A review of parallel finite element methods on the DAP

C. H. Lai and H. M. Liddell

Department of Computer Science and Statistics, Queen Mary College, University of London, London E1 4NS, UK

(Received March 1986; revised December 1986)

This paper reviews the research work that has been done to implement the finite element method for solving partial differential equations on the ICL distributed array processor (DAP). A brief outline of the principle features of the method is given, followed by details of the novel techniques required for implementation on the highly parallel architecture. Various methods of solution of the finite element equations are discussed; both direct and iterative techniques are included. The current state-of-the-art favours the use of the preconditioned conjugate gradient method. Some suggestions for future research work on parallel finite element methods are made.

Keywords: DAP (distributed array processor), SIMD (single-instruction multiple-data), FE (finite element), MIMD (multiple-instruction multiple-data), CG (conjugate gradient)

Introduction

A number of research workers have considered the application of finite element (FE) methods on the DAP. Wait, Delves, and their groups at the University of Liverpool have carried out research into the implemention of both FE techniques and the global element method (GEM) on the DAP.[1–3] Most of the work was directed towards the solution of two-dimensional (2D) elliptic problems, but three-dimensional (3D) problems have also been studied. Dixon[4,5] and his group at Hatfield Polytechnic investigated the solution of partial differential equations (p.d.e.s) arising from the two-dimensional Poisson equation following a Galerkin FE approach, which employed a parallel version of the conjugate gradient (CG) algorithm. A novel feature of their work is that the assembly of the global stiffness matrix is not required. Livesley, Modi, and Smithers[6] have also analysed storage schemes and solution methods for parallel FE computations. At Queen Mary College, Davies, Lai, Liddell, and Parkinson have been basing their work on the SERC/NAG FE library.[7,8] The latter was used as a model for the development of DAP FE software with the initial aim of producing sufficient routines to solve simple 2D and 3D p.d.e.s on meshes of modest size by simple triangular, quadrilateral, or brick elements. The SERC routines required to solve such problems are general enough to solve many other types of problems and the DAP approach has been to provide equivalent routines but designed specifically for the highly parallel DAP hardware, which maintain this generality. As a result of this work, DAP routines have been produced that provide a framework for solving FE problems on the DAP, which have been made available to the DAP user community. This approach was adopted in order to maximise progress in the short term, but will probably not yield an optimal DAP FE package, since the structure of existing packages, written for serial machines, is not optimal for DAP-oriented algorithms. The overall structure required for an efficient DAP mapping is one of the points under consideration. However, there are other advantages in basing the work on the FE library. Unlike many commercial FE packages, the library has been designed to be highly flexible, particularly for use in the development field. The documentation style is based on that used in the NAG library, as is the DAP subroutine library, so it is easy to incorporate appropriate routines from the latter.

To date, most FE research on the DAP has concentrated on the processing stage of the calculations. The mainframe DAP is a 64×64 array of simple 1-bit processors (*Figure 1*), each connected vertically to its own

0307-904X/87/050330-11/$03.00

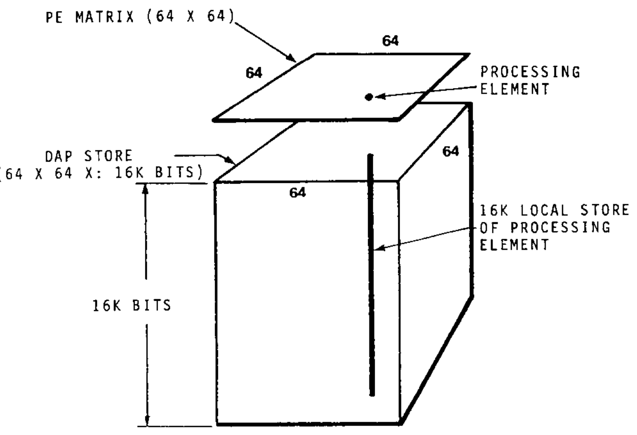

PE MATRIX (64 X 64)
64
64
PROCESSING ELEMENT
DAP STORE (64 X 64 X: 16K BITS)
64
64
16K LOCAL STORE OF PROCESSING ELEMENT
16K BITS

Figure 1 The mainframe DAP

16K bits of store and horizontally to its four nearest neighbours on a rectangular grid. There are also row and column highways, which permit fast access to elements in the same row or column. The DAP is a SIMD (single-instruction multiple-data) machine—the master control unit issues the same set of instructions to all processors—but at any instant some processors may be switched off, thus providing a degree of local autonomy. The DAP array can also be treated as a one-dimensional "long vector" of 4096 processors. The DAP is used via a host processor, the ICL 2980, which has limited interactive capabilities and is not suited to image processing and graphics applications. A more interactive environment is provided by the 1024 processor mini-DAP attached to a scientific workstation, which has good graphics and windowing facilities; the latter allows the viewing of a number of concurrent processes. This type of system encourages the development of parallel mesh generation and graphics algorithms to be incorporated into the pre- and postprocessing sections of the code. Some of the results produced in this paper were obtained on the mainframe DAP, others on the mini-DAP. The individual processor speed of the latter is slightly greater than the former, but the larger array and store size of the mainframe DAP is advantageous for large FE problems. However, future mini-DAPs are expected to have much more store than is available on the prototypes currently used.

There is also intensive research at NASA Langley on the application of the FE machine to the FE method. This machine is a MIMD (multiple-instruction multiple-data) computer. More details of the hardware can be found in Ref. 9. In this review the main emphasis is on current developments in the use of a SIMD computer (the DAP) for the FE method. However, some parallel algorithms that have been developed by the group at NASA Langley for use on their MIMD FE machine are included, since they appear suitable also for SIMD architectures.

The finite element method

The FE method is a numerical technique for obtaining approximate solutions of boundary value problems. These boundary value problems usually arise from the modelling of physical behaviour and can be divided into three groups:

1. Equilibrium problems involving stationary phenomena (i.e., phenomena that are independent of time and that can be reduced to equations of elliptic type (e.g., Laplace's equation))
2. Eigenvalue problems, which involve eigenvalue problems for elliptic operators (e.g., the Helmholtz equation)
3. Evolution problems, whose solutions depend on time and are characterised by parabolic or hyperbolic equations (e.g., the diffusion equation, the wave equation)

In the FE technique the domain of interest is divided into elements of finite dimension. These elements are referred to as 'finite elements'. Each element consists of some nodal points or nodes on its boundary. The unknown field variables Φ of an element can be approximated in terms of the nodal variables ϕ and the shape functions N of the element, namely

$$\Phi = N^{\mathrm{T}}\phi \tag{1}$$

This equation is the FE discretisation. For a standard element of square shape as given in *Figure 2* with four nodal points and nodal unknowns, ϕ_1, ϕ_2, ϕ_3, ϕ_4, the bilinear shape functions are given by

$$N_i = \tfrac{1}{4}(1 + \xi\xi_i)(1 + \eta\eta_i) \qquad i = 1, 2, 3, 4 \tag{2}$$

where ξ_i and η_i are local coordinates of node i of the standard element in *Figure 2* and ξ, η are local coordinates of an arbitrary point inside the standard element. The subscripts of the local unknowns are called the local freedom numbers. The number of unknowns per node is often referred to as 'degrees of freedom'. Here, each node has one degree of freedom, and each square element has four degrees of freedom.

