AIAA JOURNAL
Vol. 33, No. 11, November 1995

# Hybrid Parallel Computation of Transcendental Structural Eigenvalues

David Kennedy,* W. John Watkins,† and Fred W. Williams‡
*University of Wales, Cardiff CF2 1XH, Wales, United Kingdom*

**Existing coarse grain and fine grain methods are refined to enable the efficient calculation of natural frequencies and critical buckling loads of structures on parallel computers, using the Wittrick–Williams algorithm to solve the transcendental eigenvalue problems arising from the exact solution of the member stiffness equations. The methods are combined to form a novel hybrid method employing both kinds of parallelism simultaneously. Solution times of the hybrid method are up to 24% faster than the best of the existing methods and can be predicted from those of its coarse grain and fine grain components.**

## Introduction

CRITICAL buckling and undamped vibration problems of structural analysis may be solved by the application of exact, analytical solutions of the member stiffness equations. The Wittrick–Williams algorithm[1] provides a solution to the resulting transcendental eigenvalue problems that guarantees convergence on all required critical buckling loads or natural frequencies. Such analysis of three-dimensional frames[2] and prismatic plate assemblies[3] is often much more computationally efficient than analysis by the conventional finite element method.

The exact method locates the eigenvalues using an iterative scheme in which the real symmetric (or complex Hermitian) dynamic stiffness matrix $\underline{K}$ of the structure, whose elements are transcendental functions of the eigenparameter (i.e., the load factor or frequency), is assembled and reduced to upper triangular form $\underline{K}^{\Delta}$ at successive trial values of the eigenparameter. The solution time is dominated by the triangulation of $\underline{K}$ at each iteration but may be reduced either by performing the triangulation more efficiently[4] or by using procedures[5,6] that reduce the number of iterations required to converge on the eigenvalues.

Parallel processing is increasingly being employed in finite element computations in structural engineering.[7] This paper describes ways in which the exact method can be extended to run efficiently on parallel multiple instruction, multiple data (MIMD) computers. The descriptions assume a distributed memory system with message passing between the processors, but the method could also be implemented on a shared memory system with appropriate task synchronization. Recent coarse grain and fine grain parallel methods are reviewed and enhanced and are then combined to form a novel hybrid method employing both kinds of parallelism simultaneously, in a manner appropriate to the problems being solved.

The parallel methods have been incorporated in an earlier sequential Fortran program[8] for the analysis of plane frames. The methods are evaluated by measuring speedup and efficiency when this program is run, with the number of processors varied, to locate natural frequencies of frames of different sizes.

## Coarse Grain Parallelism

The authors have recently published a coarse grain parallel method[9] for locating a large number of natural frequencies of a structure. This method converges on individual frequencies using the multiple determinant parabolic interpolation method,[5] which combines bisection and parabolic interpolation.

Each of the processors is initially allocated (as nearly as possible) an equal number of frequencies to find. Often there are large variations in the time taken by each processor to find its frequencies because the convergence algorithm[5] requires significantly different numbers of iterations (and hence solution times) for each frequency, especially when coincident or close frequencies are present. To overcome this difficulty, frequencies may be redistributed among the processors so as to reduce the overall time taken for the solution. This redistribution of work requires only a few short messages to be passed between the processors whenever a processor becomes idle and often enables all of the processors to finish their work almost simultaneously. It has been shown[9] to yield the greatest benefits when the total number of eigenvalues required is several times greater than the number of processors used. If only a few eigenvalues are required, there can be a serious loss of efficiency if too many processors are used.

The method has since been improved by the addition of the two refinements presented later, which were already known[4,6] to improve solution times when using sequential computers.

### Accelerated Convergence on Coincident and Close Eigenvalues

Many structures have groups of coincident or close eigenvalues. The original form of the multiple determinant parabolic interpolation method[5] converges on each eigenvalue by bisection until it has been isolated from its neighbors. Subsequent parabolic interpolation on det $\underline{K}$ (or another suitable determinant) usually achieves convergence to the required accuracy in fewer iterations than would be required by bisection alone. However, a group of coincident eigenvalues must be found entirely, and a group of close eigenvalues almost entirely, by bisection.

The method was subsequently refined[6] by allowing parabolic interpolation using $\pm|\det\underline{K}|^{1/r}$ (i.e., on the $r$th root of the absolute value of det $\underline{K}$) to accelerate convergence on a group of $r$ apparently coincident eigenvalues. This refinement enables a group of coincident eigenvalues to be located as rapidly as a single well-separated one, with convergence on close eigenvalues being improved by several other enhancements. This form of accelerated convergence has now been included in the coarse grain parallel method described earlier in this section, as well as in the fine grain and hybrid methods presented in subsequent sections.

