Computer Physics Communications 37 (1985) 187–193
North-Holland, Amsterdam

FINITE ELEMENT OPTIMISATION ON THE DAP

L.C.W. DIXON and P.G. DUCKSBURY

Numerical Optimisation Centre, The Hatfield Polytechnic, P.O. Box 9, College Lane, Hatfield, Herts AL10 9AB, UK

In this paper we report on our experience solving steady two dimensional sets of nonlinear simultaneous partial differential equations on the ICL-DAP computer. Results are reported for the heat conduction equation, a nonlinear pde in one variable and for the Navier Stokes equations. In each case the parallel implementation of finite element optimisation on the DAP solved coarse grid problems much more rapidly than the sequential code and also enabled much fine grids to be used.

1. Introduction

Finite Element Optimisation is an approach that enables general linear or nonlinear partial differential equations to be formulated and solved by an iteration process that is guaranteed to terminate in a finite number of iterations. If the system of partial differential equations is two-dimensional and steady then the method is readily implementable on the ICL-Distributed Array Processor. In this paper we will concentrate on such problems.

In finite element optimisation any set of simultaneous partial differentail equations are first converted into an equivalent optimisation problem. The solution of partial differential equations can be converted into an optimisation problem in two different ways. If the partial differential equations are self-adjoint, then they can be derived from the minimization of a functional subject to certain boundary conditions by using the calculus of variations. We will term this the variational approach and when solving such problems it is natural to revert to the underlying functional optimisation problem. For non-self-adjoint systems such a function does not exist, but it is possible to create a family of functional integrals whose minima correspond to the solution of the problem by taking weighted least square integrals of the equations.

In this paper we will first describe the basis of the finite element optimisation approach by showing how it would be applied to the Navier Stokes equations. We will then discuss how the method is implemented on the DAP by first considering a linear variational problem – namely the solution of the heat conduction equation. Then a simple nonlinear problem will be formulated by the least squares approach and solved using both the Fletcher–Reeves conjugate gradient method and the truncated Newton method on the distributed array processor and finally DAP solutions of the Navier–Stokes problem will be discussed.

2. The finite element / optimisation approach

Finite element optimisation has been developed as a general method for solving sets of nonlinear partial differential equations. If the equations are steady and two dimensional it is particularly easy to implement the method on the ICL-DAP so the method will be illustrated in this paper using the two dimensional steady Navier–Stokes problem.

Let Ox_1, Ox_2 be two orthogonal axes in the two-dimensional space and let u be the component of velocity parallel to Ox_1 and v the component parallel to Ox_2. Similarly let p be the pressure at any point in the flow and let R denote the Reynolds number. Then the flow of incompressible fluid is governed by the equations:

$$u\frac{\partial u}{\partial x_1} + v\frac{\partial u}{\partial x_2} = \frac{\partial p}{\partial x_1} + \frac{1}{R}\nabla u^2, \tag{1}$$

0010-4655/85/$03.30

$$u\frac{\partial v}{\partial x_1}+v\frac{\partial v}{\partial x_2}=-\frac{\partial p}{\partial x_2}+\frac{1}{R}\nabla v^2, \tag{2}$$

$$\text{and } \frac{\partial u}{\partial x_1}+\frac{\partial v}{\partial x_2}=0. \tag{3}$$

A flow problem is completed by the specification of an appropriate set of boundary conditions.

To convert this problem into a first-order optimisation problem of the type normally treated in the calculus of variations we introduce additional variables and use seven field variables

$$\begin{aligned} A&=u \quad B=\frac{\partial u}{\partial x_1} \quad C=\frac{\partial u}{\partial x_2}\\ D&=v \quad E=\frac{\partial v}{\partial x_1} \quad F=\frac{\partial v}{\partial x_2} \end{aligned} \tag{4}$$

and

$$P=p.$$

In terms of these variables the Navier–Stokes equations become seven first-order equations