Consider a rectangular computational domain divided into nine square elements, as shown in *Figure 3*. At this stage it is necessary to define what is meant by the nodal geometry, steering vector, element topology, and element geometry. The nodal geometry describes the coordinates of each node; e.g., node 6 in *Figure 3* has nodal geometry (x_6, y_6) or $(\frac{1}{3}, \frac{2}{3})$ if the domain is a unit square; in general, for any node i the nodal geometry is (x_i, y_i). The steering vector is the global nodal numbering of an element in the direction of the local nodal numbering of the standard element; e.g., the steering vectors for elements 3 and 5 of *Figure 3* are [7,

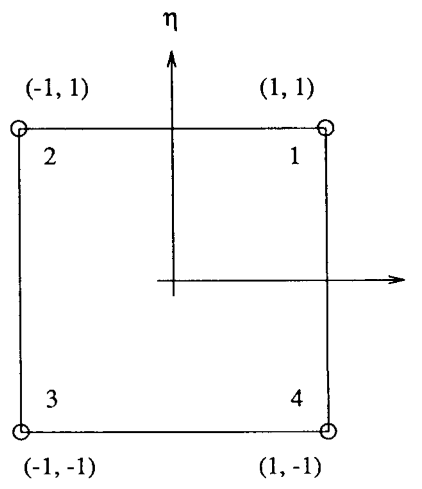


Figure 2 Local coordinates and node numbering of a standard square element

3, 4, 8] and [10, 6, 7, 11]; for any general quadrilateral as shown in *Figure 4*, the steering vector is $[i, j, k, l]$. The element topology specifies the topological shape and type of an element using n_{nod}, n_t, and the steering vector of a given element; e.g., element topology of element 5 is [4, 1, 10, 6, 7, 11], where $n_{\text{nod}} = 4$ is used to denote the number of nodes per element and $n_t = 1$ is used to denote a square element. Different values of n_t should be used for different types of elements; for a general element as shown in *Figure 4*, the element topology is $[n_{\text{nod}}, n_t, i, j, k, l]$. The element geometry specifies the geometrical location of an element; e.g., the element geometry of element 5 of *Figure 3*, assuming a unit square domain, is

$$\begin{bmatrix} x_{10} & y_{10} \\ x_6 & y_6 \\ x_7 & y_7 \\ x_{11} & y_{11} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} \\ \frac{1}{3} & \frac{2}{3} \\ \frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

Consider the 2D potential flow equation

$$K_x \frac{\partial^2 \Phi}{\partial x^2} + K_y \frac{\partial^2 \Phi}{\partial y^2} = -Q \tag{3}$$

where K_x and K_y are permeabilities in the x- and y-directions, and Q is the local source or sink.

Several mathematical formulations[10,11] can use the FE discretisation: e.g., Galerkin's method, the Rayleigh–Ritz method (applicable only to self-adjoint systems), and the least-squares method.

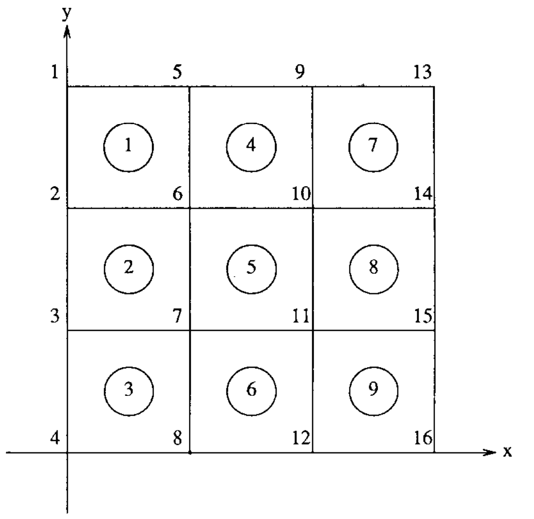


Figure 3 A computational domain with nine square elements

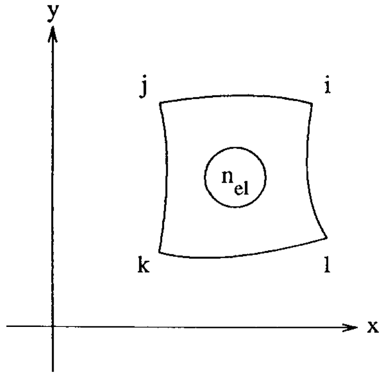


Figure 4 Global node numbering of a quadrilateral element

Galerkin's method involves the orthogonalisation process of the residual and an arbitrary change $\delta\Phi$ of the function Φ. For example, the potential flow equation has residual

$$\epsilon = K_x \frac{\partial^2 \Phi}{\partial x^2} + K_y \frac{\partial^2 \Phi}{\partial y^2} + Q \tag{4}$$

and, according to the definition, the orthogonalisation is thus

$$\langle \epsilon, \delta\Phi \rangle = 0 \tag{5}$$

i.e.,

$$\iint \left(K_x \frac{\partial^2 \Phi}{\partial x^2} + K_y \frac{\partial^2 \Phi}{\partial y^2} + Q \right) \delta\Phi \, dx \, dy = 0 \tag{6a}$$

Substituting the FE discretisation as given by equation (1) into equation (6a) gives

$$\sum_{n_{el}} \iint_{el} \left(K_x \sum_i \frac{\partial^2 N_i}{\partial x^2} \phi_i + K_y \sum_i \frac{\partial^2 N_i}{\partial y^2} \phi_i + Q \right) \sum_j N_j \, \delta\phi_j \, dx \, dy \tag{6b}$$

which leads to an element equation given by

$$\sum_j \delta\phi_j \iint_{el} \sum_i \left(K_x N_j \frac{\partial^2 N_i}{\partial x^2} + K_y N_j \frac{\partial^2 N_i}{\partial y^2} \right) \phi_i \, dx \, dy + \sum_j \delta\phi_j \iint_{el} Q N_j \, dx \, dy = 0 \tag{6c}$$

The element stiffness matrix k is thus

$$k_{ij} = \iint_{el} \left(K_x N_j \frac{\partial^2 N_i}{\partial x^2} + K_y N_j \frac{\partial^2 N_i}{\partial y^2} \right) dx \, dy = -\iint_{el} \left(K_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + K_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} \right) dx \, dy \tag{7a}$$

and the element source vector Q^{el} is

$$Q_i^{el} = -\iint_{el} Q_i N_i \, dx \, dy \tag{7b}$$

In the Rayleigh–Ritz method a functional F, called the energy functional, is defined such that the minimum is the solution of the given differential equation. For the 2D potential flow equation, we can define

$$F = \iint \left(K_x \left(\frac{\partial \Phi}{\partial x} \right)^2 + K_y \left(\frac{\partial \Phi}{\partial y} \right)^2 - 2Q\Phi \right) dx \, dy \tag{8}$$

Substituting the FE discretisation into the functional F leads to

$$F = \sum_{n_{el}} F^{n_{el}} = \sum_{n_{el}} \iint_{el} \left[K_x \left(\sum_i \frac{\partial N_i}{\partial x} \phi_i \right)^2 + K_y \left(\sum_i \frac{\partial N_i}{\partial y} \phi_i \right)^2 - 2Q \sum_i N_i \phi_i \right] dx \, dy \tag{9a}$$

Differentiating with respect to ϕ_i and setting to zero give

$$\nabla F_i^{n^{el}} = \iint_{el} \left[2K_x \frac{\partial N_i}{\partial x} \left(\sum_j \frac{\partial N_j}{\partial x} \phi_j \right) + 2K_y \frac{\partial N_i}{\partial y} \left(\sum_j \frac{\partial N_j}{\partial y} \phi_j \right) - 2Q_i N_i \right] \mathrm{d}x\, \mathrm{d}y = 0 \quad (9b)$$

Differentiating equation (9b) with respect to ϕ_i leads to the Hessian matrix $\nabla^2 F^{n^{el}}$ such that

$$\nabla^2 F_{ij}^{n^{el}} = \iint_{el} \left(2K_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + 2K_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} \right) \mathrm{d}x\, \mathrm{d}y \quad (9c)$$

which must be positive definite to ensure an occurrence of a minimum for F. The element stiffness matrix k is

$$k_{ij} = \iint_{el} \left(2K_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + 2K_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} \right) \mathrm{d}x\, \mathrm{d}y \quad (10a)$$

and the element source vector Q^{el} is

$$Q_i^{el} = \iint_{el} 2Q_i N_i \, \mathrm{d}x\, \mathrm{d}y \quad (10b)$$

The least-squares method also involves a definition for a functional F, which is given by the inner product of the residual; i.e.,

$$F = \langle \epsilon, \epsilon \rangle \quad (11)$$

A similar minimisation process to the one previously given can be carried out.

