INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 9, 731–740 (1989)

MULTIFRONTS AND TRANSPUTER NETWORKS FOR SOLVING FLUID MECHANICAL FINITE ELEMENT SYSTEMS

R. G. MILES* AND S. P. HAVARD

Department of Mathematics and Computing, Polytechnic of Wales, Pontypridd, Mid. Glam., U.K.

SUMMARY

This paper is concerned with the implementation of a multifrontal solver on a network of transputers. We briefly discuss the transputer, outline the frontal and multifrontal schemes and consider the implementation of these schemes on the network. Results are presented for two test problems in fluid mechanics showing that the solver displays close to linear speed-up.

KEY WORDS Multifrontal solver Transputer networks Fluid mechanics Finite element method

1. INTRODUCTION

Currently the solution of large fluid mechanics problems requires the use of large, powerful computers which are expensive to use and may involve long 'turn-round' times due to unavailability. A number of multiprocessor computers have been developed. These computers offer comparable speed to large serial computers at a fraction of the cost. Also the use of these computers results in a considerable reduction in total elapsed time for the above problems.

The introduction of the transputer, a basic building block for forming parallel processors, has made possible a powerful workstation hosted by a personal computer. This type of architecture is scalable in the sense that more processors may be added without any degradation in performance. In this paper we will present results using up to 16 transputers; however, we believe that the work will be applicable for a considerably higher number of processors.

To make full use of the available power, attention must be focused on developing new algorithms for these types of architecture. The fundamental characteristics of finite element method (FEM) calculations are that they are computationally intensive and use a lot of storage. They also contain a lot of inherent parallelism.

The purpose of this paper is to consider the feasibility of using a network of transputers to solve some problems in fluid mechanics using the FEM. Here we will restrict ourselves to solving problems with convex domains. The extension of this work to non-convex domains will be the subject of a subsequent paper. In Section 2 the transputer and transputer networks will be briefly described along with the characteristics of these networks which are important for algorithm design.

Within the field of structural mechanics the favoured FEM solution procedure is to partition the finite element mesh into substructures, to eliminate the interior variables and then to solve for

* Present address: The Transputer Centre, Bristol Polytechnic, Bristol BS16 IQY U.K.

0271–2091/89/060731–10$05.00

Received August 1988

the variables on the boundaries between substructures by using a preconditioned conjugate gradient method.[1] The choice of preconditioner is crucial and is the source of a lot of discussion. In this paper we will use the multifrontal method introduced by Duff and Reid[2] and Reid.[3] This method also partitions the finite element mesh and then a frontal method is applied to each substructure to eliminate the interior nodes. This generates a set of substructure matrices which may themselves be considered as superelement matrices. These superelements may again be used with the frontal method to complete the process of elimination. A back-substitution yields the solution. In Section 3 frontal and multifrontal methods will be discussed and in Section 4 we will show how these methods may be implemented on transputer networks. The implementation will be tested on two problems: firstly on Laplace's equation in a square region with Dirichlet boundary conditions and secondly on a driven cavity flow. The results of these tests are given in Section 5. In Section 6 we will conclude by comparing one of the second test results with those from a serial computer and by discussing extensions of this work.

2. TRANSPUTER NETWORKS

The transputer is a single-chip microprocessor which has been designed specifically to facilitate concurrent processing. Each chip contains a processor, memory and communication links. It is by use of these links that connections can be made to other transputers. The T414 has a 32-bit RISC microprocessor, 4 Kbytes of on-chip RAM and four communication links. The chip also has interface circuitry and bus logic to allow for external memory to be attached. On this chip all floating-point operations are performed via software; however, the T800 chip has a 64-bit floating-point hardware unit. The four interface communication links allow for communication between transputers on a point-to-point basis. Since the links and the processor are largely independent, the processor can continue executing other instructions whilst the transputer is passing messages via the links. By passing messages, information may be transferred from one transputer to another, and consequently a network of transputers may be constructed.