$$\begin{aligned} e_1&=AB+CD-\frac{1}{R}\left(\frac{\partial B}{\partial x_1}+\frac{\partial C}{\partial x_2}\right)+\frac{\partial P}{\partial x_1}=0,\\ e_2&=AE+DF-\frac{1}{R}\left(\frac{\partial E}{\partial x_1}+\frac{\partial F}{\partial x_2}\right)+\frac{\partial P}{\partial x_2}=0,\\ e_3&=B+F=0,\\ e_4&=\frac{\partial A}{\partial x_1}-B=0,\\ e_5&=\frac{\partial A}{\partial x_2}-C=0,\\ e_6&=\frac{\partial D}{\partial x_1}-E=0,\\ e_7&=\frac{\partial D}{\partial x_2}-F=0. \end{aligned} \tag{5}$$

The equivalent optimisation problem can now be posed as

$$\min I=\int_A \sum_{i=1}^{7} w_i e_i^2 \,\mathrm{d}A. \tag{6}$$

Subject to the appropriate boundary conditions. The minimisation is carried out over the seven field variables and as the integrand in (6) only contains first derivatives the theory of the calculus of variations provides natural boundary conditions that will be satisfied at any point on the boundary where any of the seven fields are not fully specified.

It will be noticed that in this approach the usual distinction between Dirichlet and Von Neumann boundary conditions vanishes as the velocity vector and its derivatives play an identical role in (4) and (8).

It will also be noticed that as the continuity equation is now simply

$$B+F=0, \tag{7}$$

then this equation can be satisfied directly by replacing the variable F wherever it occurs in (5) by the variable $-B$ and then eliminating e_3 from the summation in (6). The presence of the continuity equation therefore essentially simplifies the solution of the Navier–Stokes equations by the finite element optimisation approach; whilst in most alternative approaches it is a considerable complication.

The least squares approach illustrated above for the Navier–Stokes equation can obviously be applied in a similar manner to any set of non-self-adjoint partial differential equations. The objective function (6) can be generalised as the solution of a set of partial differential equations

$$e_i(x)=0$$

will occur at the minimum of

$$I=\int e^{\mathrm{T}} W e \,\mathrm{d}A \tag{8}$$

for any positive definite operator $W(x)$. However, in this paper we will assume the more simple form (6).

To minimise the objective (6) we will first approximate the area of interest by dividing it into a series of elements in an identical way to that used in any finite element solution technique. The distributed array processor is most suited for the solution of problems that can be divided into regular rectangular elements and that will be the case in all problems described in this paper. On such rectangular elements we will introduce the standard bilinear shape functions $\phi_i(x)$, so the

fields (4) will be approximated by

$$A=\Sigma a_i\phi_i\ (x),\quad B=\Sigma b_i\phi_i\ (x),\quad C=\Sigma c_i\phi_i\ (x),$$
$$D=\Sigma d_i\phi_i\ (x),\quad E=\Sigma e_i\phi_i\ (x),\quad F=\Sigma f_i\phi_i\ (x),$$
$$P=\Sigma p_i\phi_i\ (x). \tag{9}$$

Inside any particular rectangular element only four shape functions are nonzero those corresponding to the nodes at the corners of the element. Again as in many finite element approaches approximations to the fields $A \to P$ are defined once values are given to the field variables at the nodes. Once these values are specified the value of I (6) can be calculated, and so the nodal field values become the optimisation variables. Boundary conditions of the Dirichlet or Von Neumann type determine the appropriate nodal field values and these are simply given the correct value and deleted from the list of optimisation variables. Further details of the application of finite element optimisation to the Navier–Stokes equations can be found in Singh [4].

For any particular mesh the minimization of I (6) is now a finite dimensional optimisation problem. Many codes have been written for the solution of such problems and the necessary safeguards that need to be included into such codes to ensure that the iteration terminates in a predefined neighbourhood of the solution in a finite number of steps are now well known and can be found in many texts, including Dixon [2]. These safeguards have been incorporated in all codes based on finite element optimisation written for the ICL-DAP computer and in all the codes written for sequential machines used in the comparison study.