The element stiffness matrix usually involves integration such as

$$\iint f(x, y)\, \mathrm{d}x\, \mathrm{d}y = \int_0^1 \int_0^1 f(x, y) |\det(J)| \, \mathrm{d}\xi\, \mathrm{d}\eta \quad (12)$$

Here the Jacobian matrix J is

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \quad (13)$$

Usually equation (12) is numerically integrated, and Gaussian quadrature[10] is used, which reduces an integration to a summation, as shown below:

$$\iint f(x, y)\, \mathrm{d}x\, \mathrm{d}y = \sum_{i=1}^{n} w_i |J(\xi_i, \eta_i)| \, f(x_i, y_i) \quad (14)$$

where w_i are called the weight coefficients and n is the number of quadrature points. Details of the abscissae and weight coefficients can be found in Ref. 10.

Having obtained the element stiffness matrix k and the element source vector Q^{el}, we must construct the global matrix and the global source vector. This process essentially requires the proper positioning of each local node of an element in the global system. Consider the discretised physical domain with nine finite elements as shown in *Figure 3*; the steering vector for element 3 is [7, 3, 4, 8]. The element stiffness matrix for element 3 is $(k_{ij})_3$; therefore the stiffness contribution from element 3 to the global system is simply the assignment

$$\begin{array}{llll} (k_{11})_3 \to K_{77} & (k_{12})_3 \to K_{73} & (k_{13})_3 \to K_{74} & (k_{14})_3 \to K_{78} \\ (k_{21})_3 \to K_{37} & (k_{22})_3 \to K_{33} & (k_{23})_3 \to K_{34} & (k_{24})_3 \to K_{38} \\ (k_{31})_3 \to K_{47} & (k_{32})_3 \to K_{43} & (k_{33})_3 \to K_{44} & (k_{34})_3 \to K_{48} \\ (k_{41})_3 \to K_{87} & (k_{42})_3 \to K_{83} & (k_{43})_3 \to K_{84} & (k_{44})_3 \to K_{88} \end{array}$$

Hence the global system is

$$K\phi = b \quad (15)$$

where K is the global stiffness matrix, which is usually a symmetric positive definite matrix, ϕ is the global unknown vector, and b is the global source vector.

In this mathematical formulation the boundary conditions are of Dirichlet type. For natural boundary conditions the integration previously given should use Green's formula. For example, in Galerkin's formulation the integral given by equation (6a) should yield

$$\iint \left(K_x \frac{\partial^2 \Phi}{\partial x^2} + K_y \frac{\partial^2 \Phi}{\partial y^2} + Q \right) \delta\Phi \, \mathrm{d}x\, \mathrm{d}y$$
$$= -\iint \left(K_x \frac{\partial \Phi}{\partial x} \frac{\partial \delta\Phi}{\partial x} + K_y \frac{\partial \Phi}{\partial y} \frac{\partial\, \delta\Phi}{\partial y} \right) \mathrm{d}x\, \mathrm{d}y + \int_S q_n \, \delta\Phi \, \mathrm{d}S + \iint Q \, \delta\Phi \, \mathrm{d}x\, \mathrm{d}y$$

rather than equation (6b). Here q_n is the normal derivative along the boundary S. The line integral along the boundary can be carried out by numerical integration using the Gaussian quadrature formula.

The problem is now reduced to the solution of the set of simultaneous equations

$$Ax = b \quad (16)$$

where A is a symmetric positive definite matrix. The fourth section describes various parallel solution methods for a large sparse system of equations.

A typical FE program[8] for the solution of a field problem includes

1. Nodal geometry and element data that can be produced by a mesh generation routine or a large amount of input data
2. Element stiffness matrix and source vector calculations
3. Global stiffness matrix and source vector assembly
4. Introduction of boundary conditions
5. Solution of FE equations

Parallel implementation of finite element on the DAP

Element topology and geometry

Researchers have developed two methods for mapping finite elements onto the DAP.

S. Davies[7] and R. Wait[2,3] have used a 'long-vector' method, where elements are consecutively stored in a long vector according to the element numbering order. *Figure 5(a)* illustrates this element storage scheme on a 4 × 4 DAP for the computational domain in *Figure 3*. With this type of storage the x-coordinates of the nodes are kept in a long vector with declaration $X(,)$, and similarly for y-coordinates. The element topology is

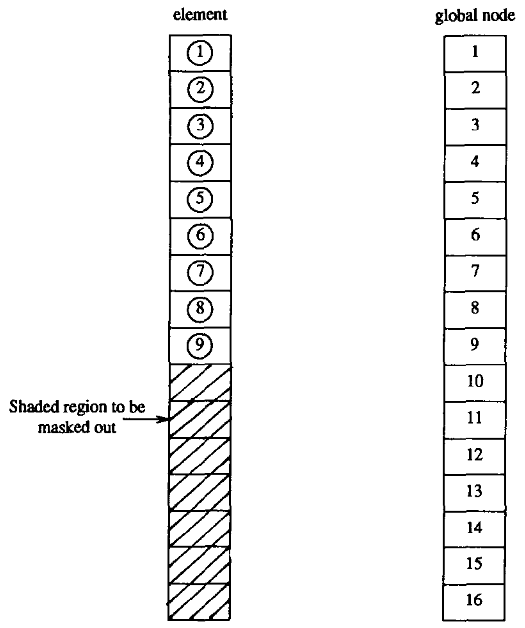


Figure 5(a) Long-vector storage of finite elements. $i = 1, \ldots, 16$ refers to the global nodal numbering, and a circled number refers to the element numbering (see *Figure 3*)

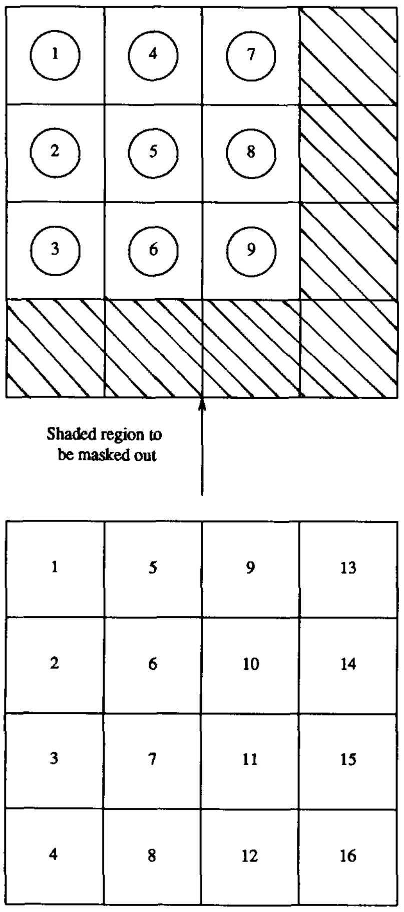
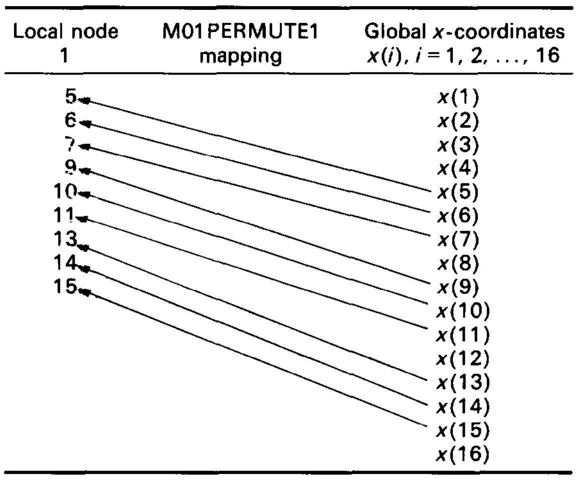