### Gauss–Doolittle Triangulation

Successful convergence on eigenvalues with the Wittrick–Williams algorithm[1] depends largely on the signs of the diagonal terms of $\underline{K}^{\Delta}$. Therefore the standard form of Gauss elimination, in which rows are pivotal in their natural order with no pivoting or row scaling, is often used. As each row is pivotal, elements in succeeding rows are each updated by a correction term involving

Received Feb. 19, 1994; presented as Paper 94-1349 at the AIAA/ASME/ASCE/AHS/ASC 35th Structures, Structural Dynamics, and Materials Conference, Hilton Head, SC, April 18–20, 1994; revision received April 27, 1995; accepted for publication April 27, 1995. Copyright © 1995 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Lecturer, Cardiff School of Engineering, College of Cardiff, P.O. Box 917. Member AIAA.

†Research Associate, Cardiff School of Engineering, College of Cardiff, P.O. Box 917.

‡Professor, Head of Structures Division, Cardiff School of Engineering, College of Cardiff, P.O. Box 917.

elements of the pivotal row. The authors' Gauss–Doolittle method[4] updates a typical matrix element by correction terms from $v$ (>1) pivotal rows simultaneously. Although the method performs essentially the same computations as Gauss elimination, the solution time is lower because fewer memory accesses are required. Time savings of 25% over Gauss elimination have been achieved on a sequential computer[4] for a large structural problem, using $v = 6$.

This sequential form of Gauss–Doolittle triangulation has been included within the coarse grain parallel method described earlier in this section. It has also been successfully parallelized to form the fine grain method of the following section and to become a component of the hybrid method presented later. The case $v = 3$ has been used in this paper to correspond conveniently with the three degrees of freedom per node of the example problems.

## Fine Grain Parallelism

In many vibration problems, and in virtually all buckling problems, only the lowest (fundamental) eigenvalue is of interest, and so the coarse grain method for finding many eigenvalues described in the previous section cannot be used. Instead, a fine grain parallel method can be used to triangulate $\underline{K}$ efficiently within each iteration of the Wittrick–Williams algorithm. Many papers[10–12] discuss parallel matrix triangulation, and large structural finite element analyses have been performed efficiently using parallel forms of Gauss elimination[13] and Choleski decomposition.[14] The authors have recently developed[15] the following parallel form of their Gauss–Doolittle triangulation method.[4]

Suppose that a distributed memory parallel computer with $q$ processors is used to triangulate an $n \times n$ real symmetric (or complex Hermitian) matrix

$$\underline{K} = \{k_{ij};\ i, j = 1, \ldots, n\} \tag{1}$$

Let $\underline{K}$ be partitioned into blocks of $v$ rows, so that the $i$th block $\underline{b}_i$ ($i = 1, 2, \ldots, n/v$) comprises the $\{(i-1)v+1\}$th, $\{(i-1)v+2\}$th, $\ldots$, $iv$th rows of $\underline{K}$. In many structural applications, $\underline{K}$ has a banded form such that $k_{ij}$ is zero if $|j - i|$ exceeds the half-bandwidth $m$. This form is assumed later but is not a requirement for the method. (For simplicity it is also assumed that $v$ and $q$ are factors of $m$ and $n$.) The $i$th block of $v$ rows in $\underline{K}^\Delta$ is given by

$$\underline{b}_i^\Delta = \underline{b}_i + \sum_{u=1}^{i} \underline{c}_i^u \tag{2}$$

where $\underline{c}_i^u$ is the contribution made to the $i$th block when the $v$ rows of the $u$th block are pivotal, noting that $\underline{c}_i^u = \underline{0}$ for $i > u + (m/v)$.

Each processor $i$ ($i = 1, \ldots, q$) initially stores an $n \times n$ matrix $\underline{K}_i$ whose $i$th, $(q+i)$th, $(2q+i)$th, $\ldots$, $\{(n/v) - q + i\}$th blocks of $v$ rows correspond to those of $\underline{K}$ and whose remaining rows are null, as illustrated in Fig. 1. Processor 1 first calculates $\underline{c}_1^1$ and forms $\underline{b}_1^\Delta$ in the first block of $\underline{K}_1$. It next calculates the contributions $\underline{c}_2^1, \underline{c}_3^1, \ldots, \underline{c}_q^1$ in blocks $2, 3, \ldots, q$ of $\underline{K}_1$ and sends a message containing these blocks to processor 2, as indicated by an arrow in Fig. 1. It then continues to calculate the remaining contributions $\underline{c}_{q+1}^1, \underline{c}_{q+2}^1, \ldots$, accumulating them in blocks $(q+1), (q+2), \ldots$, of $\underline{K}_1$.

