Parallel Computation Schemes for Dynamic Relaxation

B.H.V. Topping† and A.I. Khan‡
†*Professor of Structural Engineering*
‡*Lecturer in Computing and Information Technology*
Heriot-Watt University,
Riccarton,
Edinburgh, EH14 4AS,
United Kingdom

Abstract This paper describes a parallel algorithm for Dynamic Relaxation Method. The basic theory of the Dynamic Relaxation is presented in brief to prepare the reader for the parallel implementation of the algorithm. Some fundamental parallel processing schemes have been explored for the implementation of the algorithm. *"Geometric Parallelism"* was found suitable for the DR problem on transputer-based syetems. The evolution of the algorithm is given by identifying the steps which could be executed in parallel. The structure of the parallel code is discussed and then described algorithmically. Two geometrically non-linear parallel finite element analysis have been performed using different mesh densities. The number of processors was varied to investigate algorithm efficiency and *"speed ups"*. From the results obtained it is shown that the compuational efficiency increases when the computational load per processor is increased.

1 Introduction

Dynamic Relaxation, is a step by step method for tracing the motion of a structure from the time of loading to when it reaches a postion of equilibrium due the effects of damping. Since the method of Dynamic Relaxation is for static analysis the analyst is not concerned with the motion since this is fictitious.

The Dynamic Relaxation technique was developed in 1965 by A.S. Day [14] for the finite difference analysis of concrete pressure vessels. It is a direct or vector method of nonlinear static structural analysis. The concept of the method was infact known much earlier by Rayleigh who *"shows how, by making velocities vanish, the solution of static problems may be extracted by vibration analysis"* [19].

The important features of this method when used with a finite element idealisation are:

- The method is not for dynamic analysis but uses a fictitious damped dynamic analysis to determine the solution of a static problem.
- The method does not utilise an assembled structure stiffness matrix and hence is particularly suitable for highly nonlinear problems (material non-linear).
- The method is always expressed in terms of the current coordinates of the structure hence the method automatically permits analysis large displacements (geometric non-linear)

Since the early 1970s the method has been extensively developed by M.R. Barnes [4, 5, 6, 7, 8, 9, 10, 11] for the analysis and design of tension structures. The method is particularly suitable for use with micro-computers since a stiffness matrix need not be formed and the method is termed a vector approach. The method of Dynamic Relaxation (D.R.) is a direct application of Newton's Second Law:

$$F = M \cdot a = M \cdot \dot{V} \tag{1}$$

but since DR is a static method of analysis where the motion may be considered to be fictitious. The path to solution should be as rapid and efficient as possible. Fictitious masses should be assigned to each joint. However for the most rapid path to the solution different masses may be assigned in each of the three coordinate directions. It is best to use simple finite elements with dynamic relaxation rather than fewer complex elements.

Dynamic Relaxation has frequently been used for the analysis and design of cable and membrane structures. In this paper parallel schemes for this type of analysis are considered. In particular the efficiency of this non-linear finite element method using a transputer system are considered. The use of parallel finite element analysis with transputers is considered elsewhere [1] and the use of parallel adaptive mesh generator in reference [2]

2 Basic Theory of the Dynamic Relaxation Method

The method calculates the dynamic behaviour of the structure by direct application of Newton's second law which states that *"that the rate of change of momentum of a body is proportional to the applied force and takes place in the direction of the applied force".* In this case the mass of the structure is assumed to be concentrated at the joints and an additional viscous damping term which is proportional to the velocity of the joint is included in the formulation.

At any time t the out of balance or residual force in the x coordinate direction at joint i may be expressed as follows:

$$R_{ix}^t = M_{ix} \cdot \dot{V}_{ix}^t + C_i \cdot V_{ix}^t \tag{2}$$

where:

R_{ix}^t	residual force at time t in x direction at joint i
M_{ix}	mass 'fictitious' at joint i in the x direction
$C_i x$	viscous damping factor for joint i in the x direction
$V_{ix}^t, \dot{V}_{ix}^t$	are the velocity and acceleration at time t in the x direction at joint i

The viscous damping term $C_i x \cdot V_{ix}^t$ is proportional to the velocity, but, in an opposite direction.

2.1 Velocity Tracing

The analysis will trace the behaviour of the structure at a series of points in time t, $t + \Delta t$, $t + 2\Delta t$, $t + 3\Delta t$, ... etc. Over any time step, Δt, the velocity is assumed to vary linearly with time. Hence:

$$V_{ix}^t = \left(V_{ix}^{(t+\Delta t/2)} + V_{ix}^{(t-\Delta t/2)}\right)/2 \tag{3}$$

and

$$\dot{V}_{ix}^t = \left(V_{ix}^{(t+\Delta t/2)} - V_{ix}^{(t-\Delta t/2)}\right)/\Delta t \tag{4}$$

Hence equation (2) may be written as:

$$R_{ix}^{t} = \frac{M_{ix}}{\Delta t}\left(V_{ix}^{(t+\Delta t/2)} - V_{ix}^{(t-\Delta t/2)}\right) + \frac{C_i x}{2}\left(V_{ix}^{(t+\Delta t/2)} + V_{ix}^{(t-\Delta t/2)}\right) \tag{5}$$

Hence the integration scheme is interlacing with the residuals calculated at the end of each time step and the velocities calculated at the half time step:

- Residuals are calculated at the ends of time intervals: 0, Δt, Δt, 2Δ, ..., $(t-\Delta t)$, t, $(t+\Delta t)$, ...
- Velocities are calculated at the mid points of time intervals: $\Delta t/2$, $3\Delta t/2$, $5\Delta t/2$, ..., $(t-\Delta t/2)$, $(t+\Delta t/2)$, $(t+3\Delta t/2)$, ...

Rearranging equation (5) gives:

$$V_{ix}^{(t+\Delta t/2)} = V_{ix}^{(t-\Delta t/2)}\left(\frac{M_{ix}/\Delta t - C_i x/2}{M_{ix}/\Delta t + C_i x/2}\right) + R_{ix}^{t}\left(\frac{1}{M_{ix}/\Delta t + C_i x/2}\right) \tag{6}$$

2.2 Damping Constant

A new damping constant may be defined as:

$$k = C_{ix}\left(\frac{\Delta t}{M_{ix}}\right) \tag{7}$$

Using the damping constant k for all joints indicates that the ratio of damping force per unit mass is constant—such that joints of larger mass will be more heavily damped.

Hence equation (6) may be rewritten as:

$$V_{ix}^{(t+\Delta t/2)} = V_{ix}^{(t-\Delta t/2)}\left(\frac{1-k/2}{1+k/2}\right) + \left(\frac{R_{ix}^{t}\Delta t}{1+k/2}\right)\left(\frac{1}{M_{ix}/\Delta t + C_{ix}/2}\right) \tag{8}$$

This equation is usually written as:

$$V_{ix}^{(t+\Delta t/2)} = A \cdot V_{ix}^{(t-\Delta t/2)} + B_{ix} \cdot R_{ix}^{t} \tag{9}$$

where

$A = \left(\frac{(1-k/2)}{(1+k/2)}\right)$ constant for the whole structure

$B_{ix} = \frac{\Delta t}{M_{ix}}\left(\frac{1}{(1+k/2)}\right)$ a value for each joint and coordinate direction.

2.3 Current Coordinates and Displacements

The increase in deflection of joint i in the x direction during time interval t to $(t+\Delta t)$ is given by:

$$\Delta x_i^{(t+\Delta t)} = \Delta t \cdot V_{ix}^{(t+\Delta t/2)} \tag{10}$$

Alternatively the current coordinates may be used as a measure of the displacements of the structure:

$$x_i^{(t+\Delta t)} = x_i^{t} + \Delta t \cdot V_{ix}^{(t+\Delta t/2)} \tag{11}$$

Similar equations must be written for the y and z coordinates.

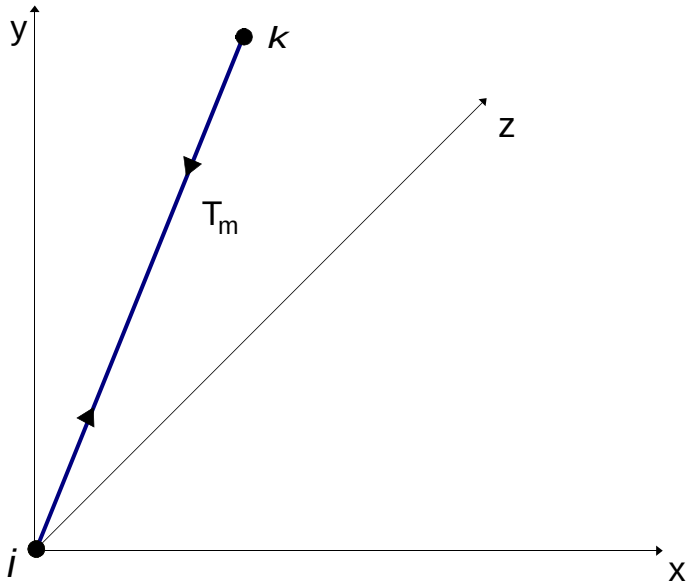

Figure 1. *Link connecting joint i to joint k*

2.4 Boundary Conditions or Supports

Boundary conditions may be imposed by assigning large masses to fixed joints.

2.5 Calculation of Residuals for Truss and Cable Structures

Once the current coordinates have been determined using (8) the new length (extended) may be calculated at time $(t + \Delta t)$. The current internal force in the link may be determined as follows:

$$T_m^{(t+\Delta t)} = \frac{EA_m}{L_m^0}\left(L_m^{(t+\Delta t)} - L_m^0\right) + T_m^0 \tag{12}$$

where:

L_m^0	initial length of bar/link
$L_m^{(t+\Delta t)}$	current length of bar/link at time $(t + \Delta t)$
EA_m	elastic modulus multiplied by the cross sectional area of the mar m
T_m^0	initial prestress in link (if any).