Figure 5(b) Upper leftmost storage of finite elements. $i = 1, 2, \ldots, 16$ refers to the global nodal numbering, and a circled number refers to the element numbering (see *Figure 3*)

kept in six long vectors with the matrix declaration ELTOP(,, 6). It is because the elements are stored according to the element numbering order that a random allocation[2] of these elements to the long vector is possible. Such an element ordering usually arises from irregular boundaries and grids.

Ducksbury[5] of Hatfield Polytechnic has used an 'upper leftmost' element storage scheme on the DAP. This storage method is also suggested by Livesley, Modi, and Smithers[6] of Cambridge University. *Figure 5(b)* shows the upper leftmost storage on a 4 × 4 DAP for the example as given in *Figure 3*. With this type of storage the x- and y-coordinates of the nodes are kept in the processors at the corresponding nodes.

In order to generate the element geometry for the *long-vector storage* (*Figure 5(a)*), we define a mapping that selects the required node number for a particular element and the corresponding coordinates. This mapping can be done via the DAP library utility routine M01PERMUTE1. The following sketch illustrates the mapping of global coordinates of all the nodes (see *Figure 3*) referred to the first node (upper right node) of the standard element (see *Figure 2*). In this example, since nodes 1, 2, 3, 4, 8, 12, and 16 are not the upper right node of any element, they are not involved in the mapping. A similar mapping must be done for the other nodes (i.e., nodes 2, 3, and 4 of the standard element) for both x- and y-coordinates.



To generate the element geometry for the *upper leftmost storage*, we can use shift functions. For example, coordinates for local node 1 are obtained as

$$\text{GEOM}(,,1, 1) = \text{SHWP(X)} \qquad \text{GEOM}(,,1, 2) = \text{SHWP(Y)}$$

Coordinates for local node 2 are

$$\text{GEOM}(,,2, 1) = \text{X} \qquad \text{GEOM}(,,2, 2) = \text{Y}$$

and so on, where the declaration GEOM(,,4, 2) is used to hold the element geometry of each element with four nodal points and two spatial coordinates per point.

One of the advantages of the long-vector storage of elements is that it can handle arbitrary element shapes. An immediate disadvantage of upper leftmost storage is that the local numbering system has to follow the one shown in *Figure 2* if the above shift functions are adopted. Moreover, the long-vector storage, which permits random allocation of elements to processors, can more easily incorporate an automatic mesh generation where there is normally a random numbering of the generated elements.

Element stiffness matrix

Instead of calculating the element stiffness matrices one by one as in a serial code, we can form them simultaneously; i.e., a serial algorithm can be used vertically. For example, the local derivates, the Jacobian matrices, and the global derivatives can be obtained vertically by a serial code. However, the code has to be accompanied by the appropriate mask for long-vector or upper leftmost element storage. To perform the numerical integration that produces the element stiffness matrix, we can adopt a numerical integration scheme, such as the one in equation (14).

In the element matrix calculation, irrespective of whether long-vector or upper leftmost element storage is used, the code is a simple extension of the serial code so that parallel operations can be carried out.

Global matrix assembly

This section deals with an important part of the parallel implementation of finite elements on the DAP. Since the ordering of the unknowns in the global matrix K (see equation (15)) is nodal, the solution must be based on a processor allocation that is node based rather than element based. Thus a FE computation on the DAP has a reassignment and global assembly phase that has no direct counterpart in a serial algorithm. If adjacent elements are assembled on different processors, it is necessary for data to be transmitted from one processor to another in order to assemble the global system. Again there are two types of assembly techniques, depending on the method of FE storage.

The Hatfield group[4] has developed a parallel CG method such that the assembly of the global stiffness matrix is not required. This method is described in the next section in more detail. The Cambridge group[6] has suggested an assembly technique based on the upper leftmost storage, which is used to obtain the global stiffness matrix in band matrix form as follows.

Consider a 4×4 DAP and a physical domain divided into nine elements with local numbering system as given in *Figure 3*, with the normal definition for element stiffness matrix $(k_{ij})_{n_{el}}$, where the subscript n_{el} represents the element number; if $k(,,i, j)$ is the corresponding declaration in a DAPFORTRAN program, then some of the matrix coefficients are

$$k(,,1,1) = \begin{bmatrix} (k_{11})_1 & (k_{11})_4 & (k_{11})_7 & 0 \\ (k_{11})_2 & (k_{11})_5 & (k_{11})_8 & 0 \\ (k_{11})_3 & (k_{11})_6 & (k_{11})_9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$k(,,2,2) = \begin{bmatrix} (k_{22})_1 & (k_{22})_4 & (k_{22})_7 & 0 \\ (k_{22})_2 & (k_{22})_5 & (k_{22})_8 & 0 \\ (k_{22})_3 & (k_{22})_6 & (k_{22})_9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where $k(,,4, 4)$ is a matrix array declaration on a 4×4 DAP. To calculate $K_{6,6}$ in the global matrix, we need the sum of $(k_{44})_1$, $(k_{33})_4$, $(k_{11})_2$, $(k_{22})_5$. Therefore the diagonal of the global matrix can be written in DAPFORTRAN as

$$k(,,2,2) + \mathrm{SHSP}(k(,,3,3)) + \mathrm{SHEP}(k(,,(4,4))) + \mathrm{SHSP}(\mathrm{SHEP}(k(,,4,4)))$$

The first off-diagonal of the band matrix is SHEP(k(„1, 4)) + k(„2, 3), the second is zero, the third is SHSP(k(„3, 1)), the fourth is SHSP(k(„3, 4)) + k(„2, 1), and the fifth is k(„2, 4).

This process may be illustrated by a diagrammatic approach, as in *Figure 6*, which involves five different stages for quadrilateral elements during the assembly phase. The tail and head of each arrow in *Figure 6* represent the subscripts i and j in the stiffness matrix k_{ij}, where ij are local node numberings (*Figure 2*). For example, in stage 4 the subscripts ij are 21 and 34; hence k_{21} and k_{34} of an appropriate element are added up. This type of global assembly has the immediate disadvantage that a specified format in the numbering of nodes and elements has to be followed. For example, the DAPFORTRAN code given previously for the global matrix assembly is valid only for the numbering system in *Figure 2*. Furthermore, it is very difficult to generalise to complicated elements.