When processor 2 receives the contributions to blocks $2, 3, \ldots, q$ from processor 1, it accumulates them in blocks $2, 3, \ldots, q$ of $\underline{K}_2$. It next calculates $\underline{c}_2^2$ and forms $\underline{b}_2^\Delta$ in block 2. It then calculates in order the contributions $\underline{c}_3^2, \underline{c}_4^2, \ldots$, and accumulates them in blocks $3, 4, \ldots$, of $\underline{K}_2$. When block $q+1$ has been updated, processor 2 sends the contents of blocks $3, 4, \ldots, q+1$ to processor 3 (see the arrow in Fig. 1), which can now calculate $\underline{c}_3^3$ and form $\underline{b}_3^\Delta$ in block 3 of $\underline{K}_3$, etc.

The remaining processors commence their work in the same way. When processor $q$ has formed $\underline{b}_q^\Delta$ and the contributions from block $q$ to the next $(q-1)$ blocks, it sends a message to processor 1 (see the long arrow in Fig. 1), which can now form $\underline{b}_{q+1}^\Delta$. Successive blocks of rows are made pivotal by successive processors, and the final triangulated matrix $\underline{K}^\Delta$ is retrieved by taking blocks $\underline{b}_i^\Delta, \underline{b}_{q+i}^\Delta, \underline{b}_{2q+i}^\Delta, \ldots, \underline{b}_{(n/v)-q+i}^\Delta$ from $\underline{K}_i$ (for $i = 1, 2, \ldots, q$).

Each processor receives messages only from its immediate predecessor and sends messages only to its immediate successor, so that the method is ideally suited to architectures having a natural ring configuration. The communications overhead increases with the number of processors, with a consequent loss of efficiency. The "staggered start" just described incurs a further time penalty that increases with $q$ and $v$ but is unlikely to be significant for large problems.

A more serious overhead arises if processors have to wait whenever a message is due to arrive. This occurs, for example, if processor $i$ has completed calculating the contributions $\underline{c}_i^i, \underline{c}_{i+1}^i, \ldots, \underline{c}_{i+(m/v)}^i$ before the other $(q-1)$ processors have each (in sequence) read a message from the previous processor, calculated contributions to $q$ blocks, and sent a message to the next processor. Therefore, for a matrix of any given order and bandwidth, a serious loss of efficiency occurs if too many processors are used to perform the triangulation of $\underline{K}$.

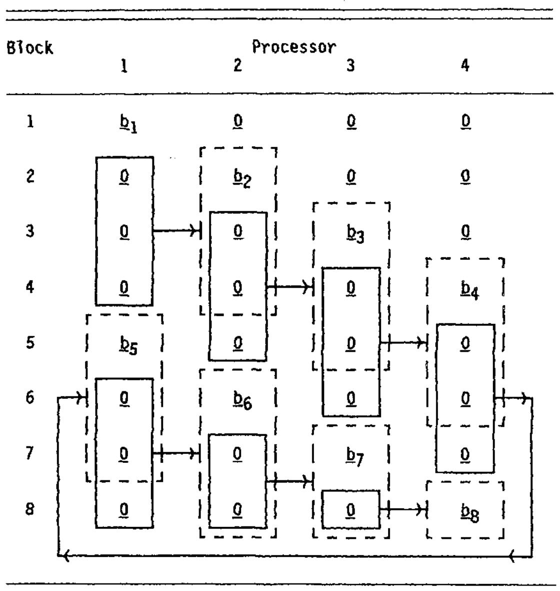


**Fig. 1 Initial contents of $\underline{K}_i$ in each of the $q$ processors for parallel Gauss–Doolittle triangulation, with $(n/v) = 8$ and $q = 4$. The blocks in the solid boxes are initially null but are updated during triangulation and passed in messages to adjacent processors as indicated, where they are added to the contents of the dashed boxes.**

The initial implementation[15] of the parallel Gauss–Doolittle triangulation method demonstrated its applicability to real symmetric (or complex Hermitian) matrices of large order $n$ and half-bandwidth $m$. Provided only a limited number of processors is used, the method is also useful in typical structural applications where $m \ll n$, for which the following refinements have been made.

The computer memory requirements of the Wittrick–Williams algorithm may be reduced[2,8] by assembling and triangulating the structural stiffness matrix $\underline{K}$ node by node and storing only an "active triangle" of $m + 1$ rows and columns. In the refined fine grain parallel implementation, each block of rows $\underline{b}_i$ corresponds to a node of the structure. Each processor holds a copy of the structure's connection list and is responsible for assembling only those blocks $\underline{b}_i$ that it will make pivotal, i.e., as indicated in Fig. 1. The assembly and triangulation are performed node by node, enabling a compact active triangle storage scheme to be used, in which the assembly of block $\underline{b}_i$ is only completed immediately before the receipt of the message enabling the rows of $\underline{b}_i$ to become pivotal.