The OCCAM programming language was designed prior to the development of the transputer, and transputers implement this language very efficiently. Although other languages, such as FORTRAN and PASCAL, may be used to implement processes on transputers, concurrency is only available through OCCAM. So the creation of parallel processes and the passing of messages must be done through OCCAM, and in this paper all of the algorithms have been implemented in this language.

Various building blocks for constructing transputer networks are now available, the most common being the four-transputer board. The board that we have used has four transputers, each with 1 Mbyte of memory attached, and two of the links on each are wired to adjacent transputers; however, eight links remain unconnected. This network needs to communicate with the outside world and so another transputer is used as a root linking with the host computer. This is a single transputer board with 2 Mbyte of memory attached. The most common host is an IBM-compatible PC, and these boards fit into the expansion slots. So with two extra boards the IBM-compatible PC will support parallel processing (see Figure 1).

More transputers may need to be configured into the network. INMOS and a number of other companies have fabricated boards with many transputers; however, for reasons of general versatility we have used a number of four-transputer boards. The boards are daisy-chained together and by using the unconnected links we may obtain the desired configuration, be it ring, array or modified hypercube. In Figure 2 is shown a simple eight-transputer two-directional ring. It is quite straightforward to write OCCAM processes to route messages in either the clockwise or anticlockwise directions, so that they get to the required processor in the shortest possible time.

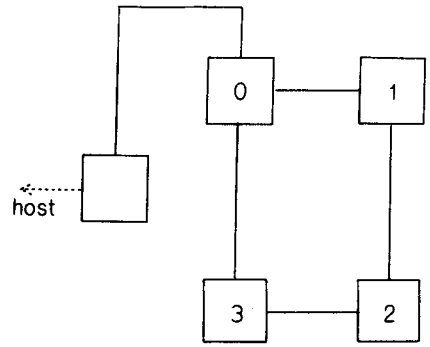

Figure 1. A simple four-transputer network with root to host

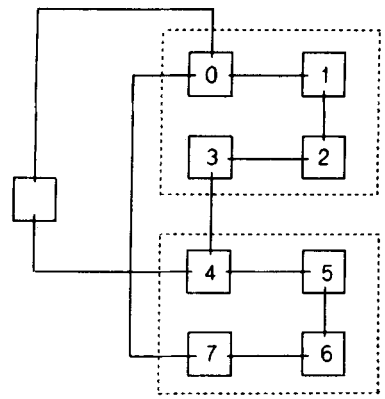

Figure 2. An eight-transputer ring configuration with root

For networks of transputers exceeding eight, a ring configuration is only useful if there is a very small amount of communication or if communication is only with nearest neighbours. For example, with 16 transputers, a ring network has diameter eight, whereas a two-dimensional array has diameter four. There is an INMOS technical note on configuring systems of four-transputer boards.[4]

The results in the last section will be presented for networking four, eight and 16 transputers. In this paper we will develop an algorithm with large granularity so the amount of communication is kept to a minimum. We have performed tests using various configurations, and the time for communication is less than 4% of the total time for even the smallest problem considered in the last section. Consequently the type of configuration is not an issue here. To summarize, the important characteristics of these networks are:

(a) Each transputer has its own memory of 1 Mbyte or more.
(b) Each transputer is controlled by its own set of instructions.
(c) Transputers communicate by passing messages, and it is straightforward to pass messages from one transputer to another in the network.
(d) There may well be a large number of transputers in the network.
(e) There is no shared memory and no shared clock; conseqeuntly each transputer is a totally independent unit which can only communicate with other transputers by message passing.

