Numerical Optimisation Centre

A PARALLEL VERSION OF THE CONJUGATE GRADIENT ALGORITHM

FOR FINITE ELEMENT PROBLEMS

by

L C W Dixon, P G Ducksbury and P Singh

TECHNICAL REPORT NO. 132 DECEMBER 1982

The Hatfield Polytechnic

PAUL DUCKSBURY

THE HATFIELD POLYTECHNIC

NUMERICAL OPTIMISATION CENTRE

A PARALLEL VERSION OF THE CONJUGATE GRADIENT ALGORITHM FOR FINITE ELEMENT PROBLEMS

by

L C W Dixon, P G Ducksbury and P Singh

TECHNICAL REPORT NO. 132 DECEMBER 1982

Abstract

The solution of the partial differential equations arising from the two-dimensional heat conduction equations has been mapped onto the ICL Distributed Array Processor. The solution method followed the Galerkin finite element approach and each processor of the DAP handled its own finite element. The solution was then completed by implementing both the linear and nonlinear versions of the conjugate gradient method. Time comparisons are given with the same solution method on the DEC 1091 system.

The authors would like to thank the SERC for their support and the Queen Mary College DAP Support Unit for permission to use their system.

SERC Grant No. GR/B/4665/5

1. Introduction

This report describes the implementation of two parallel versions of the conjugate gradient algorithm (linear and nonlinear) on the ICL DAP computer and their use in the solution of the 2-D heat conduction equations, as described by Stone, H L [7]. The solutions were compared with similar sequential codes on The Polytechnic's DEC system 10.

2. The Problem

The problem used for testing the implementations was the heat conduction equation in two dimensions. This is given below, the temperature satisfies

$$\frac{\partial}{\partial x}\left[KX\,\frac{\partial T}{\partial x}\right] + \frac{\partial}{\partial y}\left[KY\,\frac{\partial T}{\partial y}\right] = -Q \qquad (1)$$

where KX and KY are thermal conductivities in the x and y directions

Q is the local heat source/sink

T is the temperature

x and y distance coordinates.

It is well known that the solution to equation (1) will occur at the minimum of

$$I = \iint KX\left(\frac{\partial T}{\partial x}\right)^2 + KY\left(\frac{\partial T}{\partial y}\right)^2 - 2QT\ dV \qquad (2)$$

A solution method for a set of p.d.e.'s that can be viewed as an optimisation problem is described in Dixon & Singh [1]. For such a problem the objective function becomes one of solving

$$\min I = \int F(T(x))\ dV \qquad (3)$$

subject to the appropriate boundary conditions.

Now when using finite elements the domain V is covered with a set of grid points x_k and then divided into a set of elements V_i (in our case, of rectangular shape) with grid points at the intersections.

For each element there are shape functions ϕ_{ki} such that

$$\phi_{ki}(x) = 1 \text{ at } x_k$$

$$\phi_{ki}(x) = 0 \text{ at } x_j \text{ for } j \neq k \text{ and at all points } x \notin V_i$$

i.e. $\phi_{ki}(x) = (1 - \frac{x_k \pm x_i}{h})(1 - \frac{y_k \pm y_i}{h})/4$

It is now possible to approximate T(x) by a linear combination of the shape functions

$$\sum_k T_k . \phi_{k.i}(x) \tag{4}$$

so that equation (3) becomes approximately

$$\min I = \int F(\sum_k T_k . \phi_{ki}(x)) dV \tag{5}$$

on substituting (4) into equation (2) we get

$$F(T) = \iint KX\{\sum_i \frac{\partial \phi_i}{\partial x} T_i\}^2 + KY\{\sum_i \frac{\partial \phi_i}{\partial y} T_i\}^2 - 2Q\sum_i \phi_i T_i \; dV \tag{6}$$

and on differentiating this with respect to T_i to get ∇F_i

$$\nabla_i F(T) = \iint 2KX \frac{\partial \phi_i}{\partial x} \{\sum_j \frac{\partial \phi_j}{\partial x} T_j\} + 2KY \frac{\partial \phi_i}{\partial y} \{\sum_j \frac{\partial \phi_j}{\partial y} T_j\} - 2Q\sum_i \phi_i \; dV \tag{7}$$

Differentiating once more gives $\nabla^2 F_{ij}$;

$$\nabla^2_{ij} F(T) = \iint 2KX \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_j}{\partial x} + 2KY \frac{\partial \phi_i}{\partial y} \frac{\partial \phi_j}{\partial y} \; dV \tag{8}$$

When the Q_i's are point sources at the nodes then they can be moved outside the integration, then equation (6) becomes

$$F(T) = \sum_{el} \iint_{el} KX\{\sum_i \frac{\partial \phi_i}{\partial x} T_i\}^2 + KY\{\sum_i \frac{\partial \phi_i}{\partial y} T_i\}^2 \; dV - 2Q_i T_i \tag{6a}$$

equation (7) becomes

$$\nabla_i F(T)_{el} = \iint_{el} 2KX \frac{\partial \phi_i}{\partial x} \{\sum \frac{\partial \phi_j}{\partial x} T_j\} + 2KY \frac{\partial \phi_i}{\partial y} \{\sum \frac{\partial \phi_j}{\partial y} T_j\} \; dV - 2Q_i \tag{7a}$$

and equation (8) becomes

$$\nabla^2_{ij}(T)_{el} = \iint_{el} 2KX \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_j}{\partial x} + 2KY \frac{\partial \phi_i}{\partial y} \frac{\partial \phi_j}{\partial y} \; dV \tag{8a}$$

In the problems taken from Stone [7] which were the basis of those solved in this study, Neumann type boundary conditions were imposed.

The solution of problem (5) then becomes equivalent to the solution of the set of equations (6). If the partial differential equations are linear this is a set of linear equations which could be written Au = f. However, if the partial differential equations are nonlinear then (6) will also be nonlinear.

2.1 Solution of the set of equations

When solving equations of the form

$$A\underline{u} = \underline{f}$$

where A is a real N x N matrix

u an N unknown vector

f an N known vector

if N is large there may be problems with computer storage. In the finite element approach the matrix A will be held as $\sum_{e=1}^{E} A^e$ (namely, the sum of all the element matrices).

All of the values in A^e will be zero except for those occurring in rows/columns corresponding to variables in the eth element. It is, therefore, possible to solve the above set of equations without assembling the original matrix A.

The matrix A is sparse and could be assembled using a particular order of the grid points to give a diagonal structure, in this particular problem the matrix would have 9 nonzero diagonals arranged in 3 groups of three. This structure arises because any node can only lie in four elements and these share 9 nodes. Special solution methods exist utilising this diagonal structure but that advocated here is, however, based on using the element matrices individually, P Singh [6].

Using this approach we can evaluate $A\underline{u}$ element by element, by taking each element matrix in turn and multiplying it with the correct elements of the

vector $\underline{u}$, the answers being placed in the correct global positions. This makes the use of a conjugate gradient method advantageous.

3. Conjugate gradients

The conjugate gradient method is an iterative method that converges to the true solution in a finite number of iterations assuming no rounding errors. The idea was initially presented by Hestenes and Stiefel (1952)[3] and subsequently modified for optimisation purposes by Fletcher & Reeves (1964) [2]. We intend to show that it can be implemented very efficiently on the ICL-DAP parallel processing computer.

4. System/language for implementation

The DAP is an SIMD machine which has a total of 4096 individual processors arranged in a 64 x 64 matrix structure, each processor connected to its nearest four neighbours.

The language used for the implementation is the high level language DAP-FORTRAN which is an extension to the existing ICL 2900 FORTRAN, comprising 51 built-in macro routines for the manipulation of vectors and matrices.

Access to the DAP is via the Host 2980 computer. Details of the language constructs and program development process can be found in [8] and [9].