## Hybrid Coarse and Fine Grain Parallelism

Coarse grain and fine grain parallel methods have been presented in previous sections. Each method is suited to a particular class of problems and performs well on sufficiently large problems of that class. But both suffer a loss of efficiency when an excessive number of processors is employed on smaller problems. Therefore a novel hybrid approach including both forms of parallelism is now presented, in which a given eigenvalue problem may be solved on a given number of processors using either the coarse grain method, or the fine grain method, or a suitable combination of both, as follows.

Calculation of a large number of natural frequencies of a small structure is often best performed using only coarse grain parallelism, each processor being initially allocated an equal number of frequencies and using sequential Gauss–Doolittle triangulation, with the opportunity to redistribute frequencies to idle processors.

Calculation of the lowest critical buckling load for a large structure uses only fine grain parallelism, with all of the processors cooperating in parallel Gauss–Doolittle triangulation at each iteration of the Wittrick–Williams algorithm.

Otherwise, problems are often best solved using both forms of parallelism together, for example by regarding a 16-processor parallel computer as 4 clusters of 4 processors. Each cluster is allocated a set of eigenvalues by the coarse grain method, and the processors within the cluster cooperate in fine grain parallel Gauss–Doolittle triangulation to find them. The results presented in the following section indicate how to choose appropriate numbers and sizes of clusters to make best use of this hybrid parallel approach. In these results $t_{pq}$ is the elapsed time taken by the hybrid parallel method to solve a problem using $p$ clusters of $q$ processors, given overall bounds on the eigenvalues. The limiting cases $t_{p1}$ and $t_{1q}$ correspond to solutions purely by the coarse grain and fine grain methods, respectively. The success of the parallelization is measured in terms of speedup $S_{pq}$ and efficiency $E_{pq}$, defined by

$$S_{pq} = t_{11}/t_{pq} \tag{3}$$

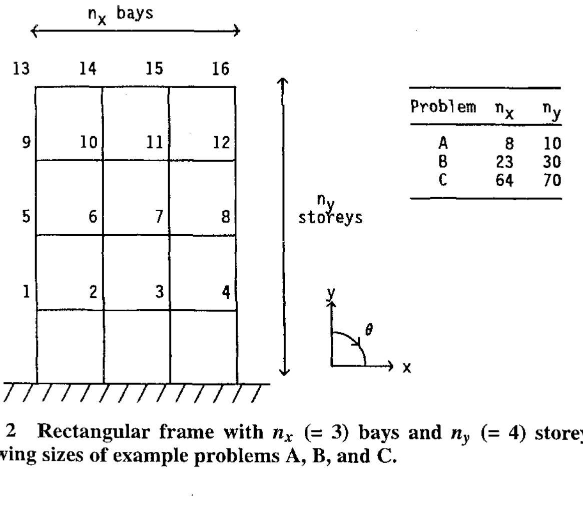
$$E_{pq} = 100\% \times S_{pq}/pq \tag{4}$$

with the ideal case of linear speedup represented by $S_{pq} = pq$.

## Results and Discussion

Solution times have been obtained for finding, to an accuracy of 1 in $10^9$, the first 16 natural frequencies of the plane frames A, B, and C of Fig. 2, in which all of the members are identical and the nodes are numbered along successive storeys as shown to minimize the bandwidth of $\underline{K}$, each node having the 3 degrees of freedom $x$, $y$, and $\theta$. The results for problems A and B were obtained using up to 16 processors on an nCUBE-2 parallel computer with a hypercube configuration. For comparison, the much larger problem C was solved using up to 16 processors on a more powerful distributed memory machine, the Transtech Paramid.

The coarse grain parallel method was evaluated by measuring the solution time $t_{p1}$ for the three problems with $p = 1, 2, 4, 8$, and 16 processors. These times are listed in Table 1, together with values of speedup $S_{p1}$, efficiency $E_{p1}$, and the total number of iterations of the Wittrick–Williams algorithm. Figure 3 plots speedup against $p$. The results demonstrate that high efficiencies are possible with this coarse grain method, partly because there is relatively little communication between processors. Increasing the number of processors reduces the solution time at the expense of a decrease in efficiency.



| Problem | $n_x$ | $n_y$ |
|---|---|---|
| A | 8 | 10 |
| B | 23 | 30 |
| C | 64 | 70 |

Fig. 2 Rectangular frame with $n_x$ (= 3) bays and $n_y$ (= 4) storeys, showing sizes of example problems A, B, and C.