If the link is a cable and $T_m^{(t+\Delta t)} < 0.0$ the force is compressive so $T_m^{(t+\Delta t)}$ must be set equal to zero.

If the link connects joints i to k, then the force in the x direction at joint i from bar m is given by:

$$\Delta R_{ixm}^{(t+\Delta t)} = \frac{T_m^{(t+\Delta t)}}{L_m^{(t+\Delta t)}}\left(x_k^{(t+\Delta t)} - x_i^{(t+\Delta t)}\right) \tag{13}$$

Current geometry need only be used to calculate the components of forces if the structure is geometrically nonlinear.

$$\Delta R_{kxm}^{(t+\Delta t)} = -\Delta R_{ixm}^{(t+\Delta t)} \tag{14}$$

The contributions of each bar at each joint of the structure are summed with the applied loading F_{ix} to give the residual force at time $(t + \Delta t)$:

$$R_{ix}^{(t+\Delta t)} = F_{ix} + \sum_{\substack{\text{members } m \\ \text{at joint } i}} \Delta R_{ixm}^{(t+\Delta t)} \tag{15}$$

This type of analysis may be used to calculate the coordinate positions of cable structures subject to prestress. In this case $T_m^{t+\Delta t}$ is set equal to a prestress value and the coordinate positions of the joints calculated using the above analysis technique. This is sometimes referred to as formfinding since the geometry of the structure due to the prestress in each member and under the action of dead load is determined. The extended length of each member when the structure is under equilibrium under the prestress may be calculated as $L_m^{t+\Delta t}$ using the current coordinates of the joints at the end of the calculation. These extended lengths under the prestress may used to determine the slack or initial lengths of the members.

It is important to note that the current coordinates at time $(t + \Delta t)$ have been used to calculated the residual components. Similar equations may be written for the y and z coordinate directions.

3 The General Iteration

The general cycle of calculations is as shown in Figure 3.

The method of Dynamic Relaxation is an iterative algorithm which consists two main procedures:

- $\lhd$ Calculation of the out of balance force (called residuals) at each node of the structure. The residuals are intially equal to the applied loading unless there is some prestress in the structure.
- $\lhd$ Calculation of the nodal velocities using the previous nodal velocities, current residuals and any damping factor.

3.1 Initial Conditions

To ensure that the initial conditions $V_{ix}^0 = 0$ and $R_{ix}^0 = P_{ix}^0$ the velocity at time $\Delta t/2$ must be given by:

$$V_{ix}^{(\Delta t/2)} = \frac{B_i \cdot P_{ix}^0}{1 + A} \tag{16}$$

This may be confirmed by defining an imaginary velocity at time $-\Delta t/2$ that is equal in magnitude but of opposite sign to $V_{ix}^{(\Delta t/2)}$ such that the velocity at $t = 0$ must be zero since the velocity is assumed to vary linearly with time. Figure 3 illustrates this concept.

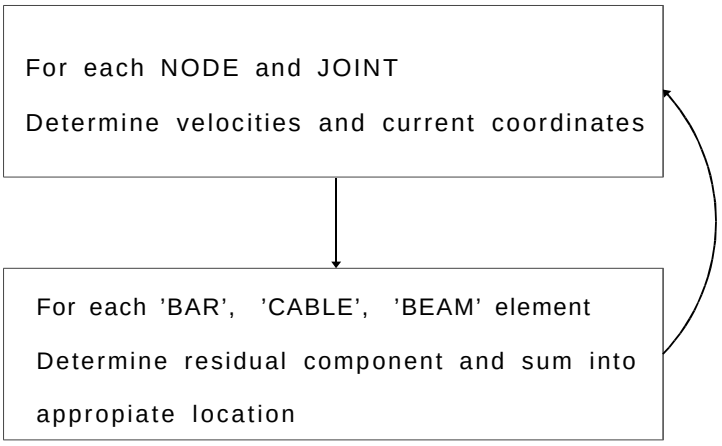

Figure 2. *General Cycle of the Dynamic Relaxation Iteration*

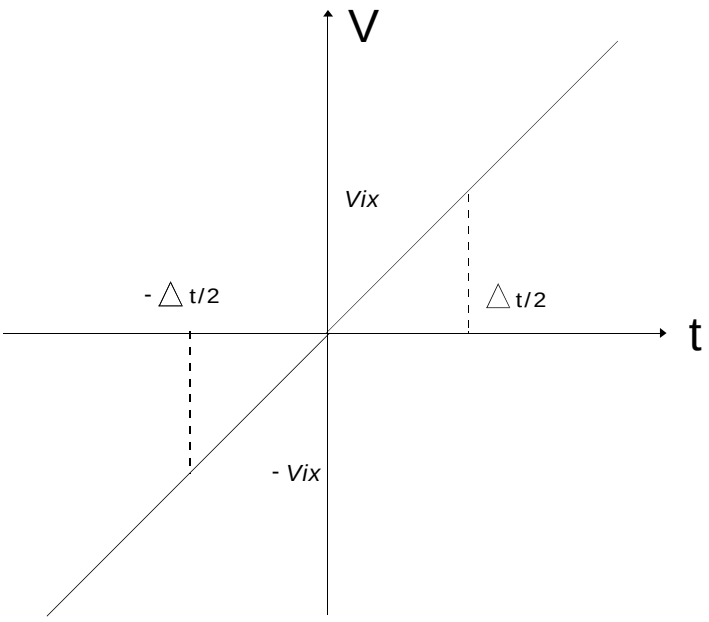

Figure 3. *Initial Conditions: velocity trace at* $t = 0$

3.2 Stability of the Analysis

The controlling paramenters for the stability of the analysis are:

- the time interval
- the nodal mass components which may be fictitious; and
- the damping factor.

3.2.1 Ficititious Masses If the time interval, Δt is too large or the masses too small then instability of the iteration may occur and the analysis will not converge to an equilibrium position. Generally convergence may be achieved by reducing the time interval or increasing the ficitious masses. It has been shown by Barnes [4] that for any Δt convergence may usually be assured by using fictitious masses defined by the following equation:

$$M_{ix} = \frac{\Delta t^2}{2}\left(\frac{S_{ix}}{2}\right) \tag{17}$$

where S_{ix} is the direct stiffness of the ith joint in the x direction

This expression is only valid when the principal stiffness directions coincide with the global coordinate system. Hence to ensure stability of the iteration the following expresion may be used:

$$M_{ix} = \Delta t^2\left(\frac{S_{ix}}{2}\right) + (\text{a term}) \tag{18}$$

Fictitious masses should be assigned to each co-ordinate direction to ensure rapid convergence to a solution.

3.2.2 Damping Factors The damping factor that causes the structure to approach the static postion most rapidly should be used for the analysis. This factor is called the critical damping factor. In figure 4 the traces for for a one degree of freedom problem with a number of different damping factors is illustrated. The sine wave represents the dynamic trace for an undamped analysis. The other traces represent three alternative paths that may be used to obtain the static solution of the problem: overdamped, underdamped and lightly damped. The overdamped trace indicates that an overdamped analysis is slow to reach convergence and since the trace does not pass through the static position the analysis does not give any bounds on the accuracy of the analysis. The lightly damped trace illustrates that an lightly damped analysis will be inefficient since the path to solution is slow with much oscillation about the static solution. The underdamped trace represents a more efficient path to the solution where the damping factor is close to the critical.

In multi-degree of freedom problems the trace will not be ideal and will be similar to that shown in figure 5. The critical damping factor may be estimated by undertaking an undamped run to obtain an estimate of the highest frequency and by using the expression derived for the critical damping factor for a one degree of freedom problem (given by Biggs [12] for example).

The critical damping factor:

$$C_{ic} = 2\sqrt{S_i \cdot M_i} \tag{19}$$

The frequency is therefore:

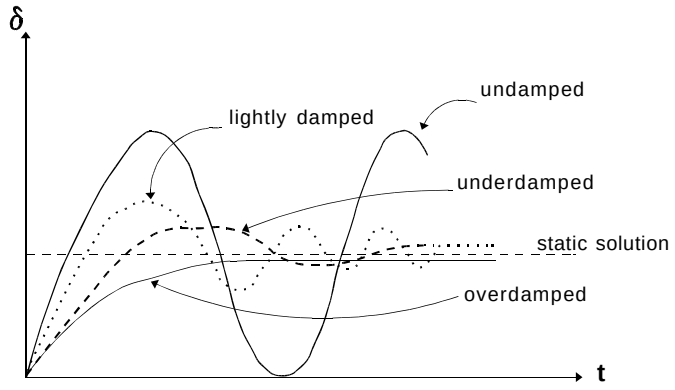

Figure 4. *One-degree of freedom time-displacement trace*

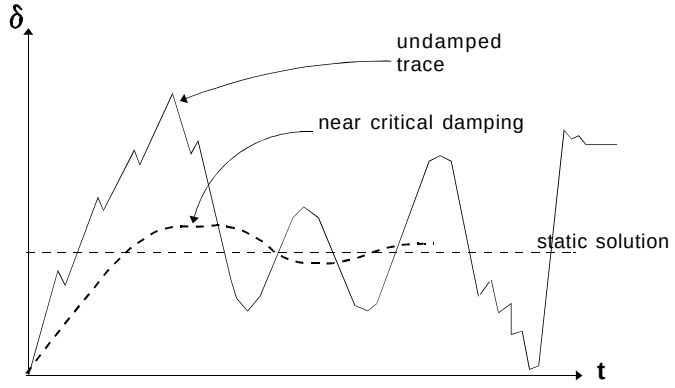

Figure 5. *Multi-Degree of freedom structure time-displacement trace*

$$f = \frac{1}{2\pi}\sqrt{\frac{S_i}{M_i}} = \frac{1}{T} \tag{20}$$

Hence

$$C_{ic} = 2\sqrt{S_i}\sqrt{M_i} = 2\pi f\sqrt{M_i}2\sqrt{M_i} = 4\pi f M_i \tag{21}$$

Hence

$$k_{\mathrm{c}} = \frac{C_{ic}\Delta t}{M_i} = \frac{4\pi f M_i \Delta t}{M_i} = 4\pi f \Delta t \tag{22}$$

The the highest frequency obtained from a multi-degree of freedom problem is then used to estimate the critical damping factor. To avoid the necessity to undertake undamped trial analysis kinetic damping may be used.