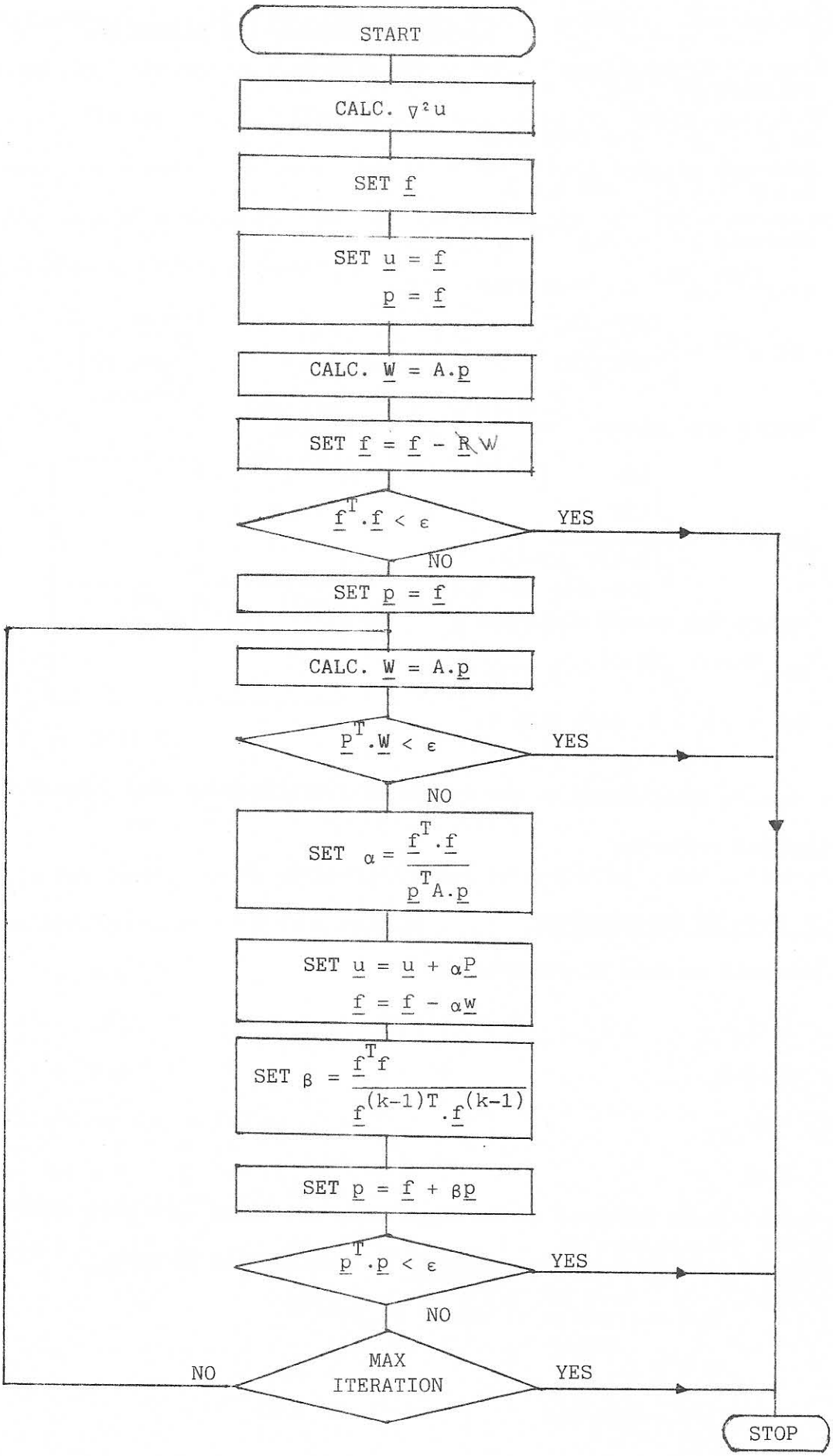
5. The algorithm

5.1 The linear algorithm

The basic linear conjugate gradient algorithm used is described below (and on Flowchart 1).

Note that we are solving the set of equations $A\underline{u} = \underline{f}$.

1. Evaluate $\nabla^2 F$ (A)
2. Initialise the right hand side vector $\underline{f}^{(0)}$ (it is set equal to the source/sink points, and zero elsewhere).
3. Set $\underline{u}^{(1)} = \underline{f}^{(0)}$ and $\underline{p}^{(0)} = \underline{f}^{(0)}$ ($\underline{u}$ being the unknown and $\underline{p}$ the search direction).


START
CALC. $\nabla^2 u$
SET $\underline{f}$
SET $\underline{u} = \underline{f}$
$\underline{p} = \underline{f}$
CALC. $\underline{W} = A.\underline{p}$
SET $\underline{f} = \underline{f} - R\underline{W}$
$\underline{f}^T.\underline{f} < \epsilon$
YES
NO
SET $\underline{p} = \underline{f}$
CALC. $\underline{W} = A.\underline{p}$
$\underline{P}^T.\underline{W} < \epsilon$
YES
NO
SET $\alpha = \frac{\underline{f}^T.\underline{f}}{\underline{p}^T A.\underline{p}}$
SET $\underline{u} = \underline{u} + \alpha\underline{P}$
$\underline{f} = \underline{f} - \alpha\underline{w}$
SET $\beta = \frac{\underline{f}^T\underline{f}}{\underline{f}^{(k-1)T}.\underline{f}^{(k-1)}}$
SET $\underline{p} = \underline{f} + \beta\underline{p}$
$\underline{p}^T.\underline{p} < \epsilon$
YES
NO
MAX ITERATION
NO
YES
STOP

FLOWCHART 1

4. Evaluate $\underline{w}^{(0)} = A.\underline{p}^{(0)}$ (Note that we can perform this operation without having to assemble the matrix A).

5. Evaluate $\underline{f}^{(1)} = \underline{f}^{(0)} - \underline{w}^{(0)}$

6. If $\underline{f}^{(1)T}.\underline{f}^{(1)} < \varepsilon$ then stop

7. Set $\underline{p}^{(1)} = \underline{f}^{(1)}$ and $k = 1$

8. Evaluate $\underline{w}^{(k)} = A.\underline{p}^{(k)}$

9. If $\underline{p}^{(k)T}.\underline{w}^{(k)} < \varepsilon$ then stop

10. Set $\alpha^{(k)} = \dfrac{\underline{f}^{(k)T}.\underline{f}^{(k)}}{\underline{p}^{(k)T}.\underline{w}^{(k)}}$

11. Update the unknown $u^{(k+1)} = \underline{u}^{(k)} + \alpha^{(k)}.\underline{p}^{(k)}$

and $\underline{f}^{(k+1)} = f^{(k)} - \alpha^{(k)}.\underline{w}^{(k)}$

12. Set $\beta^{(k)} = \dfrac{\underline{f}^{(k)T}.\underline{f}^{(k)}}{\underline{f}^{(k-1)T}.f^{(k-1)}}$

13. Update the search direction $\underline{p}^{(k+1)} = f^{(k+1)} + \beta^{(k)}.p^{(k)}$

14. If $\underline{p}^{(k+1)T}.\underline{p}^{(k+1)} < \varepsilon$ then stop

15. Set $k = k + 1$, Goto step 8.

Note that an upper bound on the number of iterations is also imposed as a termination criteria.

5.1.1 Form of calculations

The products we need to evaluate are

i) $\underline{f}^{T}.\underline{f}$

ii) $\underline{p}^{T}.A.\underline{p}$

iii) $A.\underline{p}$

iv) $\underline{p}^{T}.\underline{p}$

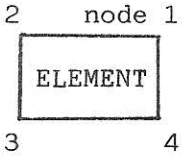
The calculations involved in the algorithms are based upon data which is associated with each of the elements, in each element we have

a 4 x 1 vector of values to form ∇F

a 4 x 1 " " " " " $\underline{p}$

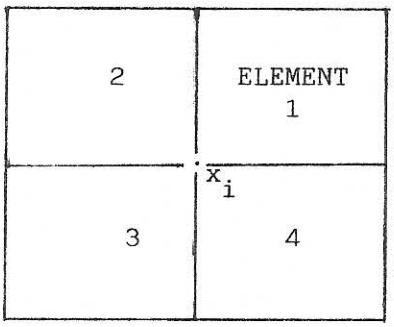
and a 4 x 4 matrix " " " " $\nabla^2 F$.

The first product $\underline{f}^T.\underline{f}$ is a straightforward vector product. For the other three products, however, we must have the values of $\underline{p}$ at each of the grid points x_i. The way in which this is achieved is to add contributions of the elements at a node. At each node there are 4 neighbouring elements, (or in the case of a boundary node just two values, or one for a corner node). The local node and element numbering is



For nodes

and



For the elements

to form P_i at node x_i

$$P_i = \sum_{el=1}^{4} P_j^{el}$$

where j is the local node in element el which corresponds to the global node i.

i.e. take the 3rd value of P from element 1

\+ 4th " " " " " 2

\+ 1st " " " " " 3

\+ 2nd " " " " " 4.

The calculations are as follows:

(i) $p^T.A.p = \sum_{el=1}^{E} [\sum_{j=1}^{4} \sum_{k=1}^{4} P_j.A_{j,k}^{el}.P_k]$

(ii) $A.p = \sum_{j=1}^{4} \sum_{k=1}^{4} A_{j,k}^{el}.P_k$ for el = 1, ..., E. In each element the four values of the resulting vector are placed back at the correct local nodes.

(iii) $$p^T.p = \sum_{el=1}^{N} [(\sum_{el=1}^{4} p_j^{el})(\sum_{el=1}^{4} p_j^{el})]$$

where N = the number of nodes in the system

and E = " " " elements " " " .

5.2 The nonlinear algorithm

The basic nonlinear conjugate gradient algorithm due to Fletcher and Reeves [2] is implemented as described below (and on Flowchart 2).