Table 1 Solution time $t_{p1}$ (seconds), speedup $S_{p1}$, efficiency $E_{p1}$ (%), and number of iterations for problems A, B, and C of Fig. 2, solved using the coarse grain parallel method on $p$ (= 1, 2, 4, 8, or 16) processors. For problems A and B, comparative results using the previously published form of the method[9] are shown in italics, followed by the percentage time savings due to the refinements of this paper

| | Number of processors $p$ | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | Problem |
| $t_{p1}$ | 54.5 | 30.2 | 19.0 | 11.8 | 6.5 | |
| $S_{p1}$ | 1.00 | 1.80 | 2.87 | 4.62 | 8.38 | |
| $E_{p1}$ | 100.0 | 90.2 | 71.7 | 57.7 | 52.4 | |
| Iters. | 121 | 125 | 136 | 149 | 181 | |
| $t_{p1}$ | *58.3* | *32.3* | *20.4* | *12.7* | *6.9* | A |
| $S_{p1}$ | *1.00* | *1.80* | *2.86* | *4.59* | *8.45* | |
| $E_{p1}$ | *100.0* | *90.2* | *71.4* | *57.4* | *52.8* | |
| Iters. | *121* | *125* | *136* | *150* | *181* | |
| Saving,% | 6.5 | 6.5 | 6.9 | 7.1 | 5.8 | |
| $t_{p1}$ | 2229.0 | 1171.0 | 623.5 | 377.8 | 208.2 | |
| $S_{p1}$ | 1.00 | 1.90 | 3.57 | 5.90 | 10.71 | |
| $E_{p1}$ | 100.0 | 95.2 | 89.4 | 73.7 | 66.9 | |
| Iters. | 117 | 123 | 130 | 146 | 170 | |
| $t_{p1}$ | *2481.0* | *1293.0* | *699.7* | *424.0* | *233.4* | B |
| $S_{p1}$ | *1.00* | *1.92* | *3.55* | *5.85* | *10.63* | |
| $E_{p1}$ | *100.0* | *95.9* | *88.6* | *73.1* | *66.4* | |
| Iters. | *117* | *122* | *129* | *145* | *169* | |
| Saving,% | 10.2 | 9.4 | 10.9 | 10.9 | 10.8 | |
| $t_{p1}$ | 9086.0 | 4893.0 | 2721.0 | 1731.0 | 1134.0 | |
| $S_{p1}$ | 1.00 | 1.85 | 3.34 | 5.25 | 8.01 | C |
| $E_{p1}$ | 100.0 | 92.8 | 83.5 | 65.6 | 50.1 | |
| Iters. | 117 | 123 | 132 | 160 | 199 | |

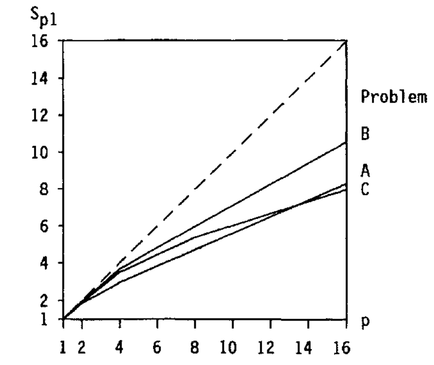


Fig. 3 Speedup $S_{p1}$ for the coarse grain parallel method using $p$ processors. (Linear speedup shown dashed.)

Comparison of the results for problems A and B indicates an increase in efficiency with problem size. The lower efficiencies for problem C are largely due to differences in the relative speeds of the two computers for calculation and communication, which will be discussed in a forthcoming publication.[16]

Table 1 also lists, in italics, comparative results for problems A and B obtained using the earlier form of the coarse grain method,[9] i.e., without the refinements of accelerated convergence on close and coincident eigenvalues and Gauss–Doolittle triangulation. The refined convergence method has a negligible effect on the iteration counts for these example problems that have no coincident or very close natural frequencies in the range considered. Therefore the table shows that the Gauss–Doolittle triangulation refinement yields time savings of between 6 and 10% over the earlier method that are largely independent of the number of processors used.

The fine grain parallel method was evaluated by measuring solution time $t_{1q}$ for the three problems with $q = 1, 2, 3, 4$, and 5 processors. Values of $t_{1q}$, $S_{1q}$, and $E_{1q}$ are given in Table 2, where the best solution times achieved for each problem are underlined. Figure 4 plots speedup against $q$. As with the coarse grain method, higher efficiencies are achieved on larger problems, but in this case when more than two–four processors are used, the solution times cease to