It is usual when developing optimisation codes to first test them on a quadratic function. In this context a quadratic function implies linear partial differential equations and hence our first example is linear.

3. Linear problems

The first problem investigated was the linear heat conduction equation

$$\frac{\partial}{\partial x_1}\left(K_1\frac{\partial T}{\partial x_1}\right)+\frac{\partial}{\partial x_2}\left(K_2\frac{\partial T}{\partial x_2}\right)=-Q. \tag{10}$$

This is a self-adjoint equation which can be derived from the minimization of

$$I=\int_A\left[K_1\left(\frac{\partial T}{\partial x_1}\right)^2+K_2\left(\frac{\partial T}{\partial x_2}\right)^2-2QT\right]\mathrm{d}A. \tag{11}$$

In this particular case it is easy to show that the set of linear simultaneous equations derived when the finite element optimisation method is applied to (11) for a given mesh is identical to the set obtained by applying Galerkin's method. The solutions will therefore be identical. As this example was being considered as the first test problem for a general approach we chose however not to assemble the stiffness matrix A, which is precisely the Hessian matrix of I. Instead we chose to rearrange (11) as

$$I=\int_A f\,\mathrm{d}A=\sum_{\mathrm{el}}\left\{\int_{\mathrm{el}} f\,\mathrm{d}A\right\}=\sum_{\mathrm{el}} I_{\mathrm{el}} \tag{12}$$

which emphasises the structured nature of the problem and as we are considering a two-dimensional problem the natural mapping of field nodes onto the rectangular grid of the distributed array processors was adopted. The unknown value of the field variable at node K was therefore held on processor K and therefore all the information necessary to calculate I_{el} was held on four neighbouring processors. Most efficient optimisation algorithms are based on the function gradient ∇I, this similarly can be written

$$\nabla I=\sum_{\mathrm{el}}\nabla I_{\mathrm{el}},$$

and use may be made of the fact that any variable only contributes to ∇I_{el} in four surrounding elements and that if a conjugate gradient code is being used then a particular ∇I_{el} only contributes to the change in those unknowns at the nodes of that element. This direct relationship between elements and nodes implies that only immediate neighbour data transfer is required.

In practise when solving the linear problem (11) it was found advantageous to also store the element Hessian matrices

$$\nabla^2 I=\sum_{\mathrm{el}}\nabla^2 I_{\mathrm{el}},$$

use being made of the fact that each element Hessian $\nabla^2 I_{el}$ only contained 16 nonzero elements which were stored in the processor at the NW node of the elements, let us term this matrix A_{el}.

The *Linear Conjugate Gradient Algorithm* therefore takes the following form, when solving $Au = f$.

(1) Evaluate the matrices A_{el} [16 values/processor];
(2) select $u^{(0)}$ and $p^{(0)} = \Sigma p_{el}^{(0)} = u^{(0)}$ [4 values/processor];
(3) set $f^{(1)} = f^{(0)} - Ap^{(0)}$ [parallel multiplication];
(4) if $f^{(1)T} f^{(1)} <$ tolerance *then* stop
(5) set $p^{(1)} = f^{(1)}$ and $k = 1$ [parallel operation];
(6) if $p^{(k)^T} Ap^{(k)} < \epsilon$ *then* stop [parallel multiplication];
else set $\alpha = f^{(k)T} f^{(k)} / p^{(k)T} Ap^{(k)}$
(7) update $u_{el}^{(k+1)} = u_{el}^{(k)} + \alpha p_{el}^{(k)}$ [parallel operation];
and $f_{el}^{(k+1)} = f_{el}^{(k)} - \alpha Ap^{(k)}$ [parallel operation];
(8) update $p_{el}^{(k+1)} = f_{el}^{(k+1)} + \beta p_{el}^{(k)}$ [parallel operation];
where $\beta = f^{(k+1)T} f^{(k+1)} / f^{(k)T} f^{(k)}$
(9) If $f^{(k+1)T} f^{(k+1)} < \epsilon$ or $K > K_{max}$
then stop
else set $k = k + 1$ and goto step 6.