Initially we guess values for $\underline{u}_i^{(1)}$ $i = 1, 2, \ldots, N$ and then evaluate $\nabla^2 F$ at $\underline{u}^{(1)}$.

1. Set k = 1
2. Evaluate $\nabla F^{(k)}$
3. If $\nabla F^{(k)T} \nabla F^{(k)} < \varepsilon$ then stop
4. If k = 1

   then set $\underline{P}^{(k)} = -\nabla F^{(k)}$

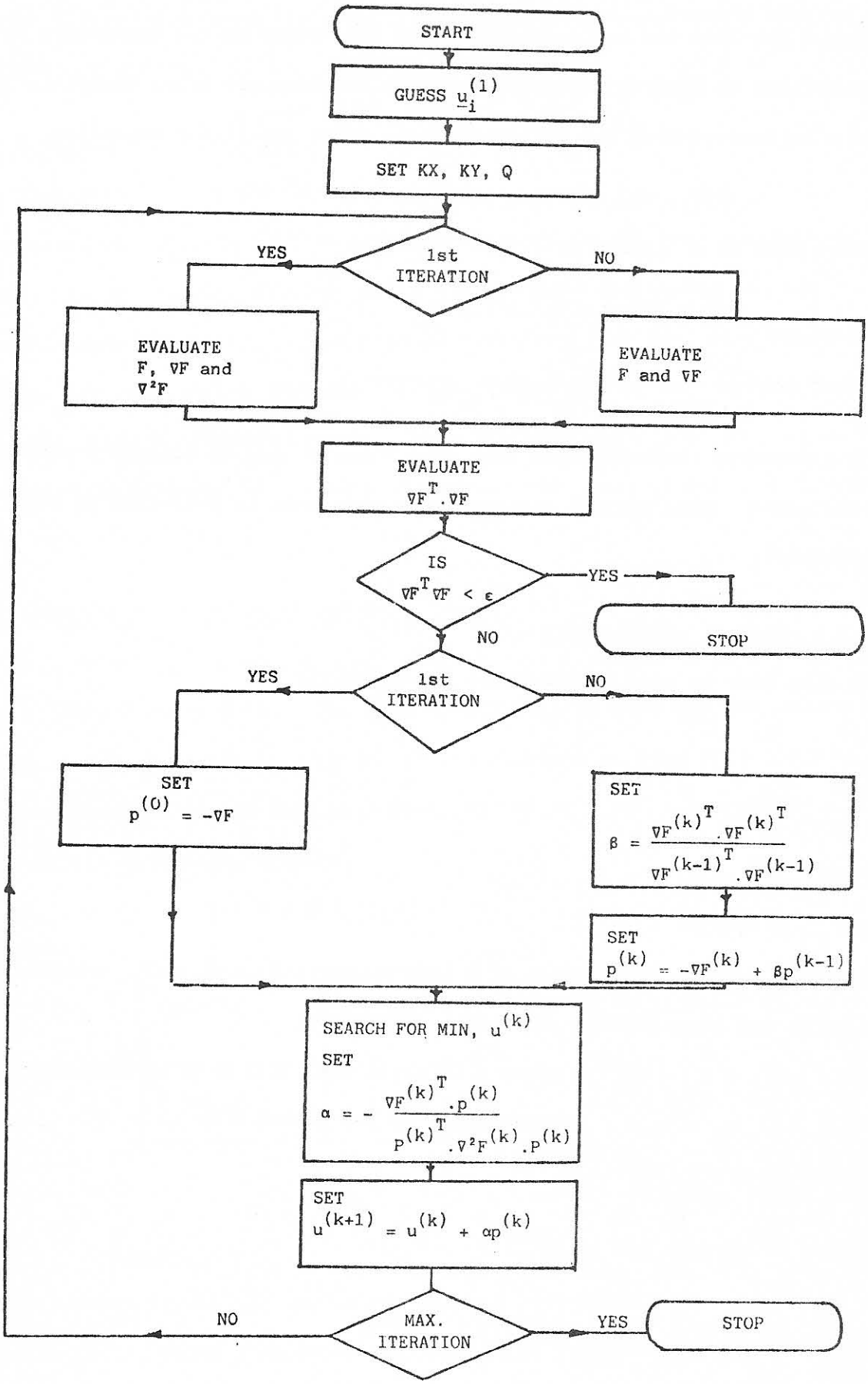
   else set $\beta^{(k)} = \dfrac{\nabla F^{(k)T} \nabla F^{(k)}}{\nabla F^{(k-1)T} \nabla F^{(k-1)}}$

   and $\underline{P}^{(k)} = -\nabla F^{(k)} + \beta^{(k)} \underline{P}^{(k-1)}$
5. Evaluate α as $\alpha^{(k)} = \arg\min (F + \alpha \underline{p})$
6. Update the unknown $\underline{u}^{(k+1)} = \underline{u}^{(k)} + \alpha^{(k)} . \underline{p}^{(k)}$
7. Set k = k+1
8. If $k \leq k_{max}$

   then GOTO step 2

   else stop.

Tests were undertaken to investigate the additional cost involved in using this algorithm to solve quadratic problems rather than the simpler algorithm described in the previous sections. Additional costs can be expected as at stage 2 the gradient vector is recalculated at each iteration by entering the finite element procedure rather than updated as in the simpler algorithm.



FLOWCHART 2

Note 1 The function F is not actually needed (or calculated) with the present objective function.

Note 2 The unknown u is our temperature.

Usually the Fletcher-Reeves algorithm involves a complicated line search at step 5 but this was not introduced into this study as the tests were in fact undertaken on quadratic functions so the solution was known analytically. So α is derived from the following fact, given the Taylor expansion

$$F(T + \alpha p) = F(T) + \alpha(\nabla F^T.p) + \tfrac{1}{2}\alpha^2.P^T.\nabla^2 F.P$$

there will be a minimum of F when $\frac{\partial F}{\partial \alpha} = 0$

$$\frac{\partial F}{\partial \alpha} = \nabla F^T.p + \alpha p^T.\nabla^2 F.p$$

hence $\alpha = \dfrac{-\nabla F^T.P}{p^T.\nabla^2 F.p} = \dfrac{-\nabla F^T.P}{P^T[\nabla F^{(k)} - \nabla F^{(k-1)}]}$

In a complete inplementation then step 5 would need to contain a parallel line search, some initial experience in this area is described in Patel (1982)[4].

5.2.1 Form of calculations

In this case we need to calculate the following

(i) ∇F.

(ii) P.

(iii) $\nabla F^T.P$.

(iv) $\nabla F^T.\nabla F$.

(v) $P^T.\nabla^2 F.P$

Again we need to form values of P (and also ∇F) at each node, and this is the same as was described for the linear algorithm.

(i) $\nabla F_i = \sum_{el=1}^{4} \nabla F_j^{el}$ where j is again the local node in element el which corresponds to the global node i.

(ii) $P_i = \sum_{el=1}^{4} P_j^{el}$

(iii) $\nabla F^T.p = \sum_{i=1}^{N} [(\sum_{el=1}^{4} \nabla F_j^{el})(\sum_{el=1}^{4} P_j^{el})]$

(iv) $\nabla F^T \nabla F = \sum_{i=1}^{N} [(\sum_{el=1}^{4} \nabla F_j^{el})(\sum_{el=1}^{4} \nabla F_j^{el})]$

(v) $P^T.\nabla^2 F.P = \sum_{i=1}^{E} [\sum_{j=1}^{4} \sum_{k=1}^{4} P_j \nabla^2 F_{j,k}^{el}.P_k]$

Section 5.3(iv) describes how some of the calculations are either avoided or kept to a minimum.

5.3 Detailed implementation

(i) Modularity

This method of solution and the problem naturally divide into a number of separate tasks, namely, setting up the heat sources and sinks, thermal conductivities KX and KY, defining shape functions, the quadratic points and weights, and calculating $\nabla^2 F$ (and ∇F in nonlinear case), in addition to the conjugate gradient routine itself. Each of these was placed in a separate subroutine to minimise any recompilation/reassembly in the event of alterations to Q, KX or KY etc.

(ii) Mapping the problem to the DAP

In the problem we have an n x n set of grid points and an (n-1) x (n-1) set of elements, where $n \leq 64$. The grid points will be held one per processor, and when we need to use information in the elements we also hold these one per processor, as though they have been shifted up 1 place diagonally with the last row and column ignored.

e.g. Global storage for a 3 x 3 grid.

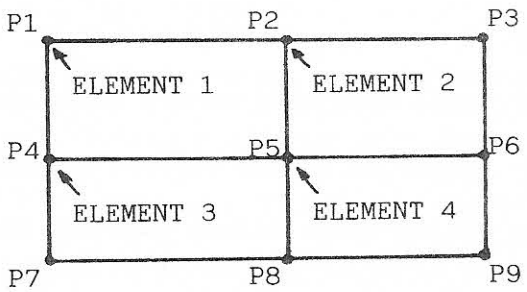


Figure 1

An array G2FEL (,, 16) is used where plane 1 will contain all the values in position (1,1), plane 2 those in position (1,2) ... plane 16 those in (4,4). One of the first tasks of the DAP entry subroutine is to set up two logical

masks, one for masking out unwanted grid points and the other for unwanted elements.