3. FRONTAL AND MULTIFRONTAL SYSTEMS

The frontal scheme as proposed by Irons[5] is a modified Gauss elimination procedure which is very frequently used with the FEM. The method may be divided into three parts, namely, the assembly of the equations, the elimination and the back-substitution. Irons suggested that elimination should take place whenever possible, rather than assembling the whole system and then beginning the elimination process. The method can be visualized using Figure 3, assuming one degree of freedom at each node. In this figure the element matrices for elements denoted 'A' will have been assembled and contributions made to the system matrix. The contributions to nodes 1, 2, 3, 4, 5 are complete and so the rows associated with these nodes will have been eliminated. Consequently the system matrix will be of order four and will have rows associated with nodes 6, 7, 8, 9. So in this example the front connects nodes 6, 7, 8, 9. As the front progresses across the finite element mesh, elements behind it will have been assembled and contributions made to the system matrix. Those nodes introduced, but not on the front, will have all of their contributions completed and so the rows associated with these nodes will have been eliminated. The elements ahead of the front have yet to be assembled and consequently their nodes are either on the front or have yet to enter the system.

The basic frontal scheme may be summarized as having the following steps:

(a) Assemble the matrix for an element.
(b) Incorporate this into the system matrix.
(c) See if any rows correspond to fully contributed nodes; if so, store the row, the variables associated with it, and the right-hand side, and then eliminate this row.
(d) Continue (a), (b) and (c) until all elements have been assembled, then continue (c) to complete the elimination.
(e) Perform the back-substitution.

In practice a number of improvements have been made to this scheme, the most important of which, for non-positive definite matrices, is to try to retain accuracy by allowing the system matrix to build up until it reaches a level where it will have a number of rows to eliminate. Some of these, enough to allow another element to be assembled, are eliminated using the largest pivots.

This scheme was modified by Hood[6] to treat unsymmetric matrices with symmetric sparsity pattern and by Duff[7,8] who treated the unsymmetric patterns. Pivots are not necessarily diagonal elements and not all fully contributed variables are eliminated. Reid[3] suggested substructuring the finite element problem and used a tree to represent the order of assembly. There may be many

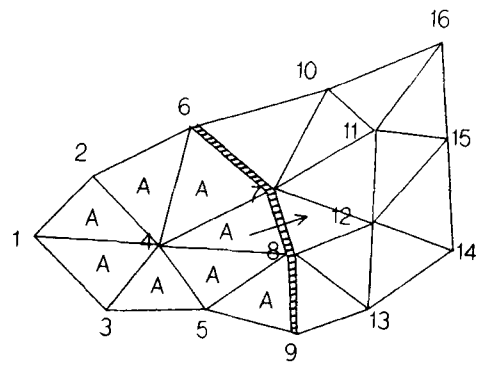

Figure 3. Finite element mesh with one front

frontal matrices and the method is called multifrontal. This may be visualized using Figure 4. Here there are three substructures. The numbers 1, 2, 3 in an element indicate to which substructure it belongs. In substructure 1 the front has assembled three elements and, after performing eliminations, the matrix associated with this front has order three associated with nodes 2, 4, 5. In the other two substructures the fronts have assembled four and seven elements and the frontal matrices are associated with nodes 6, 7, 11, 15 and nodes 9, 8, 7, 11, 15 respectively. In these two substructures the frontal method is complete and these two frontal matrices may be considered as superelement matrices. It may now be decided to merge these two superelement matrices, i.e. to coalesce the two fronts. This process is a small generalization of step (b) above. On performing this, the rows associated with nodes 11 and 15 become fully contributed and then are available for elimination. These two fronts, having been coalesced, have now reduced to the single front connecting nodes 6, 7, 8, 9.

Duff and Reid[2] applied the multifrontal scheme to solve unsymmetric sets of linear equations. They use the flexibility that this introduces to make pivot choices which are economical in arithmetic operations. Duff[9,10] has also considered the application of these schemes for solving unsymmetric sets of linear equations on shared memory multiprocessor systems.

4. MULTIFRONTAL SCHEMES ON TRANSPUTER NETWORKS

When implementing an algorithm on a transputer network, there are three principal considerations that must be taken into account. These are:

(a) granularity
(b) load balancing
(c) an overall organisational structure.

