subtitle	5. Report Date
Parallel Gaussian Elimination of a Block Tridiagonal Matrix Using Multiple Microcomputers	February 1989
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Richard A. Blech	E-4199
	10. Work Unit No.
	505-62-21
9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Paper
National Aeronautics and Space Administration Washington, D.C. 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The solution of a block tridiagonal matrix using parallel processing is demonstrated in this report. The multiprocessor system which obtained the results and the software environment used to program that system are described. Theoretical partitioning and resource allocation for the Gaussian elimination method used to solve the matrix are discussed. The results obtained from running one-, two-, and three-processor versions of the block tridiagonal solver are presented. The PASCAL source code for these solvers is given in the appendix, and it may be transportable to other shared-memory parallel processors, provided that the synchronization routines are reproduced on the target system.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Parallel processing Numerical methods Microcomputer	Unclassified – Unlimited Subject Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No of pages	22. Price*
Unclassified	Unclassified	36	A03

NASA FORM 1626 OCT 86

*For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1989