Table 2 Solution time $t_{1q}$ (seconds), speedup $S_{1q}$, and efficiency $E_{1q}$ (%) for problems A, B, and C of Fig. 2, solved using the fine grain parallel method on $q$ (= 1, 2, 3, 4, or 5) processors. The best times achieved for each problem are underlined

| | Number of processors $q$ | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | Problem |
| $t_{1q}$ | 54.5 | 31.3 | 27.2 | 35.8 | 43.5 | |
| $S_{1q}$ | 1.00 | 1.74 | 2.00 | 1.52 | 1.25 | A |
| $E_{1q}$ | 100.0 | 87.1 | 66.8 | 38.1 | 25.1 | |
| $t_{1q}$ | 2229.0 | 1148.0 | 779.5 | 753.8 | 959.8 | |
| $S_{1q}$ | 1.00 | 1.94 | 2.86 | 2.96 | 2.32 | B |
| $E_{1q}$ | 100.0 | 97.1 | 95.3 | 73.9 | 46.4 | |
| $t_{1q}$ | 9086.0 | 4867.0 | 5528.0 | 8348.0 | 10717.0 | |
| $S_{1q}$ | 1.00 | 1.87 | 1.64 | 1.09 | 0.85 | C |
| $E_{1q}$ | 100.0 | 93.3 | 54.8 | 27.2 | 17.0 | |

Table 3 Solution time $t_{pq}$ (seconds), speedup $S_{pq}$, and efficiency $E_{pq}$ (%) for problems A, B, and C of Fig. 2, solved using the hybrid parallel method on $pq$ (= 1, 2, 4, 8, or 16) processors, divided into $p$ clusters, each comprising $q$ (= 1, 2, or 4) processors used for the fine grain method. The best times achieved for each value of $pq$ are underlined

| | Number of processors $pq$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | Problem | $q$ |
| $t_{p1}$ | 54.5 | 30.2 | 19.0 | 11.8 | 6.5 | | |
| $S_{p1}$ | 1.00 | 1.80 | 2.87 | 4.62 | 8.38 | A | 1 |
| $E_{p1}$ | 100.0 | 90.2 | 71.7 | 57.7 | 52.4 | | |
| $t_{p2}$ | —— | 31.3 | 17.4 | 11.1 | 7.0 | | |
| $S_{p2}$ | —— | 1.74 | 3.13 | 4.91 | 7.79 | A | 2 |
| $E_{p2}$ | —— | 87.1 | 78.3 | 61.4 | 48.7 | | |
| $t_{p4}$ | —— | —— | 35.8 | 20.0 | 12.9 | | |
| $S_{p4}$ | —— | —— | 1.52 | 2.73 | 4.22 | A | 4 |
| $E_{p4}$ | —— | —— | 38.1 | 34.1 | 26.4 | | |
| $t_{p1}$ | 2229.0 | 1171.0 | 623.5 | 377.8 | 208.2 | | |
| $S_{p1}$ | 1.00 | 1.90 | 3.57 | 5.90 | 10.71 | B | 1 |
| $E_{p1}$ | 100.0 | 95.2 | 89.4 | 73.7 | 66.9 | | |
| $t_{p2}$ | —— | 1148.0 | 598.5 | 323.8 | 196.5 | | |
| $S_{p2}$ | —— | 1.94 | 3.72 | 6.88 | 11.34 | B | 2 |
| $E_{p2}$ | —— | 97.1 | 93.1 | 86.0 | 70.9 | | |
| $t_{p4}$ | —— | —— | 753.8 | 393.4 | 212.9 | | |
| $S_{p4}$ | —— | —— | 2.96 | 5.67 | 10.47 | B | 4 |
| $E_{p4}$ | —— | —— | 73.9 | 70.8 | 65.4 | | |
| $t_{p1}$ | 9086.0 | 4893.0 | 2721.0 | 1731.0 | 1134.0 | | |
| $S_{p1}$ | 1.00 | 1.85 | 3.34 | 5.25 | 8.01 | C | 1 |
| $E_{p1}$ | 100.0 | 92.8 | 83.5 | 65.6 | 50.1 | | |
| $t_{p2}$ | —— | 4867.0 | 2623.0 | 1484.0 | 970.2 | | |
| $S_{p2}$ | —— | 1.87 | 3.46 | 6.12 | 9.36 | C | 2 |
| $E_{p2}$ | —— | 93.3 | 86.6 | 76.5 | 58.5 | | |
| $t_{p4}$ | —— | —— | 8348.0 | 5125.0 | 3317.0 | | |
| $S_{p4}$ | —— | —— | 1.09 | 1.77 | 2.74 | C | 4 |
| $E_{p4}$ | —— | —— | 27.2 | 22.2 | 17.1 | | |

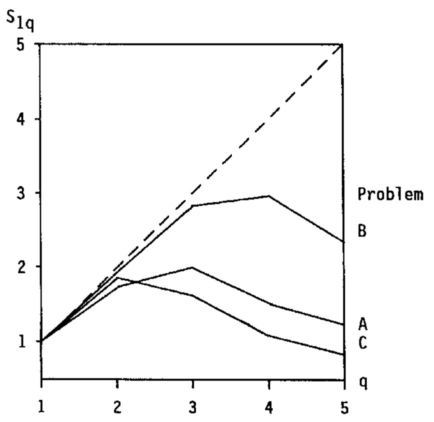


Fig. 4 Speedup $S_{1q}$ for the fine grain parallel method using $q$ processors. (Linear speedup shown dashed.)