4 Kinetic Damping

There are many parameters to determine for efficient solution of any problem Δt, K and M. The number of parameters may be reduced by the use of kinetic damping which does not require the determination of a viscous damping term. Hence only the time interval and the nodal masses are required. In this way the time interval may be fixed and the masses estimated from equations 18 and increased in the case of instability. Alternatively the time interval may be reduced.

Kinetic Damping is an alternative to viscous damping, that was suggested by Cundall [13] for application to unstable rock mechanics. The method of Kinetic Damping has been found to be very stable and rapidly convergent when dealing with large displacements.

In this case no damping factor is used hence:

$$A = 1 \tag{23}$$

$$B_i = \frac{\Delta t}{M_i} \tag{24}$$

and the kinetic energy of the complete structure is traced as the undamped oscillations proceed and all current nodal velocities are reset to zero whenever an energy peak is detected. For a linear elastic system oscillating in one mode the first kinetic energy peak achieved would represent the static equilibrium position. For practical problems, however, the process must be continued through further peaks, until the required degree of convergence is achieved. Figure (6) shows the kinetic energy trace for a typical structure.

4.1 Starting Coordinates after a Kinetic Energy Reset

With kinetic energy damping the velcities of all the joints are set to zero when a fall in the level of the total kinetic energy of the structure occurs. This fall in kinetic energy indicates that a peak has been passed. A typical kinetic energy peak is shown in Figure 7. If this peak was detected by a fall in kinetic energy at time $(t + (\Delta t/2))$ then since the current coordinates are calculated at the same time as the velocities, as shown in Figure 7, the current coordinates stored in the vectors will be $x^{(t+\Delta t)}$. It has been shown that if these coordinates are adopted as the starting postion for the next cycle of calculations then full convergence may not necessarily be achieved. The coordinates are set to $x^{(t-\Delta t/2)}$ when the peak is assumed to have occurred.

Hence:

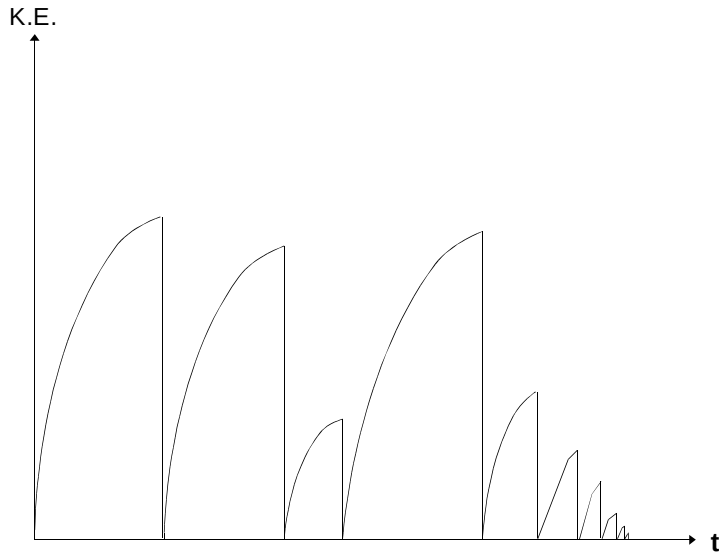

Figure 6. *Typical Kinetic Energy trace for a multi-degree freedom structure*

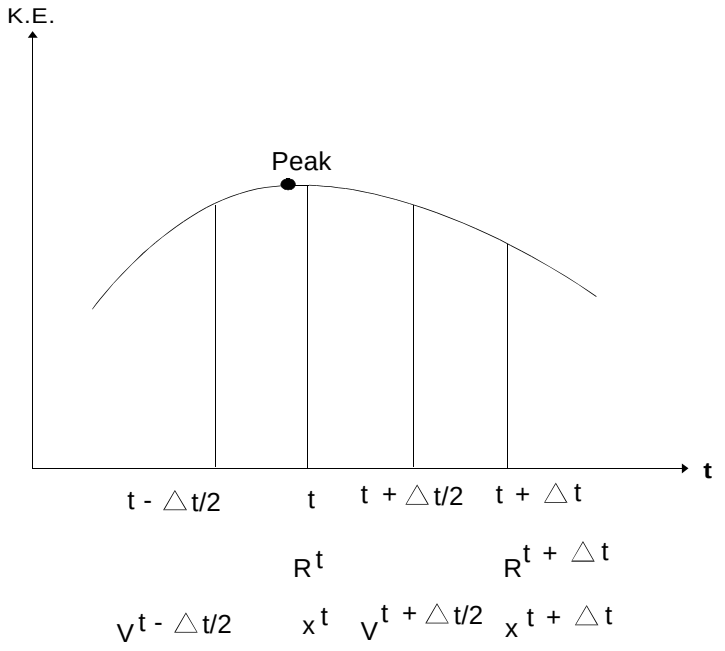

Figure 7. *The displacement trace for a typical kinetic energy peak*

$$x^{(t+\Delta t)} = x^t + \Delta t \cdot V^{(t+\Delta t/2)} \tag{25}$$

Then:

$$x^t = x^{(t+\Delta t)} - \Delta t \cdot V^{(t+\Delta t/2)} \tag{26}$$

and

$$x^{(t-\Delta t/2)} = x^{(t+\Delta t)} - \Delta t \cdot V^{(t+\Delta t/2)} - \frac{\Delta t}{2} V^{(t-\Delta t/2)} \tag{27}$$

But:

$$V^{(t-\Delta t/2)} = V^{(t+\Delta t/2)} - R^t \cdot B \tag{28}$$

Hence:

$$x^{(t-\Delta t/2)} = x^{(t+\Delta t)} - \frac{3\Delta t V^{(t+\Delta t/2)}}{2} + \frac{\Delta t^2 R^t}{2M} \tag{29}$$

When the analysis is restarted the velocities must be calculated at the mid-point of the first time step as follows:

$$V^{\Delta t/2} = \frac{\Delta t}{2M} \cdot R^0 \tag{30}$$

5 Structural Analysis

5.1 Truss and Cable Structures

The calculation of the residuals for cable and truss members has already been described in section 2.5.

5.2 Finite Element Structures

In order to keep computer storage requirements to a minimum the displacements $\{\delta^e\}$ of a typical triangular constant strain element are defined as the extensions, Δ, of the sides as shown in Figure 8. This element is referred to as the natural stiffness element and the original formulation is due to Argyris [3]. The formulation here follows the work of Barnes [4].

This formulation reduces the stiffness matrix for a constant stress triangular element to $(3x3)$ compared with the usual $(6x6)$ associated with two displacements per node. It also complies with the algorithmic procedure previously outlined for cable links. The transformation into $(9x9)$ element stiffness required for procedures in which an overall stiffness matrix is assembled is thus avoided.

The edge displacements of the element are represented by:

$$\{\Delta^e\} = \left\{ \begin{array}{c} \Delta_1 \\ \Delta_2 \\ \Delta_3 \end{array} \right\} \tag{31}$$

Corresponding element forces are the tensions along each edge:

$$\{p^e\} = \left\{ \begin{array}{c} p_1 \\ p_2 \\ p_3 \end{array} \right\} \tag{32}$$

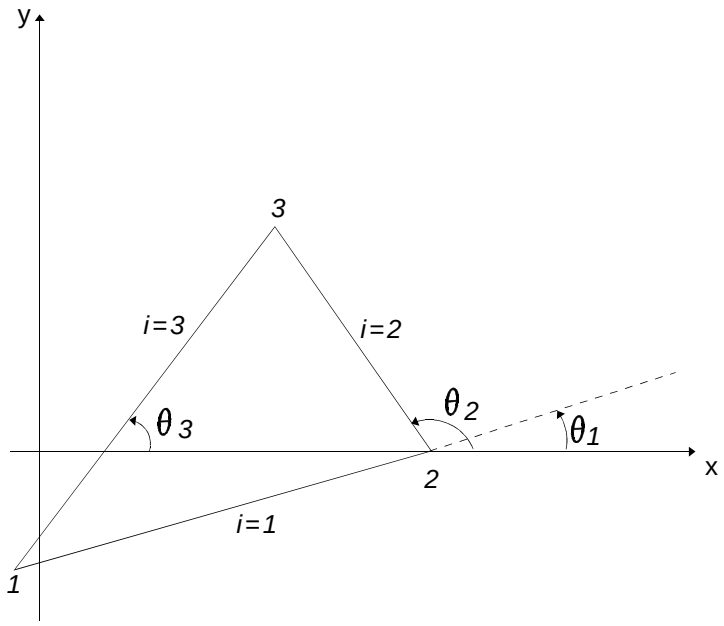

Figure 8. *Natural Stiffness Triangular Element*

The strains are assumed constant along each edge and throughout the panel and may be expressed in terms of $\{\Delta^e\}$:

$$\epsilon_i = \Delta_i / L_i \qquad\qquad i = 1, 3 \tag{33}$$

The strains ϵ_1, ϵ_2, and ϵ_3 may be expressed in terms of ϵ_x, ϵ_y and and $\tau_x y$ as follows:

$$\epsilon_1 = a_1\epsilon_x + b_1\epsilon_y - c_1\tau_{xy} \tag{34}$$
$$\epsilon_2 = a_2\epsilon_x + b_2\epsilon_y - c_2\tau_{xy} \tag{35}$$
$$\epsilon_3 = a_3\epsilon_x + b_3\epsilon_y - c_3\tau_{xy} \tag{36}$$
$$\tag{37}$$