A different approach is to use long-vector storage, which was done by Davies[7] and Wait and Martindale.[2] *Figure 7* shows the element matrices on the DAP, where each rectangular cube represents a long vector. Following the global matrix assembly given in the second section, it is very useful to have the steering vector on hand, i.e., the global node numbering of each element in the direction of the local node numbering of a standard element. For *Figure 3* the steering vectors are

Element number	Steering vectors			
1	5	1	2	6
2	6	2	3	7
3	7	3	4	8
4	9	5	6	10
5	10	6	7	11
6	11	7	8	12
7	13	9	10	14
8	14	10	11	15
9	15	11	12	16

These steering vectors define a mapping for the element stiffness matrices' contribution to the global system. For example, the diagonals of the global matrix come from k_{11}, k_{22}, k_{33}, and k_{44}, and the total stiffness contribution to the diagonal of the global matrix is

Freedom number	Stiffness contribution			
1		$(k_{22})_1$		
2		$(k_{22})_2$	$(k_{33})_1$	
3		$(k_{22})_3$	$(k_{33})_2$	
4			$(k_{33})_3$	
5	$(k_{11})_1$	$(k_{22})_4$		
6	$(k_{11})_2$	$(k_{22})_5$	$(k_{33})_4$	$(k_{44})_1$
7	$(k_{11})_3$	$(k_{22})_6$	$(k_{33})_5$	$(k_{44})_2$
8			$(k_{33})_6$	$(k_{44})_3$
9	$(k_{11})_4$	$(k_{22})_7$		
10	$(k_{11})_5$	$(k_{22})_8$	$(k_{33})_7$	$(k_{44})_4$
11	$(k_{11})_6$	$(k_{22})_9$	$(k_{33})_8$	$(k_{44})_5$
12			$(k_{33})_9$	$(k_{44})_6$
13	$(k_{11})_7$			
14	$(k_{11})_8$			$(k_{44})_7$
15	$(k_{11})_9$			$(k_{44})_8$
16				$(k_{44})_9$

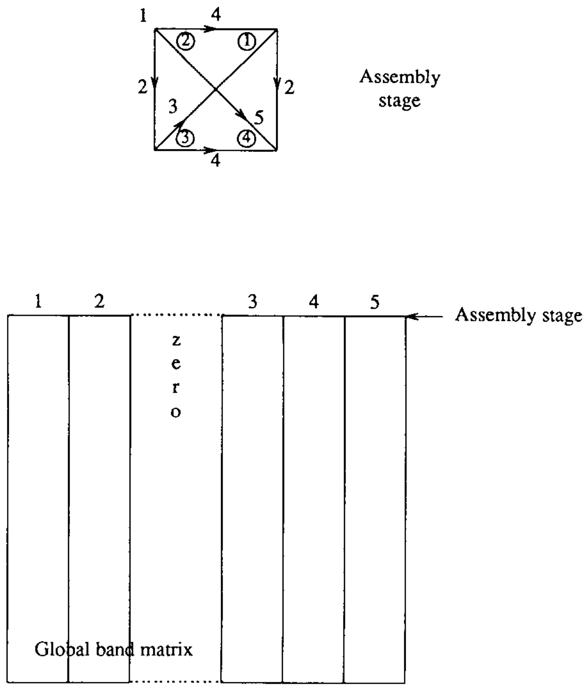


Figure 6 Different stages in parallel global matrix assembly for rectangular elements

Hence with the mapping defined by the steering vector, one can easily assign the k's to the corresponding positions. This kind of global assembly has the immediate advantage of being able to handle more complex elements without changing the code very much. Also, it can accommodate different types of element numbering systems as well as irregular boundaries.

Solution of finite element equations

The most important part of the FE calculation is the parallel solution of a set of FE equations. In this section a description is given of direct methods and iterative methods for solving a large sparse matrix system.

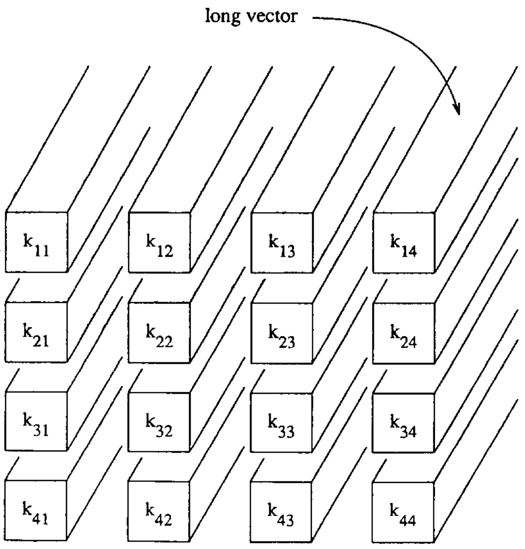


Figure 7 Long-vector storage element matrices for an element with four degrees of freedom

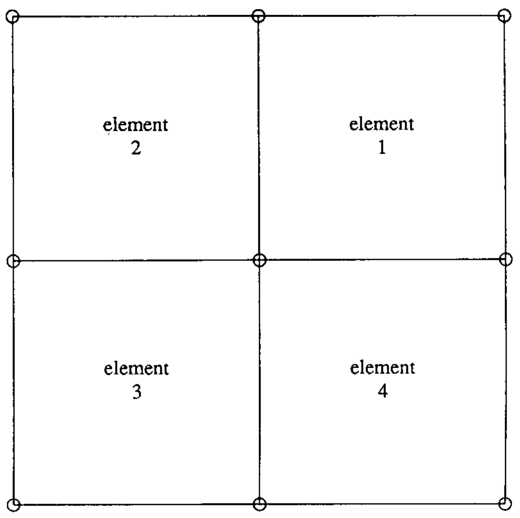


Figure 8 Neighbouring elements location of global node i using the upper leftmost storage

Direct methods

Direct solution methods such as Gaussian elimination, Cholesky factorisation, and LU decomposition (on a serial computer) can be used. However, there has been little development on parallel direct solvers on the DAP, apart from the hybrid Gauss procedure,[12] which was not specifically designed for banded systems, and some tridiagonal system solvers.[13]

The direct tridiagonal solvers on the DAP are based on a cyclic reduction algorithm.[13,14] The algorithm involves the reduction of three adjacent equations to one equation so that a new set of tridiagonal equations is formed with different coefficients. The number of equations at each level of reduction will be one-half that of the previous level. If the number of unknowns is $n = m - 1$, where $m = 2^q$ is an integer, and if the reduction procedure is carried out recursively, then after $\log_2 m - 1$ levels of reduction, only the central equation, $i = m/2$, remains, which involves only one unknown and can be solved. The cyclic reduction procedure for a tridiagonal system therefore involves the recursive calculation of new coefficients and right sides; for level $l = 1, 2, \ldots, q - 1$, one uses

$$a_i^{(l)} = \alpha_i a_{i-2^{(l-1)}}^{(l-1)} \qquad c_i^{(l)} = \gamma_i c_{i+2^{(l-1)}}^{(l-1)}$$
$$b_i^{(l)} = b_i^{(l-1)} + \alpha_i c_{i-2^{(l-1)}}^{(l-1)} + \gamma_i a_{i+2^{(l-1)}}^{(l-1)}$$
$$k_i^{(l)} = k_i^{(l-1)} + \alpha_i k_{i-2^{(l-1)}}^{(l-1)} + \gamma_i k_{i+2^{(i-1)}}^{(l-1)}$$

where

$$\alpha = -a_i^{(l-1)}/b_{i-2^{(i-1)}}^{(l-1)} \qquad \gamma_i = -c_i^{(l-1)}/b_{i+2^{(i-1)}}^{(l-1)}$$

and $i = 2^l$ to $m - 2^l$ with step 2^l.