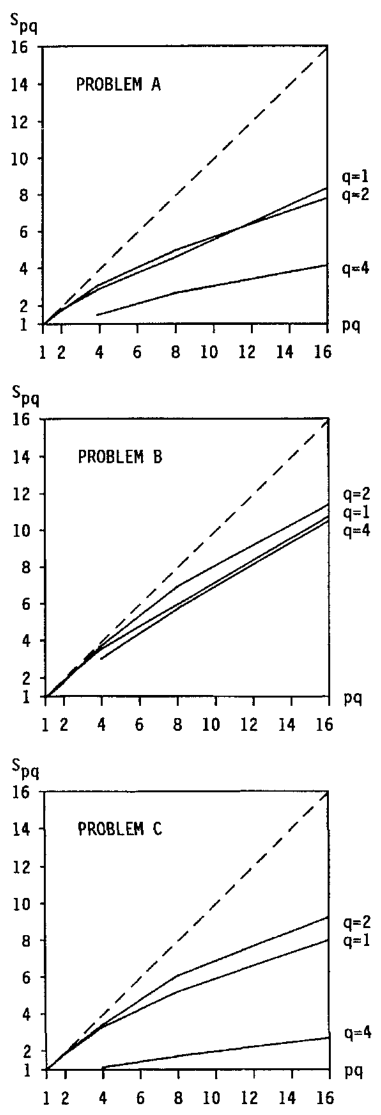


Fig. 5 Speedup $S_{pq}$ for the hybrid parallel method, using $pq$ processors divided into $p$ clusters, each comprising $q$ processors used for the fine grain procedure. (Linear speedup shown dashed.)

improve and the efficiency becomes very poor. Earlier tests[15] gave good efficiencies on up to 5 nCUBE-2 processors when triangulating large, fully populated matrices. The cause is that the fine grain method involves a substantial amount of communication between processors, and a limiting value of $q$ is reached beyond which processors lie idle waiting for messages to arrive. This limiting value depends on the matrix bandwidth and the ratio of processing speed to communications speed for the parallel computer being used.[16]

The hybrid parallel method was evaluated by running the three problems on $pq = 1$, 2, 4, 8, and 16 processors, divided into $p$ clusters of $q = 1$, 2, and 4 processors. Thus for $pq = 8$ each problem was solved three times: 1) using the coarse grain method on eight processors, 2) using four clusters of two processors, and 3) using two clusters of four processors. Values of $t_{pq}$, $S_{pq}$, and $E_{pq}$ are listed in Table 3, where each column relates to one value of $pq$ and the best time achieved for this total number of processors is underlined. Figure 5 shows the speedups attained for each problem.

The results show that when two processors are available there is little to choose between the coarse and fine grain methods for the three example problems. When 4, 8, or 16 processors are available, the hybrid method with clusters of $q = 2$ processors generally gives the fastest solutions, particularly for larger problems, giving time savings of up to 14% over the refined coarse grain method (i.e., over $t_{p1}$ in Table 3) and 24% over the previous coarse grain method evaluated in Table 1. Clusters of $q = 4$ processors give slower solution times for small problems but appear to be becoming competitive for problem B.

The hybrid method on $pq$ processors is essentially a combination of the coarse grain method on $p$ clusters and the fine grain method on $q$ processors. It is therefore surmised that the speedup $S_{pq}$ might be predicted by

$$S_{pq} = S_{p1} S_{1q} \tag{5}$$

This hypothesis is supported by Table 3, where almost all of the values of $S_{pq}$ agree with Eq. (5) to within 5% and most are within 1%. (The larger discrepancies for the $q = 4$ cases of problem C are due to specific hardware characteristics of the Transtech Paramid.) Thus a knowledge of the performance of the coarse grain and fine grain methods on different numbers of processors, as illustrated in Figs. 3 and 4, enables the performance of the hybrid method to be predicted, so that sensible choices can be made for the number $p$ and size $q$ of processor clusters. Future hybrid parallel software could automate the choice of $p$ and $q$ to minimize solution times when running on different total numbers of processors.

## Conclusions

Existing coarse grain and fine grain parallel methods have been used to calculate natural frequencies of plane frame structures on two different parallel (MIMD) computers. The coarse grain method yields high efficiencies, and its solution time has been reduced by 6–10% by refining it to use Gauss–Doolittle matrix triangulation. The fine grain method has been refined to reduce computer memory requirements for typical structural problems, but this method cannot be used efficiently when the number of processors exceeds a problem-dependent limiting value (which can be as low as 2).