Earlier the method of calculating ∇F at each of the grid points was described, (5.1 applying this to the DAP, then we can use the available shift indexing facilities. Given that we have declared an array GFEL(,,4) (Gradient of F at each ELement), we will illustrate the calculation with a 3 x 3 grid with four elements, then the indexing used to work out the gradient at the point 4, (Figure 2) would be as follows (also assume we are centered on a processor at point 4).

To get the 3rd value of ∇F from element 1 we would need to move to point 1, hence using GFEL (-,,3).

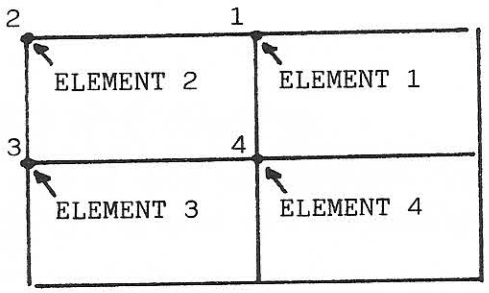


Figure 2

For the 4th value of ∇F from element 2 we would need to move to point 2, hence using GFEL(-,-,4) and similarly for the other 2 values

1st from element 3 → GFEL(,-,1)

2nd from element 4 → GFEL(,, 2) (Note that here we are using no shift, since it is our current position).

Extending this to an n x n grid and adding these values together will give us ∇F at each of the grid points, i.e.

```
GF = 0
GF(MASK1)  = GFEL(-,,3) + GFEL(-,-,4) + GFEL(,-,1) + GFEL(,,2)
```

and $\nabla F^T.\nabla F$ being worked out by

```
GFGF = SUM (GF**2)
```

Note how easy the overall summation is in parallel.

Once this information has been worked out, a later stage of the algorithm requires the calculation of $P^T.\nabla^2F.P$. This time we are centred on the element and needing the values of P at the 4 neighbouring nodes. Using Figure 2, then if we are at element 2, then

P1 will come from P(,+)

P2 will come from P(,) or just P (our current position)

P3 will come from P(+,)

and P4 will come from P(+,+)

(iii) Mapping the problem onto a sequential machine

The two main points to note here are, firstly, that given that we 'are on' a particular element we must be able to find out the global numbers of the four local nodes in order to perform our calculations (this posed no problem on the DAP with its indexing facilities) and secondly, the calculation of $P^T.\nabla^2F.P$ (or $P^T.A.P$) without forming the global matrix, the latter being done in two stages by first forming $\nabla^2F.P$ as is described earlier in the section. Both this and the former calculation when run on a sequential machine require an additional array containing for each element, the global node numbers of that element's local nodes. This is not necessary on the DAP.

(iv) Reduction of calculations

It was mentioned earlier how we could reduce some of the calculations involved. Firstly, for calculating P at each grid point, since in the algorithm we initially set P to $-\nabla F$, then at a later iteration k we can just calculate the contribution of ∇F at each grid point and modify the value of p by subtracting this from $p^{(k-1)}$. This also avoids having to store 4 values of P in each element which reduces the overall storage.

Secondly, in working out $\nabla F^T.P$ we will previously in the algorithm have worked out ∇F and P for each grid point hence

$$\nabla F^T.P = \sum_{i=1}^{N} \left[\left(\sum_{el=1}^{4} \nabla F_j^{el} \right) \left(\sum_{el=1}^{4} P_j^{el} \right) \right]$$

is simply

$$\nabla F^T.P = \sum_{i=1}^{N} \nabla F_i.P_i$$

which is the standard vector product of two 1 x 4096 element vectors (the implementation on the DAP has these held as 64 x 64 matrices and $\nabla F^T.p$ is obtained as GFP = SUM(GF*P)).

6. Test problems and results

The main problems to be described later in this section come from Stone [7] who ran n x n problems for n = 11, 21, 31.

The programs were tested at a number of stages up to the final completion and at each of the stages various data items were traced out on the DAP.

(i) The coefficient routine was checked to ensure that it was producing the correct element matrices as compared with the sequential one.

(ii) The boundary routine for adjusting the appropriate element matrices was verified,

(iii) as were the masks in the entry routine for fixing the known boundary values of temperature and gradient to 1 and 0 respectively.

6.1 The preliminary tests themselves consisted of three main cases,

(i) with the right hand side equal to zero

(ii) with the right hand side equal to -2.0

(iii) with the right hand side equal to a point source/sink.

6.1.1 CASE 1

For $\nabla^2 T = -Q$

where

Q is a constant 0

T is fixed at all boundary points to the value of 1.0

g is fixed at all corresponding points on the boundary to 0.0

KX and KY fixed at 1.0

the true solution to this problem is

$$T = 1 + x + y + xy$$

and both parallel and sequential programs were successfully run to solve this problem for a 5 x 5 and 11 x 11 grid size.

6.1.2 CASE 2

For $\nabla^2 T = -Q$

where

Q is a constant 2

and T, g, KX, KY are the same as before in Case 1

the true solution to this problem is

$$T = 1 + X + Y + XY - Q/4.(X^2 + Y^2)$$

and again both programs run the problem for 5 x 5 and 11 x 11 grid sizes, all answers agreeing with the above equation.

6.1.3 CASE 3

For $\nabla^2 T = -Q$

where

Q is a point source

T is fixed at just one boundary node, and the unknown at remaining points is eliminated

g is fixed at the one corresponding boundary point to zero

KX, KY variable and to be described later.

Before the larger sized problems referred to by Stone [7] were run, a number of small and trivial point source problems were tested.

1. A 3 x 3 problem with just one point source of value 2.0 located in the centre at position (2,2)
2. A 5 x 5 problem with 3 point sources located at the positions (2,3), (3,3) and (4,2) with values of 1.5, 2.0 and 1.0 respectively.
3. As for 2. except for a point sink located at the position (2,4) with a value of -1.0

4. As for 3. but with an 11 x 11 grid size.

All the four programs were satisfactorily compared before moving on to the main set.

6.1.3.1 POINT SOURCES (main problem)

The problems all had three point sources and two point sinks and these were located as below.

For 11 x 11 at (2,2), (2,10), (8,2), (6,6), (10,10)

For 21 x 21 at (2,2), (2,20), (14,3), (10,11), (20,20)

For 31 x 31 at (3,3), (3,27), (23,4), (14,15), (27,27)

For 41 x 41 at (6,6), (6,36), (28,6), (20,20), (36,36) and for

64 x 64 at (9,9), (9,60), (54,9), (34,34), (60,60) with values of

1.0 0.5 0.6 -1.83 -0.27 respectively.

As has been previously mentioned the boundary conditions were imposed by fixing the temperature at one point on the boundary to 1.0 and the corresponding value of the gradient to 0.0, and all other temperatures to 0.

Note that in our nonlinear algorithm the values of Q were divided by 4 since each Q comes into 4 sets of equations, its overall effect must be Q and not 4Q.

6.1.3.2 Conductivities

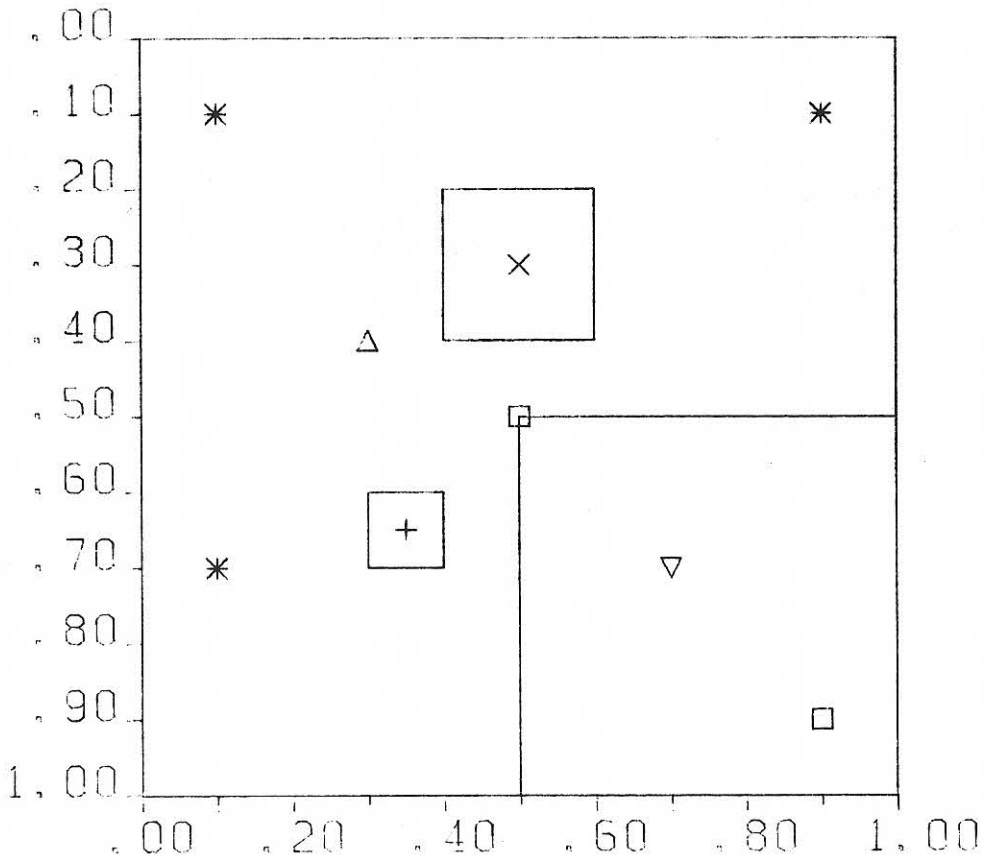
Four different conductivity distributions were used in the tests. They were specified independently in the x and y directions and located at the centre of an element.