The solution of these equations may be expressed in the usual finite element form:

$$\{\epsilon\} = \left\{ \begin{array}{c} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{array} \right\} = [G] \left\{ \begin{array}{c} \Delta_1 \\ \Delta_2 \\ \Delta_3 \end{array} \right\} \tag{38}$$

where x and y are convenient element axes. The terms in the matrix $[G]$ are given by the equation:

$$\frac{\varepsilon_x}{\left| \begin{array}{ccc} b_1 & c_1 & \varepsilon_1 \\ b_2 & c_2 & \varepsilon_2 \\ b_3 & c_3 & \varepsilon_3 \end{array} \right|} = \frac{-\varepsilon_y}{\left| \begin{array}{ccc} a_1 & c_1 & \varepsilon_1 \\ a_2 & c_2 & \varepsilon_2 \\ a_3 & c_3 & \varepsilon_3 \end{array} \right|} = \frac{\gamma_{xy}}{\left| \begin{array}{ccc} a_1 & b_1 & \varepsilon_1 \\ a_2 & b_2 & \varepsilon_2 \\ a_3 & b_3 & \varepsilon_3 \end{array} \right|} = \frac{1}{\left| \begin{array}{ccc} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{array} \right|} \tag{39}$$

and

$$\begin{aligned} \epsilon_i &= \Delta_i / L_i \\ a_i &= Sin^2\theta_i \\ b_i &= Cos^2\theta_i \\ c_i &= Sin\theta_i.Cos\theta_i \end{aligned} \tag{40}$$

and θ_i = inclination of the edge (side) i to the local x axis.
The stresses in an element are:

$$\{\sigma_{xy}\} = \left\{ \begin{array}{c} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{array} \right\} = \left[\begin{array}{ccc} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{array} \right] \left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{array} \right\} = [D]\{\varepsilon_{xy}\} \tag{41}$$

where d_{ij} are orthotropic elastic constants.
In general, the orthotropic elastic constants, d_{ij}, must correspond with the element x, y axes. But for the particular case of an isotropic plane stress element which has not buckled:

$$d_{11} = d_{22} = \frac{E}{(1-\nu^2)} \tag{42}$$

$$d_{12} = d_{21} = \nu d_{11} \tag{43}$$

$$d_{33} = \frac{E}{2(1+\nu)} \tag{44}$$

and for convenience θ_1 may be set to zero. Having obtained the $[G]$ and $[D]$ matrices the element stiffness relations are given by:

$$\{p^e\} = \left[[G]^T [D][G] t.A \right] \{\delta^e\} = [k^{ne}]\{\delta^e\} \tag{45}$$

where $t.A$ = volume of the element.
With triangular element stiffness relations in this form, panel elements may be incorporated without transformations in the basic program for cable or bar structures; the effect of panel edge forces being included in the same way as cable link forces.

For nonlinear analysis of membrane structures the effect of compressive buckling and consequent alteration of the $[D]$ matrix must be considered [4]. To account for in plane distortion of very flexible membrane elements it may be necessary to reset the $[G]$ matrix at infrequent intervals. Distortion will alter the values of θ_i in equation (39), but lengths L_i remain the unstressed lengths upon which strains must be based. Similarly the volume of the element, t.A in equation (45) will remain unchanged.

6 Parallel Dynamic Relaxation

From the theory of the Dynamic Relaxation (DR) Scheme presented in section 2 a DR scheme using constant strain triangular elements was formulated as a reference sequential model. The stiffness matrices for each element were computed using the natural stiffnesses as described in sub-section 5.2. A lumped form of mass distribution was assumed using the averaged masses of the triangular elements connected to a node as the mass for that node. The algorithm for the sequential scheme is described below:

1. Read in the input data on the topology, material properties, boundary conditions and the loading on a planar finite element domain discretized using a uniform constant strain triangular element mesh.

2. Calculate and store the original side lengths of each triangular element.

3. Calculate the nodal masses for all the nodes of the domain.

4. Form the element stiffness matrices for each element on a natural stiffness basis.

5. Assign load vectors for all the nodes of the domain. The loaded nodes have the residual vector $\mathbf{R} = applied\_loads$. All remaining nodal residual vectors are initialized to $\mathbf{0}$.

6. Initialise velocity vectors using equation 16.

7. Perform the integration for the DR method using kinetic damping.

It may be seen from the theory of DR that the nodal displacements from equation 11 are calculated independently during each time step, however the computations at the next time step cannot be readily started before the completion of preceeding time step on account of kinetic damping constraint.

From the sequential code for the DR described above it was noted that the time for the execution of step (7) described above consumed the bulk of the computational time. It was further noted that the computations for the residual vector $\mathbf{R}$ comsumed approximately 70-80% of the time spent in the integration scheme (depending upon the size of the problem). Keeping in view the above computational distribution of load in the DR algorithm following parallel algorithms were explored for determining a viable parallel computational algorithm for the DR problem.

7 A Parallel Algorithm based upon Processor Farming

The processor farming technique may be applied in cases where the computational tasks can be distributed over to the available processors with the restriction that no inter-processor communication takes place among the processors and that all the processors communicate their results directly to a pivot processor called the ROOT processor. This is also known as *Event Parallelism*.

It has been noted earlier that the calculation of the residuals accounts for the bulk of the computational load within the DR integration scheme. From equation 13 the following equation for CST elements may be determined:

$$\begin{aligned} T_{mi}^{(t+\Delta t)} &= k_{ij}^{m}\left(L_{mi}^{(t+\Delta t)} - L_{mi}^{0}\right) + T_{mi}^{0} \\ i &= 1,\ldots,3 \\ j &= 1,\ldots,3 \end{aligned} \tag{46}$$

where:

L_{mi}^{0}	initial side length of the CST element m
$L_{mi}^{(t+\Delta t)}$	current length of the CST element m at time $(t+\Delta t)$
k_{ij}^{m}	stiffness term from the natural stiffness matrix for the CST element m
T_{mi}^{0}	initial prestress in the element sides (if any).

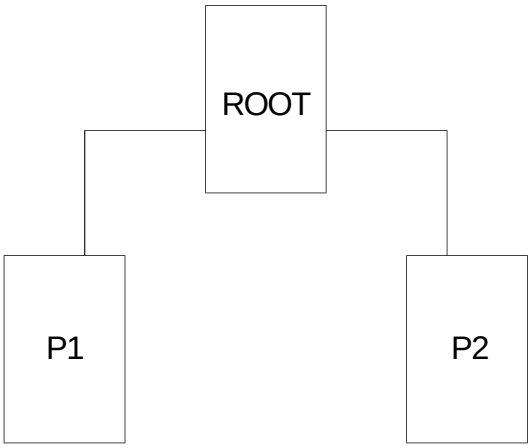

Figure 9. *ROOT + two processors*

No.	Topology	$t_{residuals}$ (min)	$t_{integration}$ (min)
1	Sequential Code	.817	0.95
2	Flood-fill code on ROOT	1.267	1.483
3	Configured code on ROOT	1.383	1.567
4	Configured code in Fig. 9 Root & two processors	0.8	1.0
5	Configured code in Fig. 10 Hypercube with root & eight processors	0.983	1.133
6	Flood-fill code in Fig. 11 Root & nine processors in pipeline	0.917	1.1

Table 1. *Computational times for the Procesor Farming scheme when analysising a uniformly graded mesh of 98 elements for 1000 iterations*

It may be seen that the residuals of each element are a function of current coordinates and the the element stiffness matrix therefore the equations for calculating the element residuals are completely decoupled. Hence residuals for each element may be computed independently in a processor farming environment. For this purpose flood-fill enviroment of the 3L Parallel C [45] was used and the algorithm was tested on upto ten processors. Pipeline and Hypercube topologies were investigated.

No effective computational efficiency was noted owing to fact that the overall communication time remained high as compared with the computational task of calculating residuals in parallel plus the

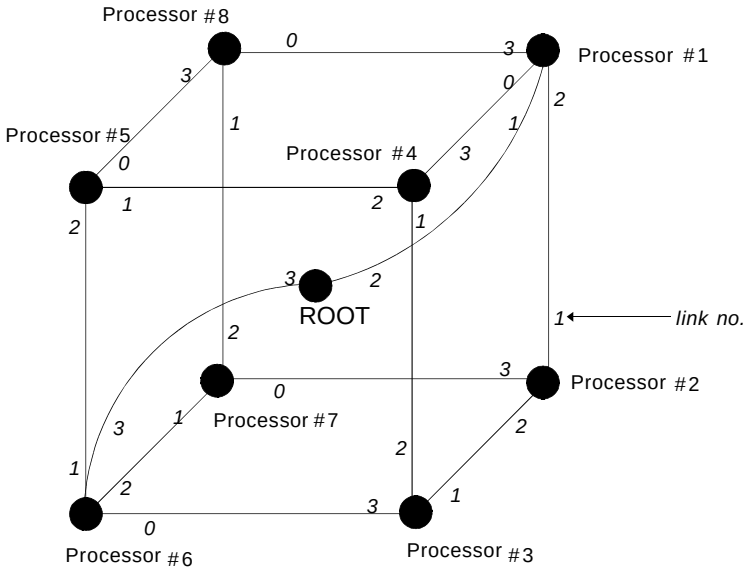

Figure 10. *ROOT + eight processors in a hypercube configuration*

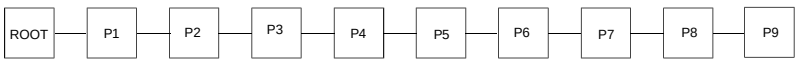

Figure 11. *ROOT + nine processors*

overheads induced by the parallel code. No computational efficiency was noted for the processor farming scheme for it to be recommended for use on transputer based systems however the data on the computational times of the scheme for various topolgies is presented for reference in Table 1. The processor farming environment in 3L Parallel C is handled by the flood-fill configurer. A router task is provided which carries the messages from the ROOT transputer to the worker transputers and vice versa. This router works independently of the topology of the network and flood-fills any valid network.