Here a, b, c are vectors holding the tridiagonal coefficients, and k is the right-side vector. Initial values $a_i^{(0)}$, $b_i^{(0)}$, and $c_i^{(0)}$ are those in the original set of tridiagonal equations. The recursive filling in of the solutions for $l = q, q - 1, \ldots, 2, 1$ gives

$$x_i = (k_i^{(l-1)} - a_i^{(l-1)} x_{i-2^{(l-1)}} - c_i^{(l-1)} x_{i+2^{(l-1)}})/b_i^{(l-1)}$$

Other direct methods, such as Gaussian elimination and Cholesky factorisation are not suitable on the DAP, unless a hybrid procedure, such as the hybrid Gauss procedure mentioned previously is used. This hybrid Gauss procedure applies to dense systems only, and therefore it would be necessary to develop a hybrid Cholesky procedure on the DAP for sparse matrices. A hybrid procedure that uses an incomplete Cholesky factorisation as a preconditioner has been developed by Wait[3] and is discussed in the subsection entitled 'Preconditioned Conjugate Gradient Methods.'

Iterative methods: Conjugate gradient method

Iterative methods seem to be attractive to most researchers seeking a parallel solution of FE equations on the DAP. A similar trend appears in the parallel solution of FE equations on a MIMD computer, such as the FE machine of NASA. There are several reasons for this general trend. First, iterative methods have the advantage that minimal storage space is required for implementation, since no fill-in of the zero positions of the coefficient matrix for the system of linear equations occurs during computation. Second, an iterative process may converge in very few steps if a good initial guess is known. Third, it seems that iterative methods parallelise better than direct methods and are therefore potentially viable techniques for solving large sparse linear systems on parallel computers.

In the development of a parallel solution of a set of FE equations on the DAP, all the groups have concentrated on the application of CG methods with or without preconditioning.

The following algorithm describes a practical CG procedure for the solution of a linear system $Ax = b$, where A is a symmetric and positive definite $n \times n$ matrix. This method was proposed by Hestenes and Stiefel in 1952 as a method for solving a symmetric positive definite $n \times n$ system of linear equations.

Conjugate gradient algorithm. Given a system $Ax = b$, where A is a symmetric positive definite $n \times n$ matrix, and x_k, r_k, p_k, z_k, b are n-dimensional vectors

```
$x_0 := 0;\ r_0 := b;\ p_0 := r_0$ {initialise}
for $k := 1$ to $n$ do;
    $\alpha := \dfrac{\langle r_{k-1}, r_{k-1}\rangle}{\langle p_{k-1}, Ap_{k-1}\rangle}$
    $x_k := x_{k-1} + \alpha p_{k-1}$
    $r_k := r_{k-1} - \alpha A p_{k-1}$
    if $(\sqrt{\langle r_k, r_k\rangle}/\sqrt{\langle b, b\rangle} < e)$ stop
    $\beta := \dfrac{\langle r_k, r_k\rangle}{\langle r_{k-1}, r_{k-1}\rangle}$
    $p_k := p_k + \beta p_{k-1}$
```

We can slightly modify this algorithm so that the matrix A does not need to be assembled in a FE procedure. This approach was considered by Ducksbury,[5] who used the property that A is held as $\sum_{n_{el}} A^{n_{el}}$ and that all the values in a given $A^{n_{el}}$ will be zero except for those occurring in rows/columns corresponding to the variables in the n_{el} element. Therefore, as mentioned in the section on global matrix assembly, the global matrix equation can be solved without assembling the global matrix A. The method involves the summation of a certain quantity $f_{L(i)}^{n_{el}}$ from a local node $L(i)$ to a global node i from its neighbouring elements; i.e.,

$$F_i = \sum_{n_{el}} f_{L(i)}^{n_{el}} \tag{17}$$

where $L(i)$ maps a local node $L(i)$ to its corresponding global node i for a particular element n_{el}. Hence the formation of (p, Ap) in the CG algorithm, with subscripts $k - 1$ omitted for simplicity, is

1. $W_{L(i)}^{n_{el}} = \sum_{L(j)} A_{L(i),L(j)}^{n_{el}} p_{L(j)}^{n_{el}} \qquad w_i = \sum_{n_{el}} W_{L(i)}^{n_{el}}$
2. $(p, Ap) = \sum_{i=1}^{n} p_i w_i$

{n is the total number of degrees of freedom}

The CG implementation on the DAP by the Cambridge group has to be performed after the global matrix assembly. Although they have suggested that matrix assembly is not required, they have not given their method of implementation.[6] One of the reasons claimed for the speedup factors reported by Ducksbury[5] is that the calculations described in this section can be omitted in his approach, but our experience has shown that the nonassembly CG algorithm based on the method given above takes longer computationally because of the summation involved in step 1. However, a substantial reduction in storage requirement can be achieved for large problems where the bandwidth is also large.

Iterative methods: Preconditioned conjugate gradient methods

The convergence rate for the CG method described previously is slow, particularly for large problems. One way to improve this slow convergence is to precondition A. The method consists of finding a nonsingular symmetric matrix C such that $\tilde{A}\ (= C^{-1}AC^{-T})$ has an improved condition number. One can then apply the CG method to the transformed system $\tilde{A}\tilde{x} = \tilde{b}$ by replacing $\tilde{r}$, $\tilde{x}$, $\tilde{b}$, and $\tilde{A}$ in the CG algorithm, and the solution $\tilde{x}$ can then be transformed back to x by the relation $x = C^{-T}\tilde{x}$.

We must modify the iteration described above by setting $M = CC^T$, $p = C^{-T}\tilde{p}$, $x = C^{-T}\tilde{x}$, $z_k = M^{-1}\tilde{r}_k$, $r_k = C\tilde{r}_k$, which leads to the following preconditioned CG algorithm.

Preconditioned conjugate gradient algorithm. Given a system $Ax = b$, where A is a symmetric positive definite $n \times n$ matrix with matrix element $[A]_{ij} = a_{ij}$, and x_k, r_k, z_k, p_k, b are n-dimensional vectors.

```
$x_0 :=$ ; $r_0 := b$   {initialise}
solve   $Mz_0 = r_0$
$p_0 := z_0$
for $k := 1$ to $n$  do
    $\alpha := \dfrac{\langle z_{k-1}, r_{k-1}\rangle}{\langle p_{k-1}, Ap_{k-1}\rangle}$
    $x_k := x_{k-1} + \alpha p_{k-1}$
    $r_k := r_{k-1} - \alpha A p_{k-1}$
    if  $(\sqrt{\langle r_k, r_k\rangle}/\sqrt{\langle b, b\rangle} < e)$ stop
    solve $Mz_k = r_k$
    $\beta := \dfrac{\langle z_k, r_k\rangle}{\langle z_{k-1}, z_{k-1}\rangle}$
    $p_k := z_k + \beta p_{k-1}$
```

M is called the preconditioner. The remaining problem is to choose a suitable preconditioner M such that a solution x can be easily obtained. The best choice of M to produce a fast convergence rate depends on $c(M^{-1}A)$: the smaller $c(M^{-1}A)$, the faster convergence rate. Here $c(A)$ is called the condition number of a non-singular matrix A and is defined as

$$c(A) = \max_i |\lambda_i| \Big/ \min_i |\lambda_i| \tag{18}$$

where λ_i, $i = 1, \ldots, n$, are eigenvalues of A. Two criteria suggested by Golub and Van Loan[15] for M to be an effective preconditioner on a serial computer are

1. $Mz_{k-1} = r_{k-1}$ is easily solved.
2. If $K = M - R$, where R is regarded as a remainder term, then $M^{-1}R$ should have small or nearly equal eigenvalues.

Numerous preconditioning methods have been proposed for sequential computers; however, considerable difficulties arise when the implementation is in a parallel environment. It is necessary to add another criterion for M when it is to be implemented on a parallel computer. This criterion is that M should be easily formed on a parallel computer.