PROBLEM 1. The model problem with KX and KY equal to unity over the entire region.

PROBLEM 2. The generalized model problem with KX and KY both constant, but with KX 100 times greater than KY.

PROBLEM 3. This was the heterogeneous test problem, here the region was subdivided into a number of smaller regions, refer to Figure 3.

Thermal Conductivities



△ KX, KY=1
× KX, KY=0
▽ KX=1, KY=100
+ KX=100, KY=1
□ HEAT SINK
✳ HEAT SOURCE

Figure 3

PROBLEM 4. This was the same as for 3. except that in one of the regions, where KX and KY had been set to 1.0, the conductivities came via random numbers. Numbers in (0,1) were generated and if less than 0.1 the conductivity in x and y directions were set to zero, otherwise they were set to one. This should have generated a large number of obstructions to heat flow. Problems 3 and 4 are of the same order of complexity, this being confirmed by the results (Graph 7).

6.2 Numerical Results

The numerical results obtained are divided into two sets depending on the code used, i.e. linear and nonlinear. Those for the linear code are given in Appendix 1, Tables 1-4 and for the nonlinear code in Tables 5-8. Timings of the runs and number of iterations (of the conjugate gradient solution routine) for each grid size are given. These results are shown graphically in Appendix 2 where the grid size is on the x-axis and the LOG of time on the y-axis. (Note that the LOG of time had to be used because of the large range of times obtained - particularly for the sequential cases).

In all but a few cases (which have been indicated) the tolerance for termination was 1E-6. A suitable upperbound to the iterations was set at 2* no. of points.

It is apparent from the graphs that the parallel implementations are considerably faster than the sequential ones, Graph 1 was taken as far as a 64 x 64 problem for the parallel case just to prove that it was possible within a reasonable time but this could not be loaded on our sequential computer.

The curves/lines for the parallel case increase only slightly in comparison with the sequential runs whose times soon become large as the number of unknowns is increased or as the ill-conditioning gets worse.

As would be expected the linear algorithm exhibits much lower times than the nonlinear algorithm which of course has the extra computational effort required in the calculation of the gradient at each iteration.

The only still unexplained results came from the nonlinear sequential algorithm when run for problems 3 and 4. The resulting gradient failed to be significantly reduced and the iteration was stopped after 2N iterations. Surprisingly the parallel algorithm on the other hand terminated successfully on g^Tg tolerance although both programs were setting the same subregions for KX and KY and using the same algorithms. This seems to imply that the parallel implementation is much less affected by rounding error even though the arithmetical precision was slightly less, i.e. 7 v 8 decimal places.

7. Conclusions

The two parallel implementations have exhibited the fact that the solution of p.d.e.'s using this method of approach is very suitable to the SIMD class of machines and in particular the DAP.

Almost certainly one of the main advantages of the DAP in this case was the fact that the processors are connected to their nearest four neighbours, via row and column data highways, and the powerful shift indexing facilities make good use of this. Thus we have a simple but an extremely effective means of communicating with other nodes and neighbouring elements.

This facility can be compared to the two sequential programs which need to keep, for each element, a separate record of the four neighbouring (local) nodes, which must be set up and then indexed correctly. This is an overhead for the sequential case, not to mention the fact that it is a complication in the writing and checking of the code which is not present in the algorithms.

From the DAP's point of view it makes no significant difference whether it is calculating just 4 element matrices (a 3 x 3 problem) or 3,969 element matrices (a 64 x 64 problem) as either most or none of the of the processors will be switched off (masked out).

The algorithms that were employed for these solutions are basic with no sophisticated improvements though many are known. In theory we should terminate with the correct solution in at most N iterations (where N is the

number of unknowns) but this assumes we use exact arithmetic with no rounding errors, which in practice is not the case. Improvements to the basic algorithms can be made, see for instance Powell [5].

Other major improvements are wellknown such as preconditioning, the multi-grid approach, etc., where the system of equations

$$A\underline{x} = \underline{y}$$

is transformed into a new system

$$\hat{A}\hat{\underline{x}} = \hat{\underline{y}}$$

which will have a much smaller condition number in order to speed up the convergence. Parallel implementation of such a method will be the basis of additional work.

References

1. L C W Dixon and P Singh, "The solution of PDE's via finite elements and optimisation on a parallel processor," TR 117, The Numerical Optimisation Centre, The Hatfield Polytechnic, 1982.

2. R Fletcher and C M Reeves, "Function minimisation by conjugate gradients," Computer Journal, vol. 7, 1964.

3. M Hestenes and E Stiefel, "Methods of conjugate gradients for solving linear systems," J. Res. Nat. Bu. Standards, No. 49, 1952.

4. K D Patel, "Implementation of a parallel (SIMD) modified Newton algorithm on the ICL DAP," TR 131, The Numerical Optimisation Centre, The Hatfield Polytechnic, 1982.

5. M J D Powell, "Restart procedures for the conjugate gradient method," Math. Programming, Vo. 12, pp 221-254, 1975.

6. P Singh, "A note on the relative efficiency of conjugate gradient algorithms for solving linear equations arising from finite element calculations," TR 126, The Numerical Optimisation Centre, The Hatfield Polytechnic, 1982.

7. H L Stone, "Iterative solution of implicit approximations of multi-dimensional PDE's," S.I.A.M. Num. Anal. Vol. 5, No. 3, Sept. 1968.

8. "The DAP-FORTRAN Language," Tech. Pub. 6918 ICL.

9. "The development of DAP programs," Tech. Pub. 6920 ICL.

Appendix 1

Linear

Table 1 - Problem 1 with KX = 1, KY = 1

Table 2 - Problem 2 with KX = 100, KY = 1

Table 3 - Problem 3 with KX, KY subdivided

Table 4 - Problem 4 with KX, KY subdivided, with one region generated from random numbers.

Normal termination on $\sqrt{p^T p}$ < 1E-6 or

$\sqrt{p^T Ap}$ < 1E-6

Nonlinear

Table 5 - Problem 1

Table 6 - Problem 2

Table 7 - Problem 3

Table 8 - Problem 4

Normal termination on $g^T g$ < 1E-6

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3
	SEQ.	PAR.	SEQ.	PAR.	
11	11.7	1.06	46	46	11.0
21	18.8	1.61	85	85	11.7
31	58.1	2.18	125	125	26.6
41	2:17.1	3.32	167	205	41.3
64	note 1	5.17	note 1	339	-

TABLE 1

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3
	SEQ.	PAR.	SEQ.	PAR.	
11	14.7	4.1	240	265	3.6
21	1:23.3	8.17	480	555	10.2
31	5:12.2	11.9	760	819	26.2
41	13:46.6	20.01	1060	1397	41.0

TABLE 2

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3	
	SEQ.	PAR.	SEQ.	PAR.		
11	11.05	5.9	246	391	1.87	
21	1:50.4	11.69	650	800	9.4	
31	6:54.0	21.34	950	1481	19.4	note 2
41	note 1	-	note 1	-	-	

TABLE 3

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3	
	SEQ.	PAR.	SEQ.	PAR.		
11	12.03	6.1	260	403	1.97	
21	2:33.5	12.79	875	880	12.00	) note 2
31	6:59.8	20.15	1000	1400	20.83	)
41	note 1	-	note 1	-	-	

TABLE 4

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3
	SEQ.	PAR.	SEQ.	PAR.	
11	33.0	8.5 6.9	38	37	3.9
21	4:15.9	15.7 12.5	70	70	16.2
31	12:38.4	22.7 18.2	100	101	33.4

TABLE 5

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3	
	SEQ.	PAR.	SEQ.	PAR.		
11	2:15.7	33.1 26.6	156	147	4.1	)
21	17:11.1	1:04.7 51.9	298	288	15.9	) note 2
31	54:37.0	1:37.9 1:18.5	439	436	33.5	)