This router may also be used in configured applications by mapping each router task over a fixed topology of network in the configured environment of the 3L Parallel C compiler. Since the configured environment permits the mapping of tasks to the slave processors, the configurer permits the user to limit or specify the processers that are used in the processor farming.

From the Table 1 items 2 and 3 it may be seen that the router task provided within 3L Parallel C compiler is demonstrated to be more efficient in the flood-fill mode than the configured mode. It can also be seen that the sequential code executes faster than the parallel code running on a single transputer. This is primarily due to the extra coding required in the parallel code for forming the information packets, routing these packets to the other processes and compiling the information packets received from the other processes.

This flood fill router task provides a convenient tool for broadcasting data to the worker processors in different configured applications without resorting to custom made routers which generally work for fixed topologies only. Other routers for configured applications permit interprocessor communication and hence may be more versatile and hence efficient.

8 Algorithmic Parallelism

Algorithmic parallelism occurs when a data set is operated upon by different segments of the code while passing through a pipeline of processors. For parallelism of this nature to be efficient it is neccessary that the data flow pipe line is:

(a) sufficiently long to allow for a number of processors to be pipelined; and

(b) the computational load over the processors in the pipeline is sufficient and is uniformly distributed.

The integration scheme of the DR method executes in the following manner:

1. Add the applied loads to the current residuals **R**.

2. Calculate and add to the residuals **R** the forces induced due to change in geometry of the mesh by working out the differences between the current and the original nodal coordinates.

3. Calculate the new nodal velocities and nodal coordinates from equations 9 and 11.

4. Calculate the total kinetic energy of the system.

5. If the current kinetic energy is less than the specified tolerence value; assume convergence and stop.

6. If the current kinetic energy is less than the kinetic energy calculated in the previous iteration then perform kinetic damping as described in section 4).

Boundary segment no.	First node	Second node
1	1	2
2	2	5
3	5	6
4	6	1

Table 2. *Boundary segment representation for subdomain 1 of Figure 12*

7. Goto step (1).

As mentioned previously the step (2) for the calculation of the nodal residuals accounts for 70-80% of the time taken by the integration scheme. Hence it appears improbable to obtain computational efficiency from algorithmic parallelism owing to the fact that the number of operations within the data-pipeline are few and one of the operation consumes 70-80% of the total computational time which may lead to un-balanced distribution of computational load within the pipeline.

9 A Dynamic Relaxation Scheme based upon Geometric Parallelism

Geometric parallelism may be defined as the process of deriving computational advantage by solving sub-regions of the system on a distributed array of processors with the neighbouring sub-regions being mapped on to the neighbouring processors thus reducing the boundary communication overheads.

The DR scheme, based on geometric parallelism exhibited encouraging results. The implementation of the parallel geometric scheme for the Dynamic Relaxation was undertaken in two stages.

In the first stage the overall mesh was divided into different number of subdomains corresponding to the number of available processors. This process of domain decomposition was carried out in the most straightforward manner by assuming square or rectangular linearly connected (each subdomain sharing a maximum of two edges with other subdomains) subdomains. Each of these subdomains contained within itself geometrically identical meshes. Thus computational load balancing was assumed within each processor and from the use of linearly connected subdomains communication scheme for a vector of processors was found to be sufficient to test the algorithm. The general decomposition problem has been considered by the authors elsewhere [22, 23].

In the second stage the controlling boundary data was identified and exchanged. Kinetic damping was synchronized over the subdomains and at convergence the results from each subdomain were received and compiled to obtain the overall results for the domain.

9.1 Partitioning of finite element mesh into subdomains

The overall data structure for a domain may be divided into following categories; the number and the coordinates of the nodes, the number of elements and the connectivities of the nodes to specify mesh elements, material properties, boundary conditions at the restrained nodes and the applied loading at certain nodes. The process of partitioning of the mesh is to extract and recompile the data structure for each subdomain on the same format but having a lesser or at the most equal (in the case when only one subdomain is formed) number of data entries for the above categories.

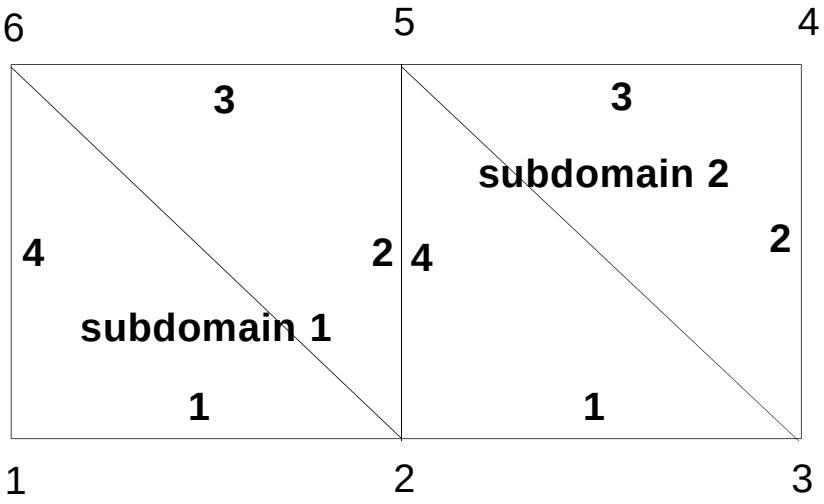

Figure 12. *A finite element mesh to be divided into two subdomains*

The boundary nodes for each subdomain are specified in a counter-clockwise manner as shown in the table 2 for subdomain 1 of Figure 12. The information on the total number of subdomains and the number of segments in the subdomain is also specified.

For the formation of the data structure of the subdomain each element of the overall finite element mesh has to be checked whether the same is located within the boundaries of the subdomain. This is implemented by the method proposed by Peparata and Shamos in reference [29] for a point-location problem in convex domain. The algorithm is described below.

1. For the subdomain calculate the vector **xq** where $xq(0)$ is the mean of the x-coordinate of the boundary nodes of the subdomain and $xq(1)$ is the mean of y-coordinates. For each triangular element of the overall mesh a vector of coordinates **xm** is calculated at the center.

2. It is assumed that the boundary nodes of the subdomain under consideration are connected to **xq** dividing the convex subdomain into finite number of triangular wedges.

3. For each triangular wedge the area coordinates **L** of **xm** are calculated and if following inequality is satisfied by the element then the element is considered to be lying within the subdomain.

$$0 \leq \mathbf{L} \leq 1.0 \tag{47}$$

All the elements thus located within the subdomain are regrouped under the data structure of that particular subdomain.

This research was commenced by studying the sequential code for the DR Scheme wherein the steps (2) to (6) for the DR algorithm in section 6 were computed for the whole domain. Various parameters such as the nodal masses and the stiffness matrix were already available in the code for

the whole mesh. These values were simply assigned to each element of the subdomain in the process of the compilation of the data structure for each subdomain. Although this saved some programming effort for the sake of computational efficiency it was an un-neccessary exercise to compute the DR parameters for the whole mesh and then to assign them to the relevant subdomains. Ideally all these parameters should have been calculated at the time of identification of elements for each subdomain.

9.2 Identification of the boundary data

A DR problem decomposed into a number of smaller sub-problems requires that the following conditions are fulfilled:

1. As the subdomains are processed in parallel it is necessary to ensure that the solution within each subdomain remains compatible with the adjacent subdomain at all times . If this condition is not met with then the compatibility conditions cannot be satisfied at the boundaries of the subdomains.

2. The path to the solution is strictly governed by kinetic energy levels computed after each iteration in the integration scheme. It is thus mandatory that all the subdomains have the kinetic energies in exactly the right proportion which when added at any given instant give the true kinetic energy level for the whole domain.

3. The kinetic energy value for the whole domain should be available within each subdomain such that kinetic damping is synchronized among all the subdomains.

In view of the above constraints it is neccessary that some information on the state of the subdomains is exchanged by the boundary nodes of the interconnecting subdomains. From the theory presented in section 2 it may be seen that the current nodal coordinates and the kinetic energy of the domain are implicitly the functions of the residual vector **R**.

If a domain is partitioned into non-overlapping subdomains then from equation 15 it may be readily inferred that the residual vectors **R** of the boundary nodes cannot be independently calculated within the subdomain. Therefore while computing residual vectors for the boundary nodes information is required from the residual vectors of the adjacent subdomains to satisfy conditions (1) and (2) stated above. The kinetic energy is defined as a quadratic function of the velocity. The velocity in the DR problem is in turn a linear function of **R**. Thus the share of the kinetic energy, depends upon the contribution to the vector **R** from the connecting subdomains, for each of the boundary nodes has to be determined in order to calculate the kinetic energy of the subdomain. However as we are only interested in knowing the total K.E. level of the domain after each time step therefore the the kinetic energy contribution at the boundary nodes may be done in equal proportions depending upon the number of connecting subdomains with the boundary node under reference. This is shown below as.

$$K.E_{subdomain} = \frac{1}{\gamma}\sum_{i=1}^{\beta}\sum_{j=1}^{3} m_{ij}.v_{ij} + \sum_{i=1}^{nn-\beta}\sum_{j=1}^{3} m_{ij}.v_{ij} \tag{48}$$

Where β is the number of boundary nodes in the subdomain, γ is equal to the total number of subdomains sharing the boundary node(i), nn is the total number of nodes in the subdomain, **m** and **v** are the nodal masses and velocities in x, y and z directions for the boundary node(i).

Adjacent subdomain	boundary node(1)	...	boundary node(n)
1	5	...	10
⋮	⋮	⋮	⋮
m	14	...	20

Table 3. *Data structure of the subdomains adjacent to the sub-domain under consideration*

Boundary node	1st adjacent subdomain	...	mth adjacent subdomain
1	2	...	5
⋮	⋮	⋮	⋮
node(n)	3	...	-

Table 4. *Data structure with reference to the boundary nodes of the subdomain*

9.3 Arrangement of the boundary node data

A table of boundary node connectivities with the adjacent subdomains was required to be set up in order to pass the information between relevent subdomains efficiently. The data structure could be set up either with respect to the boundary nodes or the boundary subdomains.