In the FEM a lot of data is manipulated and large matrices are generated. Using relatively small numbers of transputers, most code and data will be located on off-chip memory; consequently we wish to partition the data and replicate the algorithm on each transputer. So an algorithm will be developed with large granularity. The speed at which the scheme works is dependent upon how efficiently each processor is used. If one transputer is given considerably more work to do than the others, and if other processors are waiting for information from this transputer, then they will be idle. So the transputer given the most work may effectively control the network. An algorithm is required which balances the work over the available transputers. As indicated above, the same

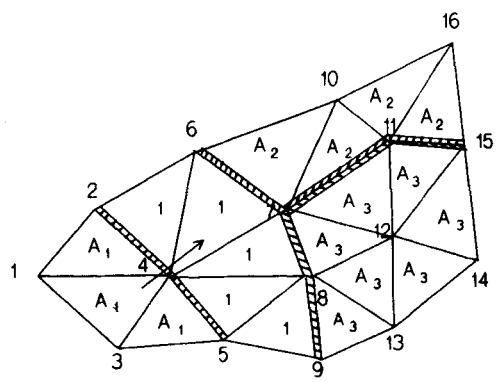

Figure 4. Finite element mesh with three substructures

process is to be allocated to each transputer so a simple data structure is required to organize the passing of messages from one transputer to another. The multifrontal scheme seems very suitable with respect to the above considerations.

A preprocessing algorithm is used which runs on the root transputer to divide the finite element mesh over the number of available transputers and to allow a frontal, or multifrontal, scheme to work on each subdivision. In this study a modified frontal scheme has been placed on each transputer, thus ensuring large granularity. The way the finite element mesh is divided among the transputers is crucial both with respect to load balancing and to the efficiency of the multifrontal method. To balance the load on the network, we want to give each transputer an equivalent amount of work to perform, especially during the first stage of the method when all of the internal variables are being eliminated. Take the example of a square region with n^2 interior nodes, each with one degree of freedom, on a square mesh. If we have eight transputers we cannot easily divide this region to obtain balanced and efficient processing. For example, if the mesh is subdivided by columns then the first and last divisions will have final frontal matrices of order n whereas the ones in between will have order $2n$. Doubling the size of the front crucially affects the efficiency of the method. We require to balance the load and to keep the frontal matrices as small as possible.

We have only worked with convex regions and have decided to use a hub and spokes. An example of this type of mesh for a square region with eight transputers is shown in Figure 5. This type of substructuring ensures a balanced load and minimizes the size of the frontal matrix. The domain is divided into sectors dependent on the number of available transputers, and then the finite element mesh is formed accordingly. Two types of mesh have been used within the sector: a mesh composed entirely of triangles and a mesh composed of quadrilaterals and triangles. In general we prefer the latter type, since this mesh will simplify the generation of contour plots.

Each transputer is sent the information relating to its sector along with information about the global structure of the mesh. The latter is necessary when transputers pass information from one to another.

Each transputer works on its own sector and, when completed, the matrices will be associated only with interface variables (i.e. those on the boundaries between sectors); all other variables will

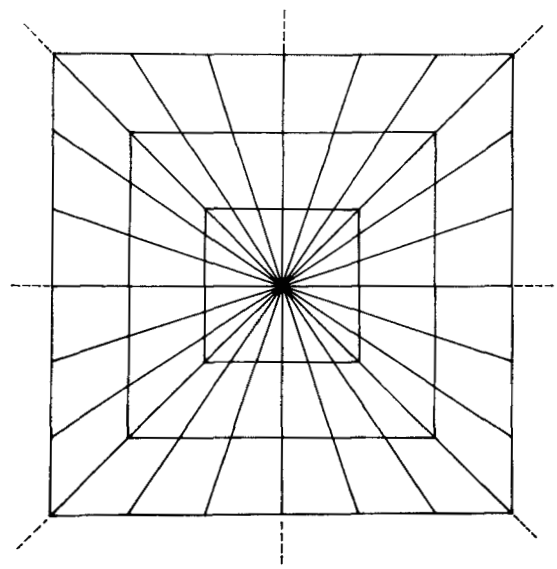

Figure 5. A simple mesh for a square domain with eight transputers

have had their associated rows stored and have been eliminated. Information must then be passed between transputers, and by combining matrices more variables have full contributions and so become available for elimination. The best organizational structure for this is a tree (Figure 6). As the tree is traversed, each transputer knows whether it is to receive or send data and is in a suitable state to do so. For the elimination phase the tree is traversed from the leaves towards the root and then in reverse for the back-substitution.

