Transactions on Information and Communications Technologies vol 3, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

A parallel finite element software package: design and implementation

F.H. Chishti, M. Razaz

School of Information Systems, University of East Anglia, Norwich, UK

ABSTRACT

This paper describes the design and implementation of a parallel finite element software package capable of solving linear boundary value problems on a network of transputers. Many of the design and development issues are discussed that had to be considered in the evolution of efficient software for running on a distributed memory MIMD computer. Typical experimental results are presented and discussed.

INTRODUCTION

Users of finite element techniques would undoubtedly benefit from increased computer resources since this would enable the consideration of more complex problems, finer meshes, larger models and higher order elements and will result in faster solution times. Parallel processing offers a natural approach to improving computational power and achieving significantly improved performance. Therefore there is a strong need to implement finite element programs to run on multi-processors machines.

Despite the simplicity of the parallel processing concept, its implementation is not so straightforward. The exploitation of parallelism within algorithms and the efficient use of the hardware calls for a much deeper understanding of such issues in order to derive any benefit.

This paper describes the design and implementation of a parallel finite element software package capable of solving linear boundary value problems on a network of transputers. In particular, we consider the general form of the 2-D *quasiharmonic* equation as given by Equation (1), which is applicable to a wide range of problems including heat flow, electronic device simulation, fluid flow, electromagnetics, structural analysis and computational mechanics.

$$-\frac{\partial}{\partial x}\left(\alpha_x(x,y)\frac{\partial U(x,y)}{\partial x}\right) - \frac{\partial}{\partial y}\left(\alpha_y(x,y)\frac{\partial U(x,y)}{\partial y}\right) + \beta(x,y)U(x,y) = f(x,y) \quad (1)$$

In heat flow problems, which are of most interest to the authors, $U(x,y)$ represents the steady-state temperature, $\alpha_x(x,y)$ and $\alpha_y(x,y)$ the thermal conductivities, $\beta(x,y)$ the thermal convective coefficient and $f(x,y)$ a heat source. Both Dirichlet and Neumann boundary conditions are considered and the domain is typically a finite, closed region in the x,y-plane which may possibly contain interior holes, as illustrated in Figure 1. Any combination of 2-D linear and quadratic isoparametric elements can be used to discretise the domain.

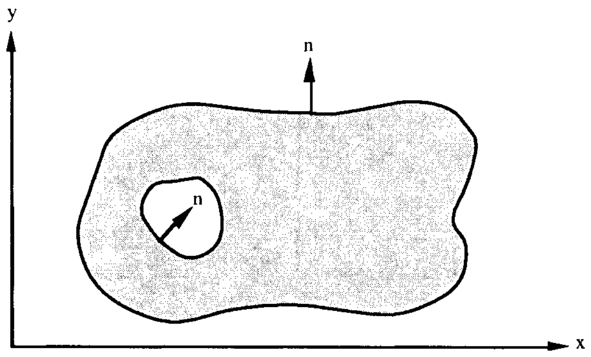

Figure 1. General domain for a 2-D boundary value problem.

$U(x,y)$ on each element can be expanded approximately as,

$$\tilde{U}^{(e)}(x,y) = \sum_{j=1}^{n} a_j \phi_j^{(e)}(x,y) \tag{2}$$

where n is the number of degrees of freedom in the element, $\{\phi_j^{(e)}(x,y)\}$ are the element shape functions and $\{a_j\}$ are constant coefficients.