The data structure with respect to adjacent subdomains identifies common boundary nodes of the adjacent subdomains and subdomain under consideration. The first row of the Table 3 shows the first adjacent subdomain 1 to the domain under consideration which has common boundary nodes 5, ..., 10 of the subdomain. The number of columns in this data structure may vary from row to row and global node numbers are always used.

The data structure with respect to boundary nodes identifies the relevant adjacent subdomain or subdomains sharing the same boundary node under consideration. The first row of the table 4 shows a boundary node 1, of the subdomain being considered, which is also shared by subdomains 2,..., 5. Similarly with this data structure the number of columns may vary from row to row.

The above schemes differ on account of the fact that the first data structuring scheme requires that a global node numbering of the nodes is available within each processsor so that the boundary node data received from an adjacent subdomain is sorted correctly. If this data structuring scheme was employed it would have increased the byte contents of the messages and would also have involved searching for matching the local node numbers with the global node numbers.

The second scheme was used in conjunction with the sorting and re-arranging of the boundary node data at the time of partitioning of the overall mesh and this completely eliminated the necessity of having global node numbering available within the subdomains.

For the second scheme, at the time of partitioning of the data for each subdomain, the boundary nodes are identified and sorted in ascending order using the global node numbers. The global node numbering is readily available at this stage since the data partitioning is done as a pre-process on the ROOT transputer. The boundary nodes are placed in the beginning of the data structure for the nodes. Thus the node numbering for the subdomain data starts with the boundary nodes first

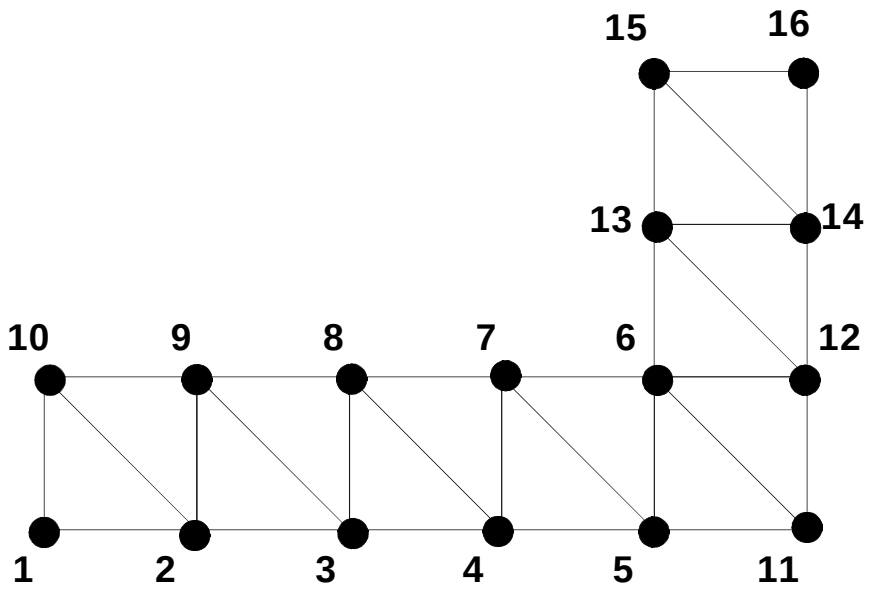

Figure 13. *A L-shaped domain with 16 nodes and 14 elements*

Order of the nodes	subdomain 1 Global nodes	subdomain 2 Global nodes	subdomain 3 Global nodes
1	2	2	3
2	9	3	4
3	-	8	7
4	-	9	8

Table 5. *The re-arrangement of nodes for the first three subdomains in Fig. 13*

with node numbers 1 to β, where β is the total number of boundary nodes for the subdomain under consideration. The element topology and other related data structures are formed accordingly, for the subdomain.

The finite element discretization shown in Fig. 13 is then divided into seven subdomains, as shown in Figure 14. This results in the arrangement of the nodes shown in table 5 for the subdomains 1 to 3. The global node numbers for each sub-domain are first sorted such that the boundary nodes are listed first in ascending order followed by other nodes in no particular order.

9.4 The Master, Worker and Router Tasks

The parallel scheme for the Dynamic Relaxation was written in the form of Master, Worker and Router Tasks. The flow charts for the Master and the Worker Tasks are represented in Figures 15 and 16.

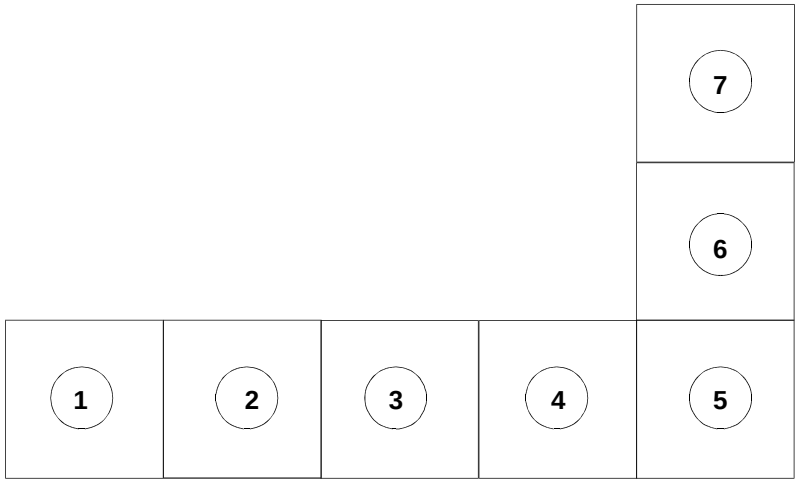

Figure 14. *L-shaped domain of Figure 13 divided into 7 subdomains*

Figure 17 shows the mapping of these task on a one dimensional array of transputers.

9.4.1 Master Task The Master task as shown in Fig. 15 executes the steps (1) to (6) as described in section 6 and then performs the data partitioning. As soon as the data for a subdomain becomes available it is sent through the Router task to the relevant slave processor. Each slave processor receives a data structure number, ie subdomain number, which is equal to the processor number in the processor array, this is achieved through the Router task. The number of subdomains used to decompose the structure will always be kept equal to the number of available processors. Since any proliferation in the number of subdomains will result in an increase in the process comunication without an increase in the available processor time. If the processor array comprises np processors connected in a pipeline the ROOT is numbered as 1 and the last processor in the pipeline is numbered np. The data structure for the subdomain 1 is kept by the Worker task on the ROOT processor which has the lowest number in the processor array, processor 2 receives the data for subdomain 2 and so on for all the data structures of the subdomains. Hence the subdomain data is always distributed among all the available processors. Every Worker task waits for the synchronization signal from the Master task to commence working. Initially all Worker tasks were enabled to send their values of the global kinetic energy to the Master task. Initially, in order to prevent a Worker from attempting to communicate with the Master task while the Master task is busy distributing the data, the Workers were made to wait for a signal from the Master task.

When the synchronization signal is sent by the Master task a flag is set up in the message. If the flag value is zero then the Router receives messages only from the Master and ignores other channels of communication i.e. from the resident Worker task or a Router task from below. When the Master sends out the synchronization signal for the last processor in the pipeline this flag is set to unity which enables the Routers to start receiving messages from the Worker tasks. It should be pointed