The choice of preconditions for the CG method on the DAP has been discussed by Ducksbury[5] and Wait[3]. Ducksbury's approach improves convergence by scaling the linear system of equations. Consider the original system $Ax = b$, scaled by matrices D_1 and D_2, such that

$$x = D_1\tilde{x} \qquad b = D_2\tilde{b} \tag{19}$$

which give rise to a new system $\tilde{A}\tilde{x} = \tilde{b}$ with

$$\tilde{A} = D_2^{-1}AD_1 \tag{20}$$

Ducksbury[5] chose $D_1^{\mathrm{T}} = D_2^{-1}$, where

$$[D_1]_{ij} = \begin{cases} a_{ij}^{-1/2} & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases}$$

Thus,

$$\tilde{a}_{ij} = a_{ij}/\sqrt{a_{ii}\,a_{wj}} \quad \tilde{x}_i = x_i\sqrt{a_{ii}} \quad \tilde{b}_i = b_i/\sqrt{a_{ii}} \tag{21}$$

Note that in Ducksbury's CG solution, A is not assembled. Hence in order to form the new system as given by equation (21), we need the expressions

$$a_{ii} = \sum_{n_{\mathrm{el}}} a^{n_{\mathrm{el}}}_{L(i),L(i)} \tag{22}$$

$$\tilde{a}^{n_{\mathrm{el}}}_{ij} = a^{n_{\mathrm{el}}}_{ij}/\sqrt{a_{G(i),G(i)}\,a_{G(j),G(j)}}$$

where $L(i)$ maps a gobal node i onto a local node for a particular element n_{el} and $G(i)$ is the inverse of L. The scaling for x and b remains the same. The transformed system can then be solved by the CG method. When the solution $\tilde{x}$ converges, it is then rescaled back to x by

$$x_i = \tilde{x}_i/\sqrt{a_{ii}} \tag{23}$$

which serves as the solution of the original system.

Wait's method is based on block preconditioning. Essentially, each processor of the DAP contains several degrees of freedom, so a simple block partition of matrix A is

$$A = \begin{bmatrix} D_1 & E_{12}^{\mathrm{T}} & E_{13}^{\mathrm{T}} \\ E_{12} & D_2 & \ldots \\ \vdots & \ldots & \ldots \end{bmatrix}$$

with unknowns

$$x = \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \underline{x}_3 \\ \vdots \end{bmatrix}$$

A simple preconditioner used by Wait and Martindale[2] is the diagonal of the matrix A, which is essentially an extension of Ducksbury's method described above, except the number of unknowns per processor is greater than 1.

A second approach adopted by Wait is based on the hybrid method of Li *et al.*[16] which consists of a block partitioning of the global matrix A such that the original system becomes

$$\begin{bmatrix} D_1 & E^{\mathrm{T}} \\ E & D_2 \end{bmatrix}\begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \end{bmatrix}$$

The partition of the unknowns into two vectors is called one-way dissection. If a Cholesky factorisation exists, then A can be written as

$$\begin{bmatrix} D_1 & E^{\mathrm{T}} \\ E & D_2 \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ W & L_2 \end{bmatrix}\begin{bmatrix} L_1^{\mathrm{T}} & W^{\mathrm{T}} \\ 0 & L_2^{\mathrm{T}} \end{bmatrix}$$

and the hybrid solution involves

1. $\underline{y}_1 = L_1^{-1}\underline{b}_1$ (forward substitution).
2. $\underline{g}_2 = \underline{b}_2 - W\underline{y}_1 = \underline{b}_2 - EL_1^{-\mathrm{T}}\underline{y}_1 = \underline{b}_2 - \underline{w}_1$.
3. Solve $L_2L_2^{\mathrm{T}}\underline{x}_2 = \underline{g}_2$ (use CG method).
4. $\underline{g}_1 = \underline{y}_1 - W^{\mathrm{T}}\underline{x}_2 = \underline{y}_1 - L_1^{-1}E^{\mathrm{T}}\underline{x}_2 = \underline{y}_1 - \underline{w}_2$.
5. $\underline{x}_1 = L_1^{-\mathrm{T}}\underline{g}_1$ (backward substitution).

Note that $L_2L_2^{\mathrm{T}}$ can be written as $D_2 - ED_1^{-1}E^{\mathrm{T}}$, and an incomplete Cholesky factorisation CG method can be applied if the preconditioner M is chosen as

$$M = D_2 = G_2G_2^{\mathrm{T}} \tag{24}$$

where G_2 is a lower-triangular matrix in the Cholesky factorisation of D_2. Hence the preconditioned CG algorithm can be applied to solve step 3, with the preconditioner given by equation (24). The implementation on the DAP is a block to processor mapping, and if L_1 consists of $L_1^{(i)}$ $(i = 1, 2, \ldots)$, then each processor will

Table 1 Two-dimensional plane-strain problem

Number of iterations using CG algorithm		
Degrees of freedom	Without scaling	With scaling
740	434	359
174	147	105

Table 2 Two-dimensional Laplace equation

Number of iterations using block preconditioning method			
	Freedom per processor		
Degrees of freedom	1	4	9
900	27	22	20
3600	54	45	40
8100		66	60
14400		88	80
22500			100

look after each subblock (i) and the hybrid solution steps can be followed. Note that we need not evaluate W and L_2 explicitly, since they can be found in terms of E and L_1.

Some results using these preconditioning methods are given below. *Table 1* shows the number of iterations for a 2D solid deformed under plane strain with prescribed external loads using the CG algorithm on the mini-DAP. *Table 2* shows the number of iterations for a 2D Laplace equation with Dirichlet boundary conditions, using a block preconditioning method, performed on the mainframe DAP. In both cases the time for one CG iteration for one degree of freedom per processor is approximately 10 ms.

Another class of preconditioners that appears to be more suitable for implementation on a parallel computer is obtained by choosing M to be a splitting of A that describes a linear relaxation method: for example, the SOR (successive overrelaxation) method, where the iteration scheme for a system $Az = r$ is

$$\left(\frac{1}{\omega} D - L\right)z^{(m)} = \left(U + \frac{1-\omega}{\omega} D\right)z^{(m-1)} + r \qquad (25)$$

and the superscript (m) denotes the number of iterations using SOR. D is diag(A), L and U are strictly lower- and upper-triangular matrices, and ω is a relaxation parameter. If $z^{(0)}$ is chosen to be zero, then one step of the SOR method applied to $Az = r$ gives $z^{(1)}$, which is the exact solution of $Mz = r$, where

$$M = (1/\omega\, D - L) \qquad (26)$$

This method is called a one-step preconditioned CG method and has been applied on a MIMD computer by Adams.[17] Other methods of splitting a matrix A that involve a parallel relaxation method can be found in Refs. 9 and 14.

The class of preconditioners described above has been extended to an m-step preconditioned CG method by Adams.[17] The idea comes from the consideration of whether it would be beneficial to have more than one step of a relaxation method in order to produce a preconditioning matrix M that more closely approximates A. Some main deductions by Adams are given below.

Consider a splitting of A defined by $A = P - Q$ with iteration matrix $I_h = P^{-1}Q$. The m-step relaxation applied to $Az = r$ is

$$P(I + I_h + \cdots + I_h^{m-1})^{-1}z^{(m)} = P(I + I_h + \cdots I_h^{m-1})^{-1}I_h^m z^{(0)} + r \qquad (27)$$

If we choose $z^{(0)} = 0$, equation (26) reduces to

$$P(I + I_h + \cdots + I_h^{m-1})^{-1}z^{(m)} = r \qquad (28)$$

which gives the m-step preconditioning matrix

$$M = P(I + I_h + \cdots + I_h^{m-1})^{-1} \qquad (29)$$

Note that M must be symmetric and positive definite to be considered as a preconditioner for the CG method. To satisfy these criteria, the following conditions of splitting a matrix A ($= P - Q$) with iteration matrix $I_h = P^{-1}Q$ must be satisfied:

1. P is a symmetric nonsingular matrix.
2. If (1) is satisfied, then
   i for odd number of m, M is positive definite if and only if P is positive definite.
   ii for even number of m, M is positive definite if and only if $P + Q$ is positive definite.
3. If $P + Q$ is positive definite, then the asymptotic convergence rate $\rho(I_h) < 1$.