Using the expansion given in Equation (2) and applying the weighted residual Galerkin method [1], [2] to a typical element governed by Equation (1) we can derive equations of the form;

$$[K]^e \; \{a\}^e = \{F\}^e \tag{3}$$

where,

$$K_{i,j}^{(e)} = \int\int^{(e)} \frac{\partial \phi_i^{(e)}}{\partial x} \alpha_x \frac{\partial \phi_j^{(e)}}{\partial x} dxdy + \int\int^{(e)} \frac{\partial \phi_i^{(e)}}{\partial y} \alpha_y \frac{\partial \phi_j^{(e)}}{\partial y} dxdy + \int\int^{(e)} \phi_i^{(e)} \beta \phi_j^{(e)} dxdy \tag{4}$$

and

$$F_i^{(e)} = \int\int^{(e)} f \phi_i^{(e)} dxdy + \int^{(e)} \tilde{\tau}_{-n}^{(e)} \phi_i^{(e)} ds \tag{5}$$

$\tilde{\tau}_{-n}^{(e)}$ represents the inward normal component of the boundary flux and all the other symbols have their usual meanings [2].

Next we need to develop expressions for the shape functions (which depend upon the elements used in the domain discretisation) and to substitute these terms into the element equations such that we can transform the integrals into a form appropriate for numerical evaluation.

The resultant system equations, derived from the summation of the individual element matrices, will have the form

$$[K] \; \{a\} = \{F\} \tag{6}$$

where the system stiffness matrix [K] has the properties of being symmetric, positive definite and banded, and can be factorised using Gaussian elimination.

DESIGN AND IMPLEMENTATION

Software Structure

The structure of the parallel software has been broken down into independent modules as shown in Figure 2. This modular structure follows closely the design of the sequential software from which it was developed [3].

```
     /*** Pre-processor ***/

   • declarations
   • system initialisation
   • read in mesh data
   • read in physical property data
   • read in load data

     /*** Solution ***/

   • form the system equations
   • apply the boundary conditions
   • solve the system equations

     /*** Post-processor ***/

   • print/plot results
```

Figure 2. General structure of the parallel program.

The sequential software used in this work is based on algorithms, code and good programming practices found in Akin [4], Burnett [1], Carey *et al* [5], Hinton *et al* [6] and Zienkiewicz *et al* [2]. It was observed that the serial FE approach in Burnett [1] would be a useful method to adopt. A command reference language similar to that found in Burnett [1] and Zienkiewicz [2] has been incorporated to make the pre-processing phase more user-friendly. The I/O is handled by this instruction language [3].

The overall software can be divided into three sections: pre-processor, solution and post-processor. We have considered the parallel design and implementation of each of these, concentrating predominantly on the solution phase as this is generally regarded as being the most computationally expensive part. Within the solution phase we initially focused on the parallel implementation of the system equation solver module. This is again by far the most compute intensive part, and that without efficient implementation of this module the parallel implementation of the rest of the program would be rendered worthless.

The solver module incorporates a direct solution strategy based on Gaussian elimination. The parallel implementation of this algorithm and the other solution phase modules are based on the well documented technique of geometric decomposition [7]. Our decomposition strategy considers the solution algorithm first.

There are three basic efficient strategies for implementing Gaussian elimination on a distributed memory MIMD computer, namely block-scattered decomposition, column-scattered decomposition and row-scattered decomposition, see for example, [7], [8], [9]. The major difference between these three approaches is the way in which the system matrices are mapped onto the processor network. The row-scattered decomposition was chosen for its parallel efficiency and ease of implementation and is implemented on a ring processor topology and used as the decomposition strategy for both the solution and pre-solution stages. A detailed description of the parallel Gaussian elimination solution strategy is given in [10]. Various methods for implementing the Gaussian elimination algorithm were tested in order to find the most efficient implementation. The results of our findings are presented in [3] and [11].

Assuming the number of rows per processor $n_P = n / P$ is a natural number (where P is the total number of processors and n is the number of degrees of freedom), the decomposition divides up the matrices evenly such that the rows with indices $p + 1 + iP$, $(0 \leq i \leq n_P - 1$ and $0 \leq p \leq P - 1)$, are placed on processor p. Figure 3 shows how this decomposition is performed across a three processor network (see [10] for further details).

The classical way to generate and assemble the FE matrix equations is based upon the element principle, that is, for every element in the mesh we formulate its stiffness and load vector contributions, and then add these terms into their relevant positions in the system stiffness matrix and load vector.