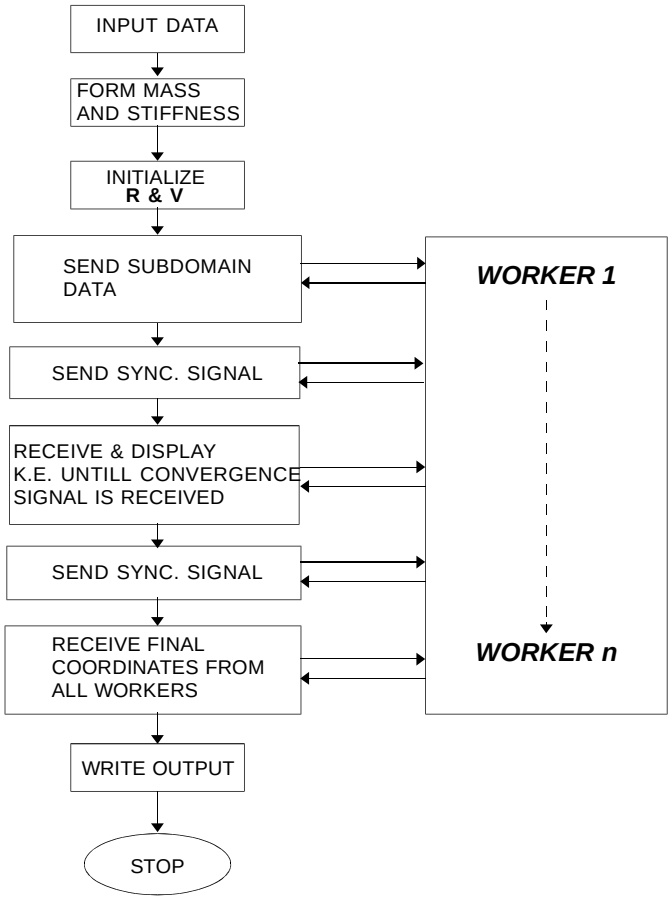

Figure 15. *The Master Task running on the ROOT transputer*

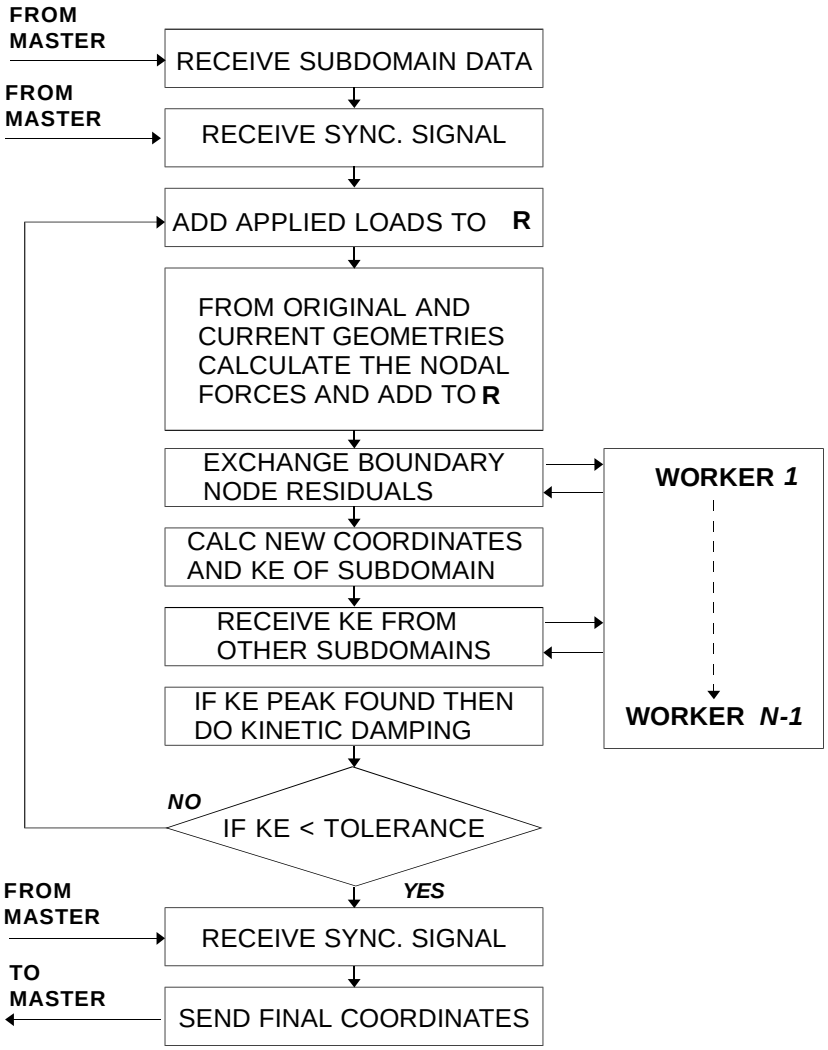

Figure 16. *The Worker Task copied on each slave transputer*

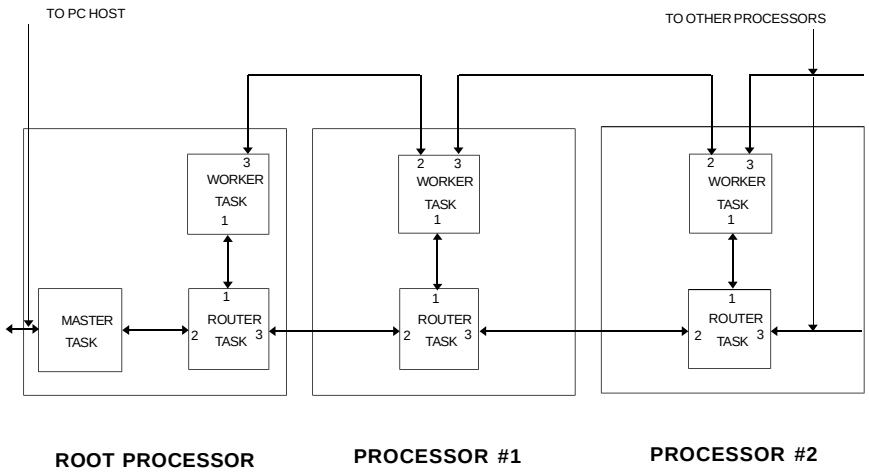

Figure 17. *The Network Configuration and the mapping of the tasks*

out that this technique only works in cases where the last message is meant for the last processor in the processor array so that the flag values for all the intermediate Routers are set equal to unity when the message is copied from one Router to another in the pipeline.

After each iteration each worker broadcasts its kinetic energy level. Hence the total kinetic energy of the complete domain (global kinetic energy) becomes available on each processor. Hence any worker may pass to the master: the global kinetic energy; the iteration number; and a flag to indicate when a peak kinetic energy is encountered. In the scheme implemented here the global kinetic energy was passed from the worker task on the root to the master task.

The Master task is informed if the global kinetic energy level after any iteration falls below the tolerance value. The Workers then wait for a synchronization signal from the Master before sending in their nodal coordinates for the respective subdomains to the Master. The synchronization is necessary to prevent a Worker from sending the final coordinates while the Master has not received the intimation of convergence and accidently receives the wrong message.

The Master task after sending out the synchronization signal to the Worker tasks then receives and compiles the final nodal coordinates of the subdomains, from the Worker tasks.

9.4.2 Worker Task The Worker Task represented in Fig. 16 receives, in addition to the data for a single subdomain, its subdomain number, the total number of subdomains in the processor array, the total number of boundary nodes and their connectivities with neighbouring subdomains stored using the second storage scheme described in section 9.3. The Worker task is provided with three I/O channels the first channel is connected to the Router task. The second channel utilizes a physical link and directly connects to the Worker task in the upper processor. The upper processor will always be the processor with a subdomain number less than the processor under consideration. The third channel is connected in the similar way to the Worker task in lower processor, as shown in Fig. 17.

The Worker receives and sends data to the Master task at the ROOT processor through the Router task using its first channel. The second and third channels are used by the Worker to exchange the data consisting the residual vectors **R** for boundary nodes and the kinetic energy levels from the other Workers.

The Worker task executes the DR integration scheme described in section 7 on the data structure of the subdomain received by it. The Worker task normally communicates twice with the neighbouring processor during each iteration. However if a kinetic energy peak is encountered then the number of communications become three for that iteration. The first communication occurs when the function for the computation of the boundary node residuals for the subdomain is called. The second series of communications occurs when the kinetic energy of the subdomains is broadcast.

The residuals are calculated twice in an iteration in which a kinetic energy peak is encounterd, hence the number of communication points increase to three per iteration in such cases. When the kinetic energy level within the subdomain falls belows the tolerance value then the Worker task on the ROOT processor sends out a message telling the Master task that all the Workers have converged. This is by virtue of complaince of condition (3) in section 9.2 under which kinetic energy convergence within any arbitrarily chosen subdomain guarantees that the same condition will have occured on all the other processors. The Worker then awaits the synchronization signal from the Master task prior to sending its nodal coordinates to the Master task.

9.5 Router Task

The Router Task is provided with three I/O channels. The Router task waits on all the three channels simultaneously using the 3L Parallel C function called `alt_wait_vec()`. This function allows a task to await without extensively claiming processor time on a vector of channel addresses specified to it. Whenever a channel from the specified vector becomes ready to communicate with the task the function returns giving the address of the waiting channel and the task may then execute. The first channel is connected to the resident Worker task. The second channel is connected to the Router on the upper processor with the exception of the Router Task on the root processor which is connected to the Master Task via channel 2. The third channel is connected to the Router on the lower procesor as shown in Figure 17.

The message passing protocol for the Router task is established in two phases. For every communication packet to pass it must be preceeded by an introductory packet. The introductory packet for the Router task carries:

- the processor number (which is the same as the subdomain number of the resident Worker Task);
- the processor number of the sender task;
- the processor number for the target task;
- the length of the message in bytes of the following data packet;
- a flag called `root` is included to determine if the communication is from the Master task processor or the Worker task; and
- a flag called `control` is included to arrange for multiple packets to be sent from one task to an other task without interference from remaining tasks.

The algorithm for the Router task is given as under:

```
dynamically allocate space for a buffer of size SIZE_OF_BUFFER say 50K

loop for ever
{

wait until one of the channels has become ready to communicate

   do
   {
receive the first packet (header packet)
    and unload the values of
add1    = the subdomain number of the sender
add2    = the subdomain number of the addressee
msglen  = the size, in bytes,  of the following buffer
root    = 0 or 1
    control = 0 or 1

if SIZE_OF_BUFFER < msglen then
re-allocate space of buffer for holding the second packet

receive the second packet into the allocated space for the buffer

if add1 = add2 and root = 0 then
(root = 0 means message from the Master)
send out the buffer through the first channel to the resident
    Worker Task

if add1 = add2 and root = 1 then
 (root = 1 means message to the Master)
send out the buffer through the second channel to the upper
    Router or Master tasks

if add1 < add2 then
decrement add2 by one in the header packet and
send the header packet and the buffer through
the third channel to the lower Router task

if add1 > add2 then
decrement add1 by one in the header packet and
send the header packet and the buffer through
the second channel to the upper Router task
   }
   repeat while the value of flag ‘‘control’’ remains equal to 1
}
```

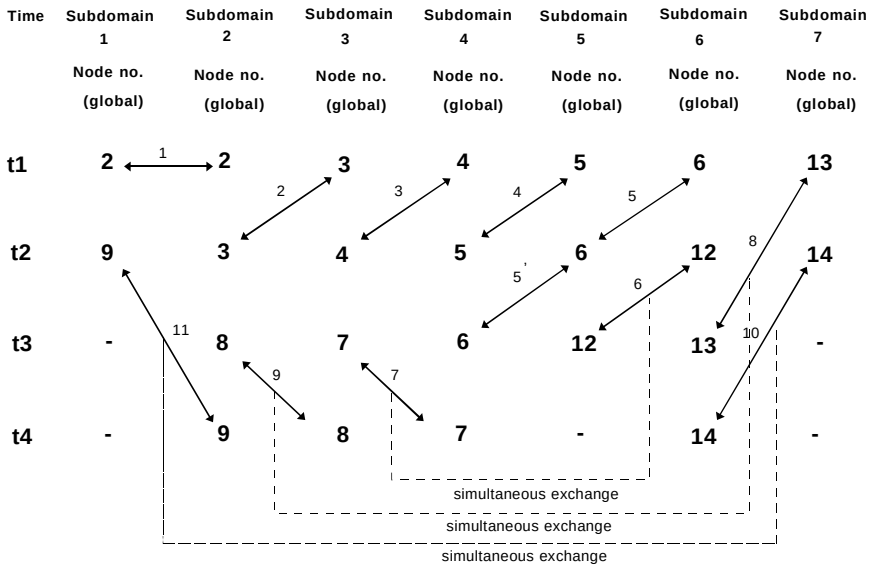

Figure 18. *Communication Algorithm for the boundary nodes*

9.6 Interprocessor communications for exchange of residuals and the kinetic energy values

9.6.1 Boundary node residual exchange The boundary node residuals must be exchanged every time the residual vectors **R** are computed within the integration scheme for the subdomain.