TABLE 6

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3	
	SEQ.	PAR.	SEQ.	PAR.		
11	3:32.1	53.8 43.1	242*	241	note 4	
21	51:16.0	2:16.3 1:49.2	882*	609	note 4	
31	note 1	3:12.9 2:34.5	note 1	863	-	note 2

TABLE 7

GRID SIZE	TIME(S)		ITERATIONS		GAIN note 3	
	SEQ.	PAR.	SEQ.	PAR.		
11	3:37.0	57.2 45.8	242*	257	note 4	
21	note 1	2:27.8 1:58.2	note 1	664	-	
31	note 1	3:22.2 2:41.0	note 1	904	-	note 2

TABLE 8

* 2 x no. of points imposed as limit, maximum gradient not reduced significantly towards this limit, namely 0.12, 0.012 and 0.18 respectively.

nn times when $\frac{\partial \phi_i}{\partial x}, \frac{\partial \phi_i}{\partial y}$ are stored at first iteration

Notes

Note 1 Some jobs not run due to time considerations, and in the case of Table 1 (64 x 64) sequential) there was insufficient store available. This test case required 116,836 words (~114K) of store plus the program area itself which is beyond the limits for running an over-night batch job. The only way of running the job would be to use random access files on disc for holding some of the arrays.

Note 2 Necessary to reduce terminating tolerance to 0.0001.

Note 3 The improvement of DAP version over sequential DEC 1091 version.

Note 4 No gains were worked out due to the unexplained sequential results.

Appendix 2

Graph 1	-	Problem 1	Linear
Graph 2	-	Problem 2	
Graph 3	-	Problem 3	
Graph 4	-	Problem 4	

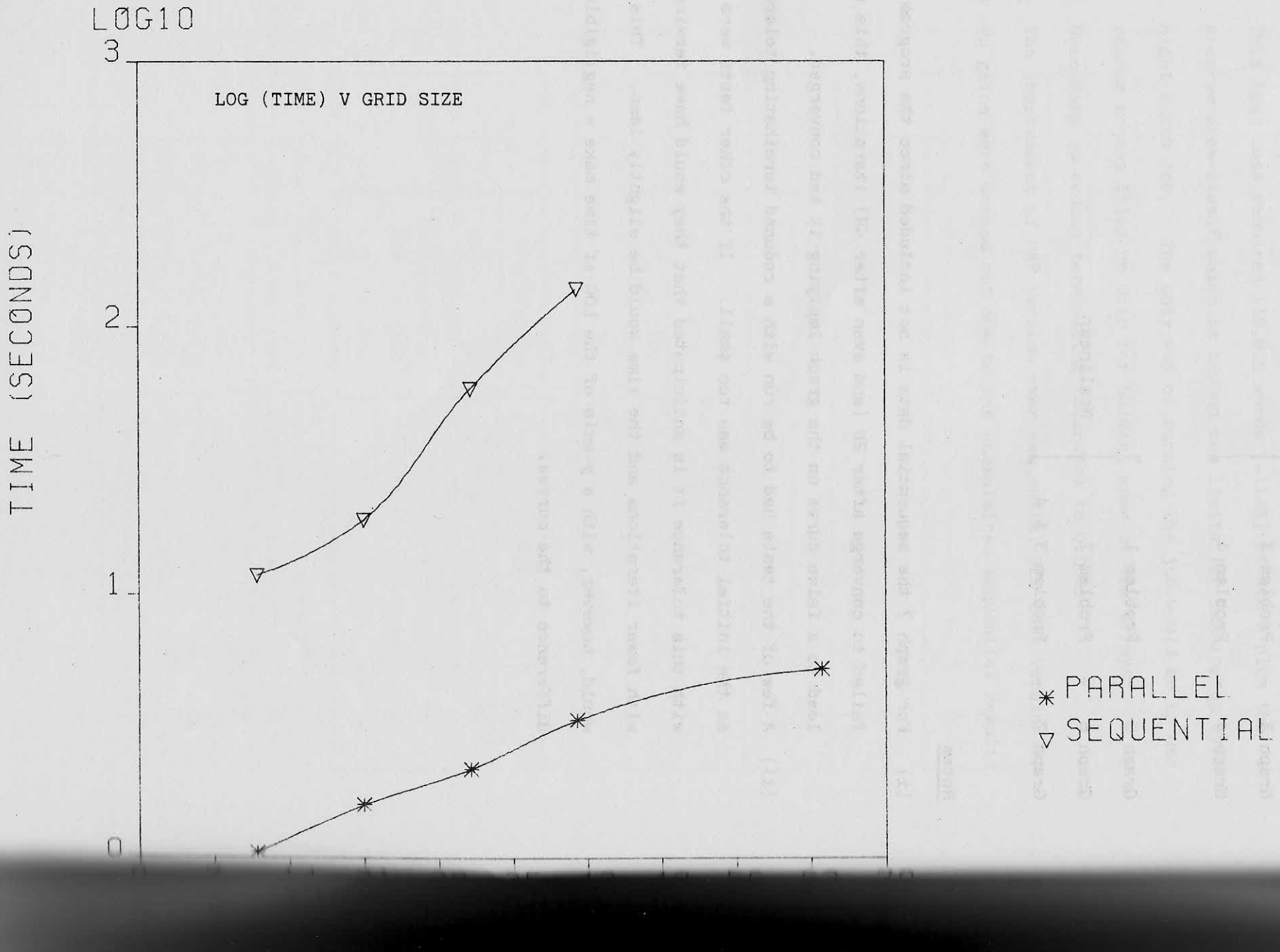
Graph 5	-	Problem 1	Nonlinear
Graph 6	-	Problem 2	
Graph 7	-	Problem 3 & 4	

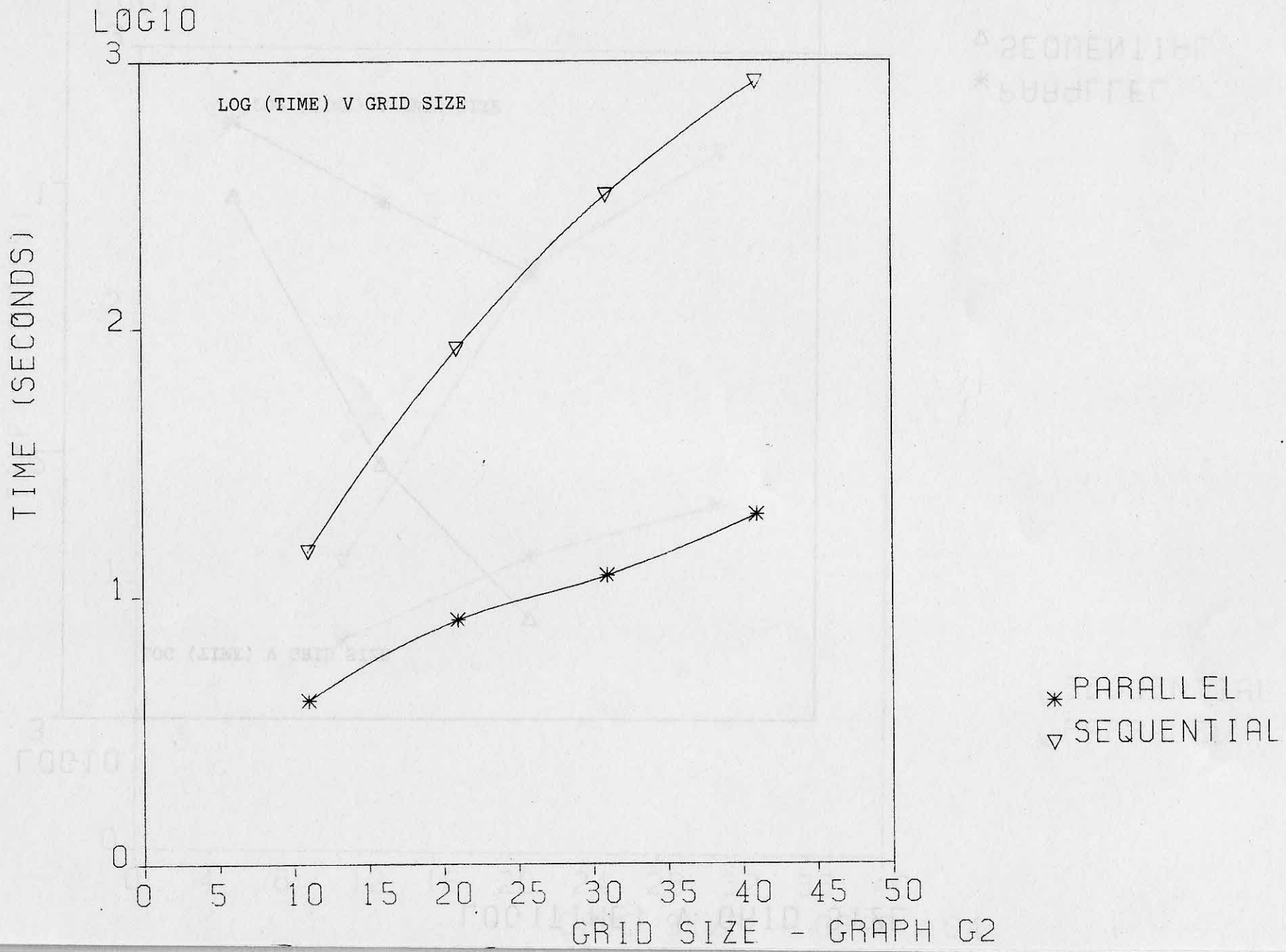
Notes

(i) For graph 7 the sequential data is not included since the program failed to converge after 2N (and even after 4N) iterations, this would lead to a false curve on the graph implying it had converged.