As the tree is traversed during the elimination phase, the efficiency may be improved by sharing work over the available processors. At level 0 of the tree, each transputer is working independently on its own sector of data. When this has finished, each odd-numbered transputer in Figure 6, T_{2j+1} ($j = 0 \ldots 3$), sends the resulting frontal system matrix to T_{2j}. This local root transputer coalesces the matrices and, if the resulting matrix is sufficiently large, with sufficient rows to be eliminated, it will allocate work to its associated transputer. The associated transputers are shown in dashed line blocks in the figure. The same principle is applied to the other levels of the tree apart from the highest level. In general suppose that we have t transputers. So at level p ($0 < p < \log_2 t$) transputer T_{i2^p} ($i = 0 \ldots t/2^{p-1}$) receives information from $T_{(i2^p + 2^{p-1})}$ and uses some of the associated transputers $T_{(i2^p+j)}$ ($j = 1 \ldots 2^p - 1$) to assist in the elimination.

This sharing of work in the elimination process for higher levels of the tree is best explained at level 1 with, say, transputers T_0 and T_1 working on adjacent sectors of the finite element mesh. When the two transputers have completed the level 0 work, T_1 sends its frontal matrix for sector 1 to T_0. T_0 coalesces the received matrix with its own and this results in a number of variables having full contributions and so being available for elimination. Each of the rows associated with these variables must be eliminated from all of the other rows. At this level two transputers are available, so the local root transputer T_0 forms a packet of all the fully contributed rows and half of the other rows. This it sends to T_1. Now these two transputers can work independently on the elimination. At completion, T_1 returns its half of the matrix to T_0 which together with that on T_0 forms the complete matrix. There is some necessary duplication of work in this process, but a considerable time saving may be achieved.

This same idea may be used at all levels of the tree except the top level. The local root transputer makes the decision of how many assistants to use depending on the size of the matrix and the number of rows to be eliminated. At the top level all rows are now available for elimination, and so very much more communication is required if more than one transputer is used; consequently we have not coded a shared process for this level. We have looked at the time taken in that part of the elimination phase where processor communication is involved, i.e. at

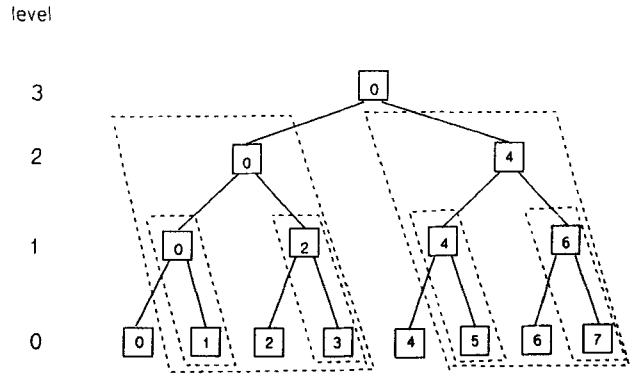

Figure 6. Tree stucture for eight transputers

levels exceeding 0. We have found that up to a 50% saving in time is achieved by using the above approach rather than having no sharing of resources. At each level the number of eliminated rows is stored for use in the back-substitution.

At each level of the back-substitution, the transputer knows how many values are to be determined. These values are then passed to the next transputer so that it can continue the process. During this part of the algorithm some transputers are idle, waiting for values to be passed to them from higher levels; however, at the lowest level all are active, working on their own sector.

5. RESULTS

The first test problem for this algorithm was Laplace's equation in a square region with Dirichlet boundary conditions. We have used a triangular mesh with one degree of freedom per node; consequently the element matrix is extremely easy to form.