We note that A only occurs in the form Ap and that this product can be formed element by element

$$Ap = \sum_{el} A_{el}\ p_{el}$$

and stored element by element. Full details of the above algorithm are given in Dixon, Ducksbury and Singh [1].

The heat conduction equation was solved for four standard test problems introduced by Stone [5]. In these one temperature was fixed at 1 and all the others were optimisation variables, which implies $\partial T/\partial n = 0$ at all other points of the boundary.

The problem was solved with point sources/sinks of 1.0, 0.5, 0.6, −1.83, −0.27 located at nodes (9,9), (9,60), (54,9), (34,34) and (60,60), respectively, for a 64×64 grid and at relative positions for a smaller grid.

Various conductivity distributions were employed ranging from $K_1 = K_2 = 1$ to randomly generated conductivities in each subregion. Full details are given in Ducksbury [3].

The performance of the algorithm in solving the $K_1 = K_2 = 1$ problem is shown in fig. 1, where the solution time on the ICL-DAP is contrasted with that on the DEC 1091. At the mesh of 41×41 nodes where only approximately one third of the processors on the DAP were in use, the speed up was already 41 compared to the DEC 1091 (14 compared to the ICL 2980). No larger problem could be run on the DEC 1091, but on the DAP a problem of size 64×64 required approximately 4.24 s of CPU time and a problem of size 127×127 (i.e. 16129 unknown) required 132.12 s.

Similar results for one of the ill-conditioned problems are shown in fig. 2.

To verify the expectation that the introduction of more complex shapes, albeit shapes built with

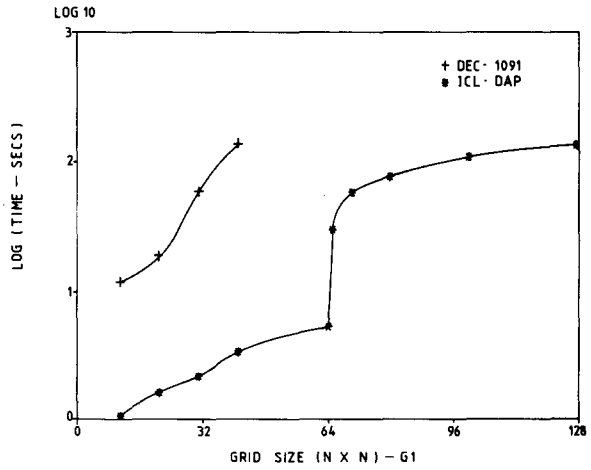

Fig. 1. Performance of the algorithm in solving the $K_1 = K_2 = 1$ problem.

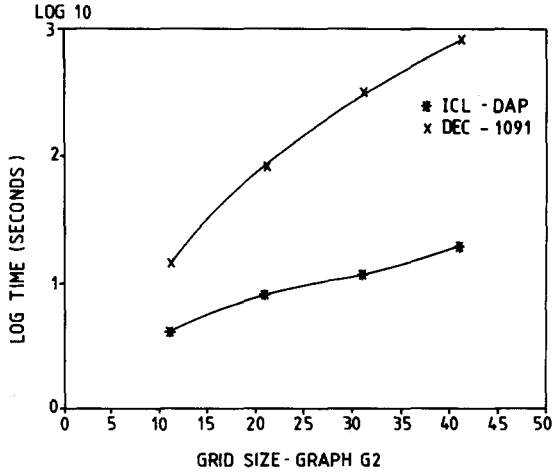

Fig. 2. Performance for one of the ill-conditioned problems.

rectangles, would not effect the DAP solution time, a complex shape was defined by a logic mask and an otherwise similar problem was solved in very similar time. This again contrasts with the effect on CPU times of many other finite element codes, where the complex boundaries would upset the relationship between local and global variables and destroy any diagonal band structure being used, thus significantly increasing the CPU time.