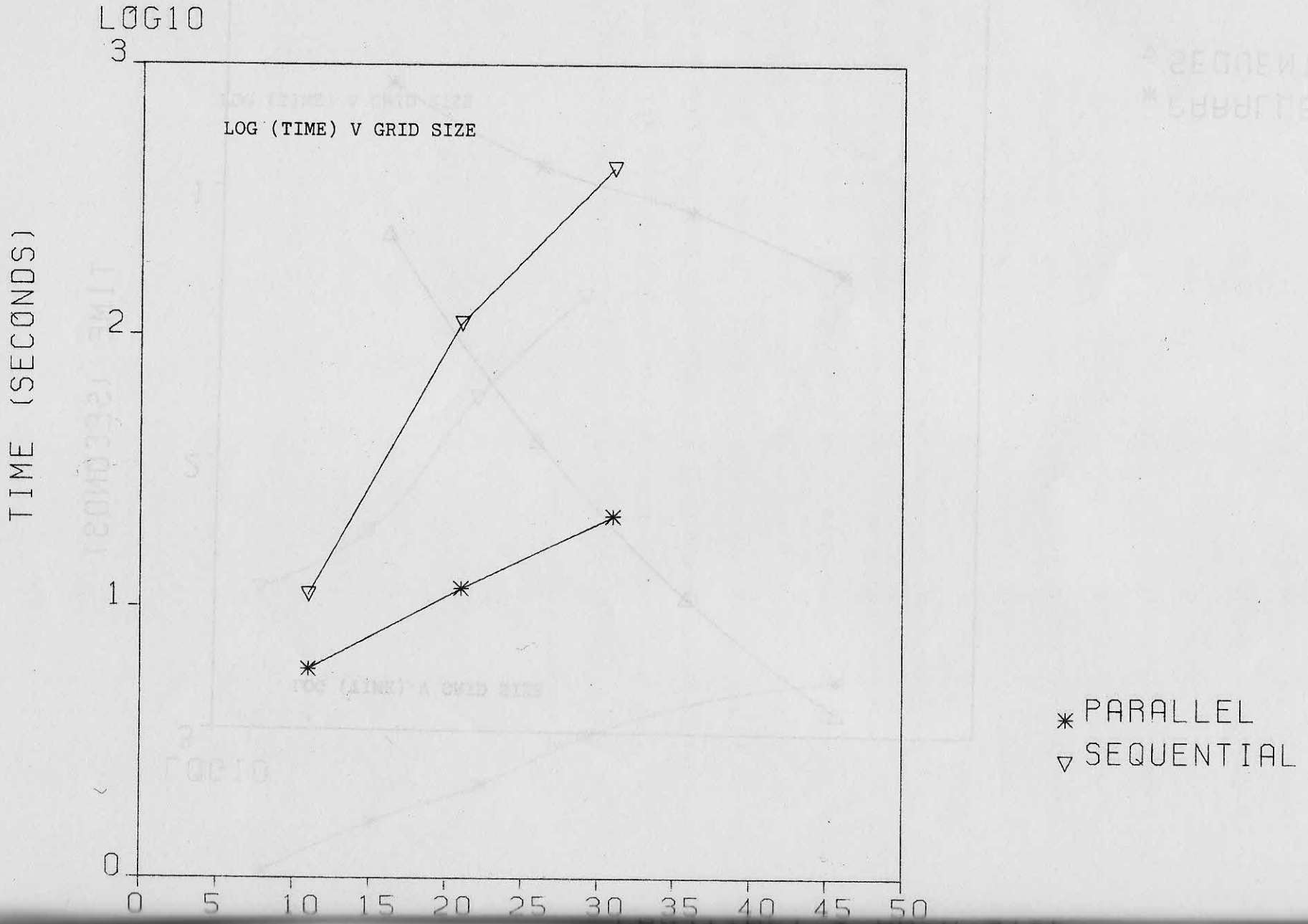
(ii) A few of the tests had to be run with a reduced terminating tolerance as the initial tolerance was too small. If the other tests were run with this tolerance it is anticipated that they would have terminated with fewer iterations and the time would be slightly less. This would, however, with a y-axis of the LOG of time make a negligible difference to the curves.