First of all the efficiency was examined. To assess this it is useful to divide the algorithm into three stages:

(a) the level 0 eliminations in Figure 6
(b) the other levels of the elimination phase in Figure 6
(c) the back-substitution.

For a number of problems separate timings have been taken in each of these three stages.

The results are presented in Figure 7 for 597, 1193 and 2385 variables using four, eight and 16 transputers respectively. For simplicity we will use the pair (597, 4) to denote a problem with 597

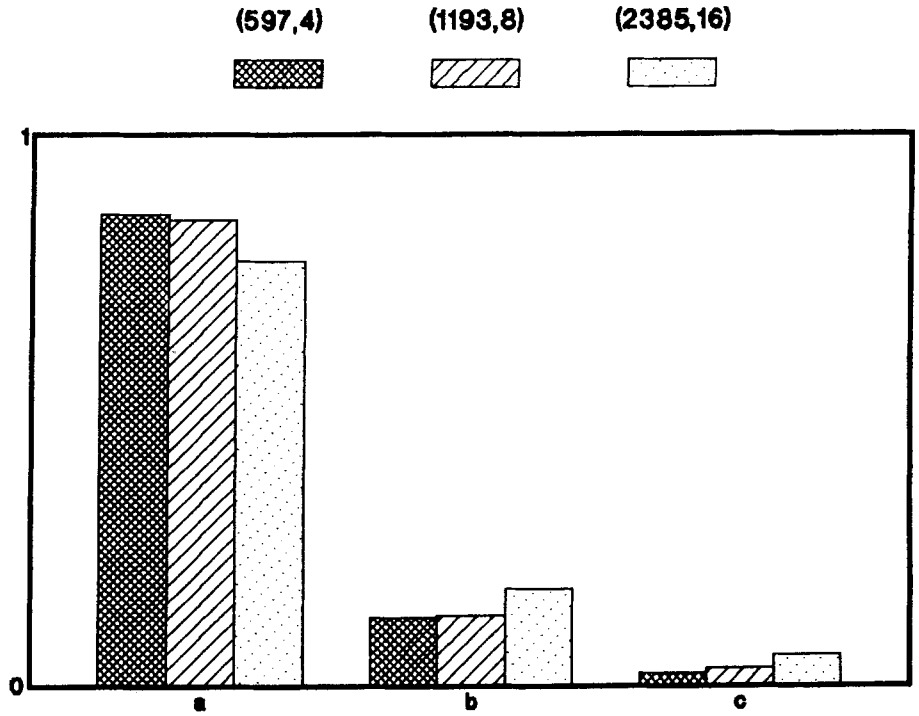

Figure 7. Histograms showing the percentage of time spent in stages (a), (b) and (c)

variables and four transputers. For the stage (a) eliminations the transputers are working entirely independently and during stage (b) sharing of resources may be taking place, so we can estimate that an efficiency greater than 85% is achieved in all cases. The results show almost linear speed-up. Running the (597, 4), (1193, 8) and (2385, 16) problems takes approximately 13·4, 14·6 and 16 s respectively. When the number of transputers is increased, more levels are introduced into the organizational tree. Consequently a smaller percentage of the time is spent in stage (a) and larger percentages in (b) and (c); however, close to linear speed-up is achieved. We have also solved some large problems; for example, the (5745, 16) problem takes approximately 58 s.

The second test problem was a driven flow over a cavity with Reynolds number one. The primitive variable approach is adopted and the non-linearity due to the advection terms of the momentum equations is handled by expressing the problem in iterative form. The preprocessing algorithm determines the mesh given the number of transputers. We have used six-noded triangles with 15 degrees of freedom and eight-noded quadrilaterals with 20 degrees of freedom, so that the pressure shape functions are one order lower than those for velocity. A set of processes has been written, in OCCAM, which are very similar to the NAG finite element software subroutines, to perform all of the necessary operations to assemble an element matrix. In particular the code must first of all determine the type of element so that the correct basis functions and integration routines are selected. It must also assign a suitable numbering system to the variables. These processes replace the element matrix process in the first test and then with a small number of changes the code runs as before. These changes are due principally to there being more than one degree of freedom at each node, and to this degree of freedom being three at some nodes and two at others.