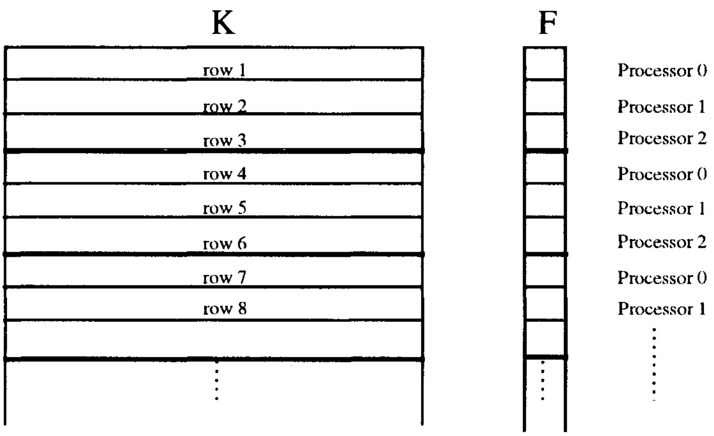

Figure 3. Row-scattered data decomposition across a three processor network.

We considered the implementation of this element-based method on a network of transputers and found that although the element stiffness matrix and load vector can be generated independently for each element, the assembly of the distributed system of equations according to the row-scattered decomposition required a large amount of communication between the processors for assembling the matrix terms into the correct global positions. The large communication overhead results in the parallel element-based scheme being highly inefficient. To avoid this we designed a new algorithm based upon a nodal assembly criterion. The description of which can be found in [3] and [12]. Although the code necessary to implement the node-based method is more verbose, it has the advantage that both the generation and assembly stages can be done independently and thus, no information has to be communicated between the processors.

The independent nature of each of the solution phase modules has made their inclusion into the overall software quite straightforward. Furthermore, as some of the arrays are distributed across the network of processors it is possible to reduce their size. In particular, we can define the data structures for storing the system matrices to be half their original size and therefore make large savings on memory space.

In order to introduce parallelism into parts of the sequential software a *partitioned array* is used to refer to any array that has been split up amongst the processors as part of the geometric decomposition. Whereas in the sequential software, for example, a **DO**-loop might range over all the FE nodes, in the parallel version it might range over a (different) sub-section on each processor.

Extra code was added to the system initialisation module so that each processor can: establish its unique numerical identifier; discover how many other processors are working on the problem; calculate which part of the problem it is to work on; and set up all the variables it will need for subsequent inter-processor communication necessary for the solution phase.

The mesh, property and load data modules remain unchanged from their sequential counterparts. Each processor is run in parallel and executes each of these routines in sequence. For efficient concurrent implementation new parallel code has been written for each of the solution phase modules. At the end of a simulation each processor stores its results in a file. A utility program then takes these files and creates a single results file, which is then used for post-processing such as plotting, printing and so on.

Each processor runs a copy of the same program but works with a different section of the problem. Where necessary, different actions are programmed for different processors by branching on a processor's unique, numerical identifier. This identifier is allocated to each processor, by the system, according to the processor's position in the network. At the beginning of each run (i.e. through the execution of the system initialisation module) a processor discovers this identifier and the number of other processors used, and hence its own role in the overall scheme. This information is used by each processor, for example, to calculate what part of the problem it is to handle.

This new ensemble of code (see Figure 2) outlines the implementation of the software package for solving linear boundary value problems [13].

Implementation Firmware

The targeted multiprocessor machine is the Meiko Computing Surface, a high performance transputer-based parallel computer [14]. The Meiko system consists of 32 T800 transputers (each rated at 10 MIPS and 1.5 MFLOPS), together with a number of older T414 transputers, which perform system functions [15]. Each T800 has 2 megabytes of memory. The architecture of the transputer does not readily permit the use of virtual memory so a program must fit in the available memory on a processor or it cannot be run. The processor also includes a micro-coded scheduler which allows multiple processes to be run on the processor by time-slicing between them.