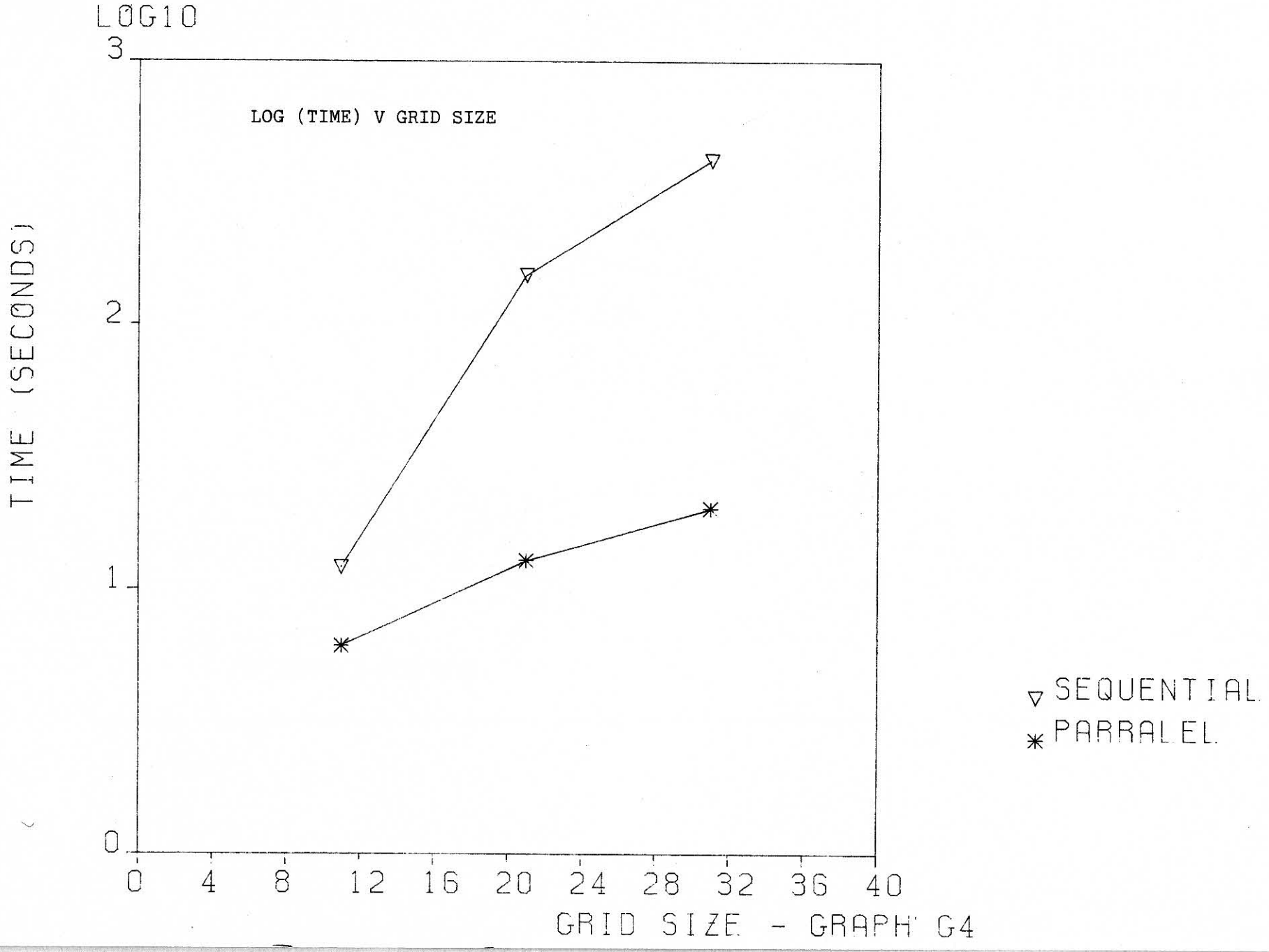
LOG(TIME) V GRID SIZE

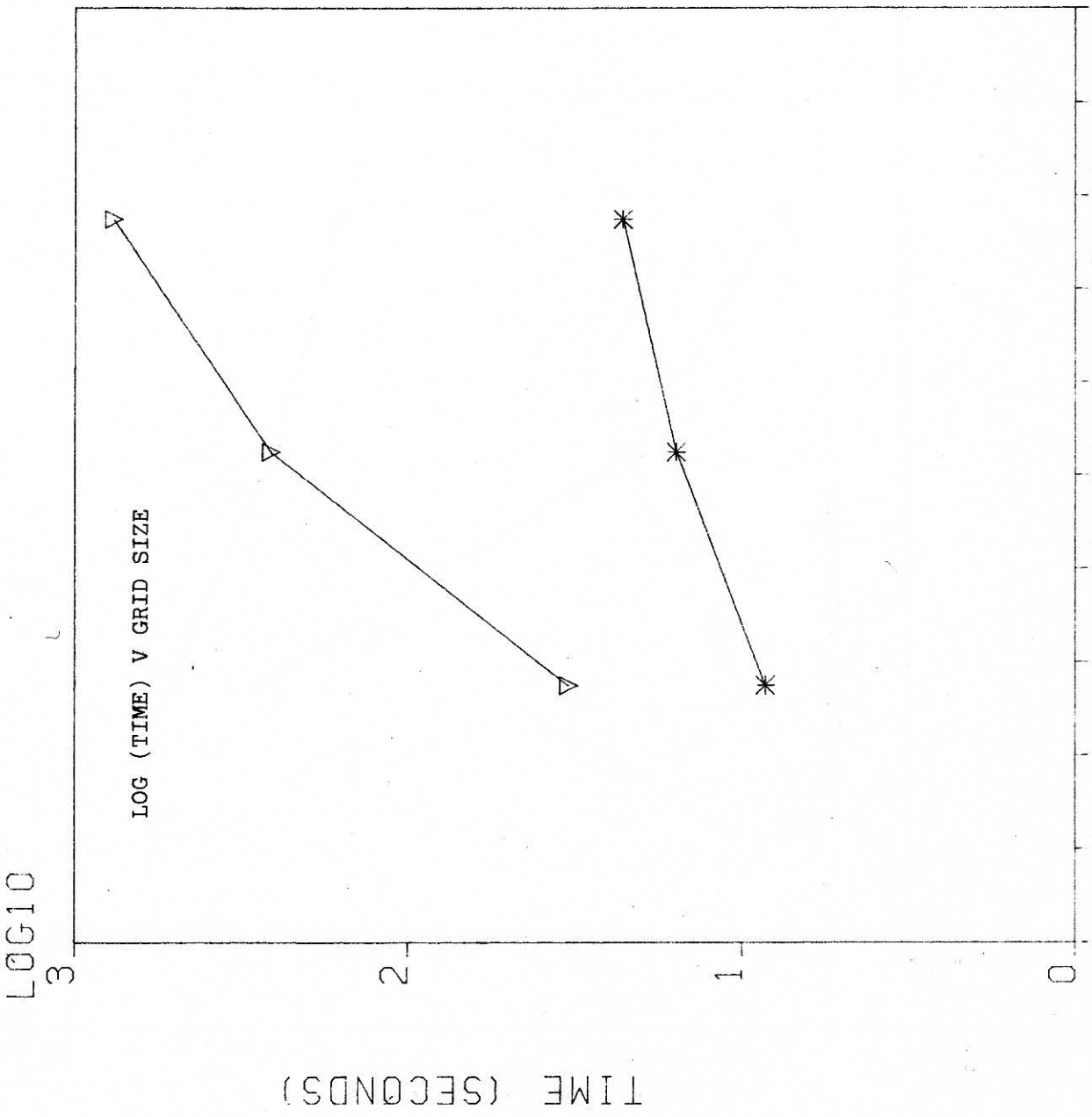


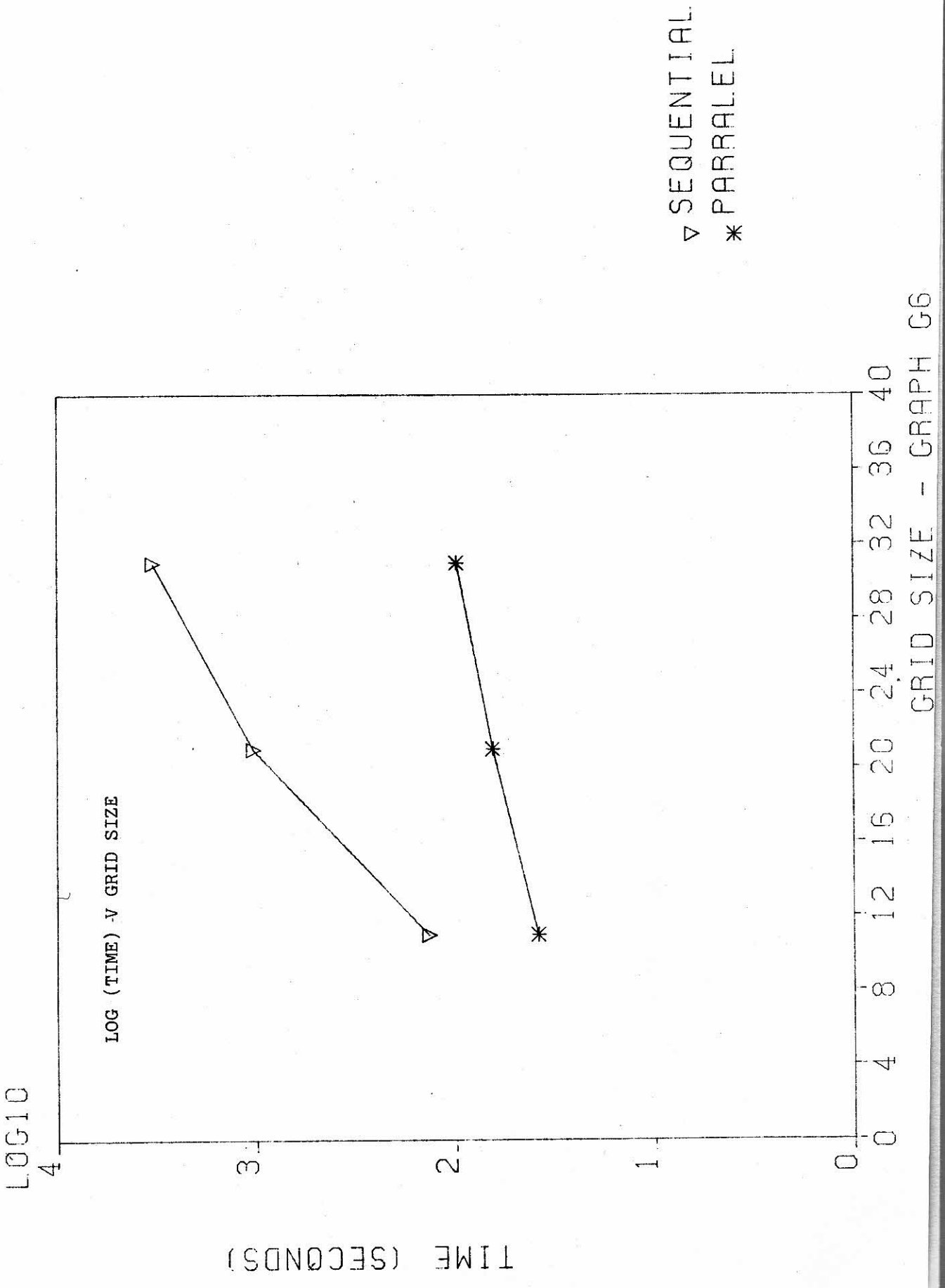

LOG10
LOG (TIME) V GRID SIZE
TIME (SECONDS)
3
2
1
0
0 5 10 15 20 25 30 35 40 45 50
GRID SIZE - GRAPH G2
PARALLEL
SEQUENTIAL

LØG(TIME) V GRID SIZE




LOG10
3
2
1
0
LOG (TIME) V GRID SIZE
TIME (SECONDS)
0 4 8 12 16 20 24 28 32 36 40
GRID SIZE - GRAPH G4
SEQUENTIAL
PARRALEL


LOG(TIME) V GRID SIZE
LOG10
3
2
1
0
TIME (SECONDS)
LOG (TIME) V GRID SIZE
▽ SEQUENTIAL
✳ PARRALEL


LOG10
4
3
2
1
0
TIME (SECONDS)
LOG (TIME) -V GRID SIZE
0 4 8 12 16 20 24 28 32 36 40
GRID SIZE - GRAPH G6
▽ SEQUENTIAL
* PARRALEL

LOG(TIME) V GRID SIZE