Note that the third condition must be imposed, otherwise the iteration scheme for $Az = r$ will diverge.

It is necessary to examine whether an m-step preconditioning is better than a one-step preconditioning; i.e., if M_m and M_1 are the m-step and one-step preconditioning matrices, then $c(M_m^{-1}A) < c(M_1^{-1}A)$ means that m-step provides faster convergence. A detailed examination of these condition numbers is given by Adams,[9] which includes cases for even and odd numbers of m. He has concluded that the m-step preconditioned CG method gives more improvement than the one-step preconditioned CG method if a suitable relaxation method is chosen.

The m-step preconditioning algorithm requires a slight modification to the previous preconditioning algorithm; i.e.,

```
replace
    solve $Mz_k = r_k$
by
    apply m steps of a relaxation to $Az_k = r_k$
    where splitting of A is $P - Q$; i.e.,
    $z_k^{(m)} := P^{-1}Qz_k^{(m-1)} + P^{-1}r_k$     with $z_k^{(0)} := 0$
    $z_k := z_k^{(m)}$
```

This method has been implemented on the DAP, and numerical results for the previous 2D plane-strain problem using the m-step point Jacobi relaxation CG method are presented in *Table 3*. Results for a 2D steady-potential-flow problem with Dirichlet boundary conditions and square elements using the m-step point Jacobi relaxation and the m-step four-colour Gauss–Seidel iteration method are presented in *Table* 4.

These results were obtained on a mini-DAP attached to a Perq scientific workstation. Notice that the number of iterations is a decreasing function of m. The time per iteration for the four-colour Gauss–Seidel is approximately four times that of the point Jacobi, so the latter is generally preferred on the DAP. The time for the point Jacobi iteration is approximately equivalent to the block preconditioning for one degree of freedom per processor, i.e., about 10 ms. For the 2D plane-strain problem even values of m do not converge, which means $P + Q$ is not positive definite (see *Table 3*). On the other hand, if $P^{-1}Q$ represents a point Jacobi relaxation method, one can always be sure that $P + Q$ is positive definite for a square element with four nodes discretisation of a 2D steady-potential-flow problem. Note that the number of degrees of freedom per processor is one in the above cases; the extension of the method for more than one degree of freedom per processor is currently being investigated.

Conclusions

The current status of the implementation of the FE method on the DAP has been presented. Two types of mapping for the problem have been adopted: long vector and upper leftmost storage. The authors find the former more flexible. There has been little development as yet in applying direct methods of solution to the sparse set of FE equations. Current development is based on an iterative method, namely, the preconditioned CG method. Successful preconditioners that have been implemented include diagonal scaling, block preconditioning using a nested dissection technique, and various relaxation (m-step) methods.

One approach to the problem of obtaining FE software in the new computer environment presented by the highly parallel architecture is to provide a toolkit of parts (algorithms and subroutines) that can be used by other workers in the field. This has been the aim of the DAP FE library development work. Much remains to be done to complete the set of FE tools; this will include the development of pre- and postprocessing graphics and mesh generation, error analysis techniques, investigation of methods for time-dependent and nonlinear problems, and the extension to complex arithmetic for some of the problems that occur in electromagnetic theory. For much of this work the computer environment provided by the mini-DAP + graphics workstation is highly desirable. While parallel techniques for the various aspects of the problem are being developed, it is useful to be able to interface the DAP routines with the serial routines provided by the SERC FE library. This approach should lead to the gradual evolution of a fully integrated parallel computer workstation environment for FE calculations.

Table 3 Two-dimensional plane-strain problem

Number of iterations using m-step point Jacobi preconditioner

	Degrees of freedom	
m	740	170
1	464	125
2	—	—
3	328	89
4	—	—
5	320	87
6	—	—
7	282	95

Table 4 Two-dimensional steady-potential-flow problem

Number of iterations using m-step preconditioning total degrees of freedom = 735

m	Point Jacobi	Four-colour Gauss–Seidel
1	44	97
2	25	24
3	22	19
4	18	15
5	16	13
6	14	12
7	13	10
8	12	10

Acknowledgements

This work was supported by the Science and Engineering Research Council under grant number GR/D/59427. The authors would like to thank Dr. R. Wait for his permission to reproduce his results in *Table 2*.

References

1 McKerrell, A. and Delves, L. M. 'Solution of the global element equations on the ICL DAP', *ICL Technical J.*, 1984, 4(1), 50–58
2 Wait, R. and Martindale, I. 'Finite elements on the DAP', in 'The mathematics of finite elements and applications', Vol. 5, J. R. Whiteman, ed., Academic Press, London, 1985, pp. 113–122
3 Wait, R. 'The solution of finite element equations on the DAP', presented at the *Int. Conf. on Vector and Parallel Processing, Leon, Norway*, June 1986
4 Dixon, L. C. W., Ducksbury, P. G., and Singh, P. 'A parallel version of the conjugate gradient algorithm for finite element problems', *Numerical Optimisation Centre, TR132*, 1982
5 Ducksbury, P. G. 'Parallel array processing', *Ellis Horwood Series in Electrical and Electronic Engineering*, 1986
6 Livesley, R. K., Modi, J. J., and Smithers, T. 'The use of parallel computation for finite element calculations', *Cambridge University, CUED/F-CAMS/TR248*, 1985
7 Davies, S. 'Notes on a DAP finite element library', private communication, *DAPSU, QMC*, 1985
8 Greenough, C., Emson, C. R. I., and Smith, I. M. 'The NAG/SERC finite element library—an applications software library for finite element analysis', *Rutherford Appleton Lab. Rept. RAL-84-107*
9 Adams, L. M. 'Iterative algorithms for large sparse linear systems on parallel computers', *NASA Contractor Report 166027*, 1982
10 Baker, A. J. 'Finite element computational fluid mechanics', McGraw-Hill, New York, 1976
11 Mitchell, A. R. and Wait, R. 'The finite element method in partial differential equations', Wiley, New York, 1977
12 Bowgen, G. S. J., Liddell, H. M., and Hunt, D. J. 'The solution of N linear equations on a P-processor parallel computer', *Technical Report, DAPSU, QMC*
13 Bowgen, G. S. J. and Whiteway, J. 'A parallel algorithm for solving tri-diagonal systems', *Technical Report, DAPSU*
14 Hockney, R. W. and Jesshope, C. R. 'Parallel computers', Adam Hilger, Bristol, 1981
15 Golub, G. H. and Van Loan, C. F. 'Matrix computation', North Oxford Academic, Oxford, 1983
16 Li, M. R., Nour-Omid, B., and Parlett, B. N. 'A fast solver free of fill-in for finite element problems', *SIAM J. Num. Anal.*, 1982, **19**, 1233–1242
17 Adams, L. M. 'm-Step preconditioned conjugate gradient methods', *NASA Contractor Report 172130*, 1983
18 Greenough, C. and Robinson, K. 'Examples in the use of the finite element library: steady state potential flow', Rutherford Appleton Lab Rept., *RAL, RL-82-060*, 1982
19 Lai, C. H. 'Application of DAP to computational aerodynamics', *Diss., Dept. of Aeronautical Engineering, Queen Mary College*, 1985