The Meiko system is hosted by a Sun-4 SPARC workstation, and runs under SunOS4.1 which is a version of the UNIX operating system. The host provides the main user environment, holds all the Meiko related filestore and manages the usage of the Meiko processors.

The provision of tools and facilities required specifically for parallel programming are provided through a program development toolset known as CSTools (*Communicating Sequential Tools*) [14]. This allows parallel code to be written in FORTRAN 77. Parallel programming in CSTools is based on the *Communicating Sequential Processes* model [16]. The basis of the model is the structuring of a single application as a set of ordinary sequential programs, organised to co-operate on improving the performance of a single overall task. These programs exchange data and synchronise only by means of message-passing. To exploit a multi-processor machine, different processes are arranged to execute simultaneously on different processors.

All inter-processor communications are handled by a set of Meiko specific libraries within CSTools. The communication routines are designed to provide a high level model through which the application software can communicate with the hardware. Different modes of communication are available to the programmer. In our work we have adopted a non-blocking synchronous message-passing protocol [17] because, firstly, it enables us to overlap communication work with calculation work and secondly, Meiko recommend it as being the most efficient, safest way to pass messages.

The CSTools routines provide *point-to-point* communications. That is, they allow any one message to be sent from any given process to another specific, given process. Operations such as "broadcast this message to a group of other processes in the network" or "synchronise all processes in the network" or "swap this data between two processes" are not supported as primitives in

Transactions on Information and Communications Technologies vol 3, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

CSTools. Such high-level operations must be programmed explicitly using the lower-level routines provided.

In designing our parallel algorithms we needed to take into account the targeted hardware, as a transputer network forces a specific set of constraints on the user which are not often applicable to other parallel machines currently in use. Some of these constraints are:

- relatively slow inter-processor communications;
- four inter-processor links per transputer;
- simultaneous communication and calculation;
- small amount of memory on each processor;
- finite, often small, number of processors.

These characteristics have to be taken into consideration if optimal efficiency is to be exploited.

The Programming Environment

In order to run the parallel software a configuration file must be created which specifies what code will run on each processor. For example, assuming the program is stored in a binary file called *feprog*, the text shown in Figure 4 would form a valid configuration file.

```
par
     processor 0 for 16 feprog
     network is unarytree
     closeto 0 15
endpar
```

Figure 4. Configuration file for a 16 processor ring network.

This configuration file instructs the operating system to select 16 processors and to connect them in a linear list. The first and the last processors should be connected and the program *feprog* should be run on each processor. Assuming this information was stored in the file 16.par, the software would actually be run by executing the following command:

```
mrun 16.par
```

R

The configuration file is very flexible as it allows the user to define the number of processors, the processor types, the memory specification and other network topologies.

Input and Output

Reading data into the program proved to be straightforward since the underlying parallel software allows multiple processors to open the same file for reading. However, a problem arose when the software was run on a large number of processors, because the host operating system places a limit on the number of files that can be simultaneously open. This necessitated the addition of some extra code to synchronise the processors when opening files for reading or writing. We established that it would be more efficient during the pre-processing stage if every processor were to read in the data file separately. In this way, although there is some duplication in work in generating initial data values, we avoid processors from remaining idle and having to pass large quantities of data between themselves.

At the end of the pre-processing phase [K] and {F} are distributed across the network of processors in the form required by our parallel Gaussian elimination solver. During the software verification stage these matrices were tested for correctness against those derived from the sequential code. This involves each processor writing its output, in a straightforward fashion, to a different file. A simple utility program has been written which can rehash these output files to reproduce [K] and {F} in a single data file.

EXPERIMENTAL RESULTS

In this section we present the performance of various parts of the parallel software for a steady-state heat conduction test problem. Table 1 shows execution times, speed-up and efficiency for the generation and assembly of system equations, the Gaussian elimination, forward reduction, back substitution and the whole program execution times on different numbers of processors.