4. Nonlinear problems

The first nonlinear problem to be studied was given by

$$\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} - Ru\frac{\partial u}{\partial x_1} = 0, \tag{13}$$

an analytical solution is known, namely

$$u = 6x_1/Rx_2^2,$$

and the boundary conditions were set to match this. The finite element/optimisation method introduces three fields $A = u$, $B = \partial u/\partial x_1$, $C = \partial u/\partial x_2$ so that for a 64×64 grid there are 3×4096 unknowns before the specification of the boundary conditions. The resulting minimization problem was solved by 2 nonlinear optimisation codes, namely a parallel version of the Fletcher–Reeves conjugate gradient code and a parallel version of the truncated Newton algorithm. In the truncated Newton algorithm, a set of linear equations needs to be solved at each iteration and the linear conjugate gradient code generated for the heat conduction problem was used for this purpose. The *nonlinear conjugate gradient* code can be summarised as follows

(1) Initialise $x_{el}^{(k)}$, $k = 0$.
(2) Calculated $I(x_{el}^{(k)}) = \sum_{el} I_{el}(x_{el}^{(k)})$.
(3) Calculate $\nabla I(x_{el}^{(k)}) = \sum_{el} \nabla I_{el}(x_{el}^{(k)})$.
(4) If $\| \nabla I \|_2^2 < \epsilon$ or $k > k_{max}$ then stop.
(5) If $k = 0$ then set

$$p_{el} = -\sum_{j=1}^{4} \nabla I_{el,j}, \qquad *$$

else set $p^{(k+1)} = -\nabla I^{(k)} + \beta p^{(k)}$, *

where $\beta = \| \nabla I^{(k)} \|_2^2 / \| \nabla I^{(k-1)} \|_2^2$.

(6) Call a line search routine to estimate the step length α
(7) Update $x_{el}^{(k+1)} = x_{el}^{(k)} + \alpha p_{el}^{(k)}$, $k = k + 1$ and goto step 3.

As we have seen the parallelism of the problem can be used to compute I and ∇I very efficiently on the parallel processor; the relation between p_{el} and ∇I_{el} is also very simple as only four elements contribute to ∇I for any particular variable.

Similarly the *Truncated Newton Method* can be summarised as follows:

(1) Set $k = 0$, initialise $x_{el}^{(0)}$; calculate $I(x^{(0)})$.
(2) Set $\alpha = 1$, calculate $\nabla I(x^{(k)})$ and $\nabla^2 I(x^{(k)}) = A$.
(3) If $\| \nabla I_2^2 \| < \epsilon$ or $K > K_{max}$ then stop.
(4) Solve the problem $AU = -\nabla I$ by the linear c.g. method described in section 3.
(5) If u satisfies Wolfes Test I [2] goto step 6 else step 7.
(6) If $I(x^{(k)} + \alpha u)$ satisfies Wolfes Test II + III [2] *then* set $x^{(k+1)} = x^{(k)} + u$; $k + 1$ and goto step 2.
else call a line search to find a value α that does satisfy Wolfes tests II and III, then update x and k and goto step 2.

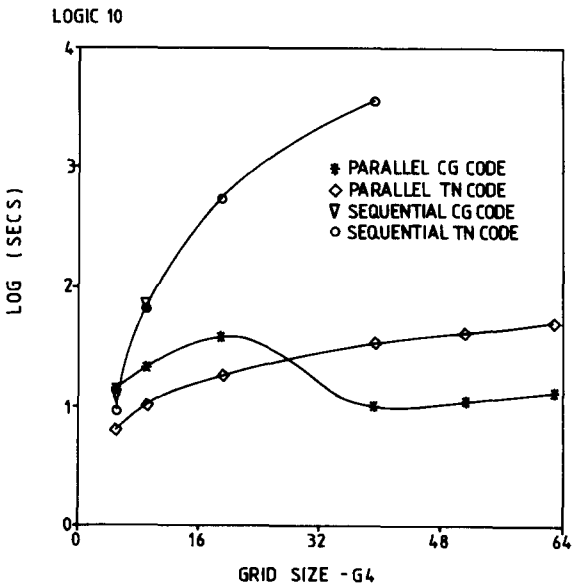

Fig. 3. Results of the tests for the truncated Newton method.