Timings have been taken as for the first test problems and results very similar to those in Figure 7 are achieved. The stage (a) percentages are slightly higher owing to the large amount of processing taking place in the assembly, and consequently slightly higher transputer efficiency is achieved. The times for one iteration of the (591, 4), (1179, 8) and (2355, 16) problems were approximately 20·6, 22 and 23·7 s respectively. Again there is slight degradation when the number of transputers is increased, but close to linear speed-up is achieved. We have also run one larger problem, (5809, 16), and this takes approximately 2 min to complete one iteration.

6. CONCLUSIONS

This paper demonstrates the feasibility of using the multifrontal method as solver for finite element systems, with convex domains, on transputer networks. It has been shown that high transputer efficiency is achieved and that the algorithm displays close to linear speed-up. By way of comparison we have run the second test problem on a VAX 86 50. We used a modified frontal scheme and a mesh with 1202 variables. One iteration on the VAX took 31·5 CPU seconds. This compares with the (1179, 8) problem which took 22 s. On reading the literature for the transputer, it may be considered that the above result is slower than expected; however, it must be remembered that we have nearly all code and data residing on off-chip memory. Hockney and Jesshope[11] have performed some benchmark tests to check the difference in using on-chip and off-chip memory and they report a 3:1 time difference. This, as they point out, is the expected result since, at best, the external memory interface cycles at three transputer clock cycles whereas on-chip memory cycles within a single transputer clock cycle.

It was stated in the Introduction that the work will be applicable to numbers of transputers exceeding 16. This we believe to be the case provided shared processing is used in stage (b) (the stages are indicated in the last section). Then, although a smaller percentage of the time will be

taking place in stage (a), a larger percentage will be taking place in (b) and this will be shared across the processors.

The generalization of this work to non-convex domains is possible by considering them as a union of convex domains and then forming a mesh in each convex region. Sectors which are at the intersection of two regions must be treated carefully since the number of variables and consequently the size of the frontal matrix is considerably bigger. These sectors must have fewer internal variables so as to preserve load balancing. The extension of this work to non-convex regions and consideration of the above issues are to be the subject of further study.

REFERENCES

1. B. Nour-Omid, A. Raefsky and G. Lyzenga, 'Solving finite element equations on concurrent computers', *Parallel Computations and their Impact on Mechanics*, AMD, 86, 1987, pp. 209–227.
2. I. S. Duff and J. K. Reid, 'The multifrontal solution of unsymmetric sets of linear equations', *Harwell Report CSS 133*, 1983.
3. J. K. Reid, 'TREESOLVE, A FORTRAN package for solving large sets of linear finite-element equations', *Harwell Report CSS 155*, 1984.
4. G. Hill, 'Transputer networks using the IMSB003', *Technical Note 13*, INMOS, 1987.
5. B. M. Irons, 'A frontal solution program for finite element analysis', *Int. j. numer. methods eng.*, **4**, 5–32 (1970).
6. P. Hood, 'Frontal solution program for unsymmetric matrices', *Int. j. numer. methods eng.*, **10**, 379–399 (1976).
7. I. S. Duff, 'Design features of a code for solving sparse unsymmetric linear systems out of core', *Harwell Report CSS 89*, 1981.
8. I. S. Duff, 'MA32—a package for solving sparse unsymmetric systems using the frontal method', *Harwell Report AERE R/10079*, 1981.
9. I. S. Duff, 'Parallel implementation of multifrontal schemes', *Harwell Report CSS 174*, 1985.
10. I. S. Duff, 'Multiprocessing a sparse matrix code on the Alliant FX/8', *Harwell Report CSS 210*, 1988.
11. R. W. Hockney and C. R. Jesshope, *Parallel Computers 2*, Adam Hilger, 1988.