This example considers the problem of uniform heat generation in a unit square plate. In non-dimensional form the governing differential equation is represented by

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + 8 = 0 \tag{7}$$

and the boundary condition on the perimeter of the square is, temperature T =0.

Myers [18] presents the closed form solution and shows that the steady-state temperature at the centre is 0.5894 for this problem. We can use this value to check the accuracy of our simulations.

The complete domain is discretised using a uniform 32x32 nodal mesh consisting of 961 linear quadrilateral elements. It can be observed that we can also solve over a quarter of this domain as the problem has symmetric geometry, properties and boundary conditions.

The centre temperature derived using our parallel finite element analysis software is 0.5888. This result shows that the centre temperature is in excellent agreement (i.e., to within 3 decimal places) with the exact value stated by Myers.

It is important to note that the sequential generation and assembly times reported in Table 1 are for an element based assembly, as this is the fastest serial method. The parallel timings are for the new node based generation and assembly technique developed. Although, against a node based serial implementation our parallel algorithm showed near linear speed-up, when we compare against the element based routine the efficiency is not so impressive. It should be noted, however, that the serial element based generation and assembly algorithm did not translate well on to our transputer network, a consequence of which was to develop the new node based scheme.

It can be noted that the time taken to execute the whole program is considerably longer than the time for performing all the other steps defined in the tables. This is because the total execution time includes the time taken for I/O during both the pre-processing and post-processing stages (i.e. reading and writing data to the external memory). These times are highly system dependent and can therefore vary from run to run. Furthermore, to explain the trend it was observed that when many processors request disk access a bottle-neck may occur as only a single processor can access the disk at any one time. If there are many processors requesting access then this can lead to substantial processor idle time, and hence the program execution time goes up.

No. of Procs	Time, T (sec)	Speed-up, S	Efficiency, E
	Generation and Assembly		
1	3.81	1.00	100.00
2	4.63	0.82	41.14
4	2.26	1.69	42.15
8	1.19	3.20	40.02
16	0.60	6.35	39.69
	Application of 124 Dirichlet Boundary Conditions		
1	0.35	1.00	100.00
2	0.52	0.67	33.65
4	0.68	0.51	12.87
8	1.18	0.30	3.71
16	2.22	0.16	0.01
	Gaussian Elimination		
1	96.80	1.00	100.00
2	53.18	1.82	91.01
4	27.87	3.47	86.83
8	17.57	5.51	68.87
16	12.35	7.84	48.99
	Forward Reduction		
1	1.43	1.00	100.00
2	3.18	0.45	22.48
4	4.38	0.33	8.16
8	4.56	0.31	3.92
16	4.60	0.31	1.94
	Back Substitution		
1	1.45	1.00	100.00
2	1.39	1.04	52.16
4	1.54	0.94	23.54
8	1.43	1.01	12.67
16	1.41	1.03	6.43
	Whole Program execution excluding I/O		
1	103.84	1.00	100.00
2	62.90	1.65	82.54
4	36.73	2.83	70.68
8	25.93	4.00	50.01
16	21.18	4.90	30.64
	Whole Program execution including I/O		
1	114.39	1.00	100.00
2	77.95	1.47	73.37
4	53.30	2.15	53.65
8	44.40	2.58	32.20
16	51.76	2.21	13.81

Table 1. Performance of the parallel software for linear steady-state heat conduction in a square plate.

The results shown in the table are typical of the trends observed in other examples. Here we summarise our general observations, more detailed analyses can be found in [3]. We have noticed that parallel implementations for Gaussian elimination and the generation and assembly phase perform well on transputer networks, whilst the boundary conditions, forward reduction and back substitution algorithms do not. The total execution times of these parts of the software are relatively minor, and therefore do not adversely affect its performance. However, the time taken for disk I/O can severely affect the overall performance of the software.

CONCLUSION

We have aimed to show that efficient design and implementation of parallel finite element software depends on many factors, such as, the properties of the system equations concerned, the topology of the targeted multi-processor architecture and the number of processing elements available.