The methods have been combined to form a novel hybrid method employing both forms of parallelism simultaneously by dividing the available processors into clusters. Using a cluster size of two fine grain processors, the hybrid method is up to 14% faster than the refined coarse grain method alone and up to 24% faster than the original coarse grain method. These solution times can be predicted from those of the coarse grain and fine grain methods, enabling suitable numbers and sizes of clusters to be chosen.

## Acknowledgments

The authors gratefully acknowledge support from the Engineering and Physical Sciences Research Council under Grant GR/H45490. They also thank the Department of Physics and Astronomy and the Department of Computing Mathematics at the University of Wales College of Cardiff for allowing the use of their computing facilities.

## References

[1]Wittrick, W. H., and Williams, F. W., "A General Algorithm for Computing Natural Frequencies of Elastic Structures," *Quarterly Journal of Mechanics and Applied Mathematics*, Vol. 24, Pt. 3, 1971, pp. 263–284.

[2]Anderson, M. S., and Williams, F. W., "BUNVIS-RG: Exact Frame Buckling and Vibration Program, with Repetitive Geometry and Substructuring," *Journal of Spacecraft and Rockets*, Vol. 24, No. 4, 1987, pp. 353–361.

[3]Williams, F. W., Kennedy, D., Butler, R., and Anderson, M. S., "VICONOPT: Program for Exact Vibration and Buckling Analysis or Design of Prismatic Plate Assemblies," *AIAA Journal*, Vol. 29, No. 11, 1991, pp. 1927, 1928.

[4]Williams, F. W., and Kennedy, D., "Fast Gauss–Doolittle Matrix Triangulation," *Computers and Structures*, Vol. 28, No. 2, 1988, pp. 143–148.

[5]Williams, F. W., and Kennedy, D., "Reliable Use of Determinants to Solve Non-Linear Structural Eigenvalue Problems Efficiently," *International Journal for Numerical Methods in Engineering*, Vol. 26, No. 8, 1988, pp. 1825–1841.

[6]Kennedy, D., and Williams, F. W., "More Efficient Use of Determinants to Solve Transcendental Structural Eigenvalue Problems Reliably," *Computers and Structures*, Vol. 41, No. 5, 1991, pp. 973–979.

[7]Farhat, C., and Roux, F.-X., "Implicit Parallel Processing in Structural Mechanics," *Computational Mechanics Advances*, Vol. 2, No. 1, 1994, pp. 1–124.

[8]Williams, F. W., and Howson, W. P., "Compact Computation of Natural Frequencies and Buckling Loads for Plane Frames," *International Journal for Numerical Methods in Engineering*, Vol. 11, No. 7, 1977, pp. 1067–1081.

[9]Watkins, W. J., Kennedy, D., and Williams, F. W., "Efficient Parallel Solution of Non-Linear Structural Eigenvalue Problems," *Information Technology for Civil and Structural Engineers, Proceedings of Civil-Comp 93, The 5th International Conference on Civil and Structural Engineering Computing*, Civil-Comp Press, Edinburgh, Scotland, UK, 1993, pp. 235–244.

[10]Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D., *Solving Problems on Concurrent Processors*, Vol. 1, Prentice–Hall, Englewood Cliffs, NJ, 1988, pp. 363–397.

[11]Golub, G. H., and van Loan, C. F., *Matrix Computations*, 2nd ed., Johns Hopkins Univ. Press, Baltimore, MD, 1989, pp. 260–330.

[12]Loli Piccolomini, E., and Barulli, M., "Parallel Triangular Solvers on Transputer Networks," *Transputer Applications and Systems '94*, IOS Press, 1994, pp. 805–816.

[13]Chien, L. S., and Sun, C. T., "Parallel Processing Techniques for Finite Element Analysis of Nonlinear Large Truss Structures," *Computers and Structures*, Vol. 31, No. 6, 1989, pp. 1023–1029.

[14]Storaasli, O. O., Nguyen, D. T., and Agarwal, T. K., "Parallel-Vector Solution of Large-Scale Structural Analysis Problems on Supercomputers," *AIAA Journal*, Vol. 28, No. 7, 1990, pp. 1211–1216.

[15]Watkins, W. J., Kennedy, D., and Williams, F. W., "Efficient Parallel Gauss–Doolittle Matrix Triangulation" (submitted for publication).

[16]Watkins, W. J., Kennedy, D., and Williams, F. W., "On Estimating Machine Dependency of Fine and Coarse Grain Parallel Structural Computations" (submitted for publication).