(7) Replace A by an appropriate positive definite matrix and goto step 2.

Full details in both codes are given in Ducksbury [3].

Tests were undertaken on the above problem for a variety of values of R, the most interesting being those with $R = 500$. The grid size was varied up to 64×64. The results are shown in fig. 3. The sequential c.g. code required in excess of one hour to solve a 39×39 grid (4111 unknowns), the DAP just 34 s, this being a speed up of 104 over the DEC 1091 (35 over the ICL 2980) at a point where only 3/8 of the processors of the DAP are in use. For a grid of 64×64 processors (12036 unknowns) the conjugate gradient method on the DAP required 50.76 s, whilst the truncated Newton method only required 13.21 s. Further details can be found in Ducksbury [3].

5. The Navier–Stokes problem

The Navier–Stokes problem described in section 2 has been implemented and tested on the ICL–DAP, unfortunately it was found that there was insufficient store available to use the Truncated Newton method that had proved more efficient on the previous problem but no difficulties

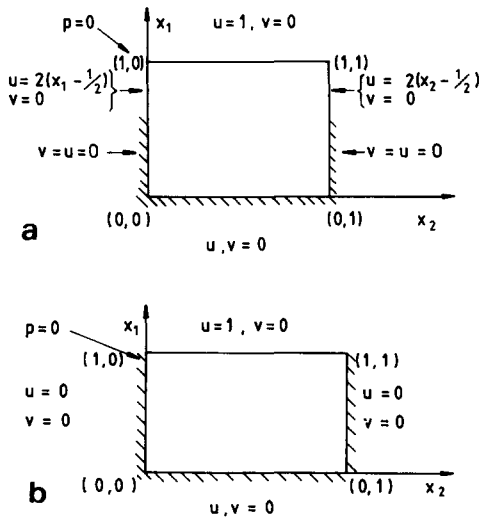

Fig. 4. The boundary conditions for the Navier–Stokes problem.

Table 1

	3×3	5×5		9×9	17×17
Value of I	0.2037	0.080917		0.03533	0.01730
CPU	2:56	5:39		14:11	13:51
Total CPU			36:37		
Iterations	82	164		417	389

were experienced in using the conjugate gradient code.

The boundary conditions imposed are illustrated in fig. 4a, this is a well posed problem and as anticipated the optional function value decreases steadily as the grid is refined (see table 1).

The results shown in this table were obtained by starting the optimisation problem with the finer mesh from the solution with the coarser mesh. With the bilinear element, interpolation at this point does not alter the function value, providing the boundary conditions can be accurately represented by the shape function, which is the case for the above problem.

Difficulties were however experienced when this approach was applied to the cavity driven flow problem (see fig. 4b). On this problem the singularity at the upper corners cannot be matched accurately by the boundary conditions implied by the bilinear element. So when interpolation was attempted the function value increased, as indeed did the optimal function value. This increase reflects the fact that the approximate problem is becoming increasingly similar to an ill posed problem and emphasises the need for special treatment at the upper corners.

6. Conclusions

The finite element optimisation method can be applied to the solution of general nonlinear sets of partial differential equations.

When the set is a function of two dimensions and is steady the problem maps readily onto the ICL Distributed Array Processor and very efficient codes can be implemented.

References

[1] L.C.W. Dixon, P.G. Ducksbury and P. Singh, Technical Report No. 132, Numerical Optimisation Centre, The Hatfield Polytechnic (1982).
[2] L.C.W. Dixon, E. Spedicato and G.P. Szegö, Nonlinear Optimisation: Theory and Algorithms (Birkhäuser Press, Basel, 1980).
[3] P.G. Ducksbury, PhD thesis, The Hatfield Polytechnic (1985).
[4] P. Singh, PhD thesis, The Hatfield Polytechnic (1983).
[5] H.L. Stone, SIAM J.N.A. 5 (September 1968).