Our parallel software package does offer some advantage for solving linear boundary value problems on transputer networks.

The Gaussian elimination and generation and assembly algorithms perform well on transputer networks. These algorithms also happen to form the most computationally expensive part of the overall software. The implementation of the forward reduction and back substitution algorithms and boundary conditions do not perform so well.

Speed-ups for the whole software can be achieved, but this depends on the size of the problem considered and is marred by the I/O constraints. Much better overall speed-ups can be achieved if the I/O constraints are removed.

The parallel algorithms implemented provide encouraging results, however, a comparison between the performance of the sequential and parallel codes reveals that the present implementation only records modest performance improvement. For example, on larger processor networks there is an upper bound on the overall efficiency of our parallel approach to the solution of medium sized finite element problems. Quite often the limitations to efficiency are imposed by hardware restrictions, such as those caused by access to external memory and the speed at which messages can be transmitted between processors.

REFERENCES

1. Burnett, D. S., *Finite Element Analysis from Concepts to Applications*, Addison-Wesley Pub. Co., 1987.
2. Zienkiewicz, O. C. and R. L. Taylor, *The Finite Element Method Volume 1*, McGraw-Hill, 1989.
3. Chishti, F. H., *Finite Element based Thermal Analysis: Sequential and Parallel Methods*, PhD thesis, University of East Anglia, England, Spring 1993.
4. Akin, J. E., *Application and Implementation of Finite Element Methods*, Academic Press Inc., 1984.
5. Carey, G. F. and J. T. Oden, *Finite Elements: Computational Aspects Volume III*, Prentice-Hall, 1984.
6. Hinton, E. and D. R. J. Owen, *Finite Element Programming*, Academic Press, London, 1977.
7. Fox G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors Volume I*, Prentice-Hall International, 1988.
8. Ortega, J. M., *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, 1988.
9. Saad, Y. and M. H. Schultz, 'Parallel Direct Methods for Solving Banded Linear Systems', *Linear Algebra and its Appl.* 88/89, pp. 623-650, Apr. 1987.
10. Chishti, F. H., A. R. Clare and M. Razaz 'Gaussian Elimination of Symmetric, Positive Definite, Banded Systems on Transputer Networks,' in Transputer/Occam Japan 4 (Ed. S. Noguchi and H. Umeo), pp. 73-84, *Proceedings of the 4th Transputer/Occam Int. Conf.*, Tokyo, Japan, IOS Press 1992.
11. Chishti, F. H., A. R. Clare and M. Razaz 'Parallel Solution of Symmetric Banded Systems on Transputers', Vol. 3, pp. 1949-1952, *Proceedings of the 26th IEEE International Symposium on Circuits and Systems (ISCAS'93)*, Chicago, USA, 1993, IEEE, 1993.

12. Chishti, F. H., A. R. Clare and M. Razaz 'Transputer Implementation of Parallel Generation and Assembly for Finite Element Systems,' submitted to World Transputer Congress '93 (WTC'93), Sept. 1993, Aachen, Germany.
13. Chishti, F. H. and M. Razaz, 'Parallel Solution of FE-based Heat Conduction Problems on Transputer Networks,' accepted *Proc. 8th Int. Conf. Numerical Methods for Thermal Problems*, July 1993, Swansea, UK.
14. Meiko Scientific Ltd., *Meiko Hardware Reference Guide*, Meiko Scientific Ltd., Bristol, 1992.
15. Inmos Ltd., *Transputer Reference Manual*, Prentice-Hall, 1988.
16. Hoare, C. A. R., 'Communicating Sequential Processes', *Comm. ACM*, 21/8 pp. 65-93, 1978.
17. Clare, A. R., *Introductory User Guide for the Meiko Computing Surface at UEA*, Computing Centre, University of East Anglia, 1991.
18. Myers G. E., *Analytical Methods in Conduction Heat Transfer*, Genium Pub., Schenectady, New York, 1971.