In the scheme presented here, the contribution to the residuals from a particular subdomain is calculated. This nodal residual due to the subdomain is communciated to all adjacent subdomains to this common node. Communication takes when the residuals for the sub-domain boundary nodes 1, ..., β are computed. Once the communication has taken place then the residuals for the remaining $1+\beta$, ..., n nodes are calculated.

The boundary nodes for each subdomain were placed at the beginning of the list of the nodes for that subdomain, at the time of the data partitioning. The boundary nodes were sorted into the ascending order using their global node numbers in the overall mesh. This exercise guarantees that at any given time there will be at least one pair of subdomains ready to exchange information. With this scheme it is possible that more one pair of subdomains would exchange boundary node information simultaneously.

The physical occurance of the bounadary node exchange is represented in Figure 18. The boundary node distribution shown in Fig. 18 is based on the mesh shown in Fig. 13. The global node numbers of the boundary noes have been written in columns for each subdomain. The communication process is illustrated by the arrows which are numbered in the order of their execution. In reality the subdomains have no information on the global node numbering and follow their local node numbering starting from 1,..., nn with nn being the total number of the nodes within the subdomain. Each subdomain processor is only aware of the total number of boundary nodes within the subdomain. Since the boundary nodes are placed at the begining of the subdomain nodal list they may be quickly identified.

The communication process for the exchange of boundary node data proceeds in the following manner:

- **Time is t_1 :** It can be seen from Fig. 18 at time t_1 boundary node information may be exchanged between subdomains one and two as both have their common boundary node (global node 2) ready to exchange boundary data. This is shown as *exchange 1* in Fig. 18. After the exchange has taken place the next node in the list in subdomain one (i.e node 9) and the next node in subdomain two (i.e. node 3), become ready to exchange data.

- **Time is t_2 :** The subdomains two and three are now ready to exchange data for node 3. This shown as *exchange 2* in Fig. 18. Node 8 of subdomain 2 and Node 7 of subdomain 3 then become ready for data exchange.

The above communication process proceeds in the same fashion for *exchanges 3 to 11* with the exception that the *exchange 5* takes place in two steps. This happen because the node 6 is commonly shared by more than two subdomains. By progressively reading the communication steps shown in the Figure 18 it may be seen that the subdomains (1,2) & (6,7), (2,3) & (6,7) and (3,4) & (5,6) exchange the boundary node data simultaneously since they become ready at the same time.

9.6.2 Algorithm for exchange of boundary node data As each boundary node is to be processed within each subdomain as a complete node and not as a node divided between two subdomains it is necessary to gather the contributions of the residual vector **R** from the connecting subdomains for that boundary node. Since global node numbers for the boundary nodes are not utilised on the subdomain processors, a protocol is required to ensure that only nodes of the same global node number exchange data.

Once the data for a boundary node on a particular subdomain processor has been calculated, the data structure is examined to determine the lowest subdomain processor which shares the boundary node.

As soon as this lowest subdomain processor has calculated the residual data for this shared node it sends the data to the next higher number subdomain processor common to the node. Once the next higher processor has recieved the residual data it adds its contribution to the residuals and sends the accumulated residuals either: to the next higher numbered subdomain connected to the node; or if the subdomain is the highest numbered subdomain connected to the node then the accumulated residual are sent back to the next lowest processor connected to the node. This is repeated until the total accumulated residuals are passed back to the lowest numbered processor connected to the node.

In each case the receiving processor must be ready to receive data form data from the sending processor. In the case where the receiveing processor is already communicating with another processor then the message will be queued awaiting attention from receiving processor.

In this way boundary node data is always initially sent from a lower numbered subdomain processor to a higher numbered subdomain processor.

This algorithm was improved by introducing a conditional check prior to the sending of the information wave which permitted the direct exchange of boundary node data for boundary nodes within only two subdomains. Once the two domains are ready to send and receive data on a particular boundary node then the exchange is as follows: the even domain number is always put in the send mode and the other proceesor is put in the receive mode; then after cumulatively adding the component of **R** received from the even processor the odd processor will send the cumulative **R** value to the node to the even processor which it copies to its data structure. This minor modification to the algorithm improved the computational speed of the parallel DR scheme generally in the order of 10%. This

improvement was due to the elimination of the requirement to send a header packet before the main message in such cases.

The algorithm for the exchange of boundary node data is given below.

```
define a header buffer similar to the one used in communicating through
the Router task but without the flags.

allocate space for the information buffer which comprises a
double precision vector with dimension being as three,
hence the information packet is 24 bytes long on the T800
architecture.

loop for i = 0, beta (no. of boundary nodes in the subdomain)
{
 for the node(i) find
 (1) ii = the total number of adjacent subdomains connected
     to this node minus one
 (2) find the max and the min subdomain numbers
     from the list of connected subdomains.

 if ii = 1 then (only two subdomains are connected to the node)
 {

  diff = adjacent subdomain no - the local subdomain no

  if diff = -1 then
  exchange the residuals using the second channel connected
  to the upper Worker task
  <skip the following instructions and take the next i>

  if diff = 1 then
  exchange the residuals using the third channel connected
  to the lower Worker task
  <skip the following instructions and take the next i>
 }
  execute the following block of instructions
  (sending vector R(i) to other subdomains)
  {
  if the subdomain number is = min then
  propogate the residual vector R(i) through the third channel,
  in the downward direction for acumalitive addition of the
  residuals

  if (min < the subdomain number is < max) then
  add to the received buffer the local value of R(i) and
  send the residual vector R(i) through the third channel, down
```

```
  if the subdomain number is = max then
  add to the received buffer the local value of R(i)
  copy the resultant value of R(i) into the local value
  of R(i) and send the buffer through the second channel
  to the upper Worker task
  }


  execute the following block of instructions
  (receiving total value of vector R(i) from other subdomains)
  {
  if the subdomain number is not equal to min then
  copy the received buffer into the location of R(i) and
  send the buffer up through the second channel

  if the subdomain is = min then
  copy the received buffer into the location of R(i)
  }
}
```

9.6.3 Algorithm for the calculation of the total kinetic energy of the domain The algorithm for calculating the total kinetic energy of the domain within each subdomain is done on the same scheme discussed in the previous section. The local value of the kinetic energy for the subdomain is calculated according to equation 48.

The kinetic energy contributions are passed along the pipeline and summed along the pipeline.

The first or the last Worker Task in the pipeline may be chosen as the pivot, in this implementation the last Worker Task has been chosen as the pivot. The Last Worker Task sends its kinetic energy value to the adjacent processsor up the pipeline. The Worker Task in this adjacent processor adds its own component of the kinetic energy to the value received and send is up again to the next Worker Task in the pipeline. This process is continued untill the first processor in pipeline is reached. The Worker Task in the first processor adds its component to the received buffer and then send the buffer down the pipeline. The total value of the kinetic energy thus gathered is copied throughout the Worker Tasks in the pipeline. The algorithm for this procedure is as given below.

```
execute the following block of instructions
{
 if the subdomain number < the total number of subdomains then
 {
  if this is not the first subdomain then
  {
  (this is for an intermediate processor in the processor pipeline)
  receive buffer of K.E from the third channel below,
  add the K.E for the subdomain to the buffer,
  send the buffer through the second channel to the upper Worker
  and goto the next block of instructions
  }
```

```
  (this is for the 1st processor in the processor pipeline)
  add the received buffer to the K.E. of the subdomain

 }
 else
 {
  (this is for the last processor in the  processor pipeline)
  send the K.E. of the subdomain through the second channel to
  the upper Worker
  }
}

execute the following block of instructions
{
 if the subdomain number = 1 then (its the 1st processor)
 send the K.E. buffer through the third channel to the
 Worker below

 if 1 < the subdomain number < total number of subdomains then
 copy the received buffer into the Total K.E variable for the
 subdomain and then send the buffer to the Worker below through
 the third channel

 if the subdomain number = the total number of the subdomains
 (meaning that its the last processor in the pipeline)
 then copy the received  buffer into the Total K.E. variable for
 the subdomain
}
```

10 Examples

Eaxmple 1 and 2 are based upon the finite element descretizations using CST element un-structured meshes. The Example 1 has been descretized, using a uniformly graded mesh, into seven subdomains and solved using three different mesh densities. The effects of changing mesh density on the computational time, communication time, *"speed-up"* and efficiency are presented.

In Example 2 the effects of shifting the same computational load over to different number of processor is shown with respect to computational time, communication time *"speed-up"* and efficiency.

The Parallel C compiler used for the purpose does not allow direct access to PC based timing utilities on processors other than the ROOT processor. Hence the on board timers avialable on each transputer were used to get an estimate of the time spent by the processors in actual computation and in communication. The transputer has two 32 bit timers. One timer is used for high priority tasks and is incremented every microsecond. The other timer is for low priority processes and is incremented every 64 microseconds. Since the Master and the Worker tasks were not assigned the *"urgent"* status hence the timer values examined and added within the Worker tasks pertained to the low priority

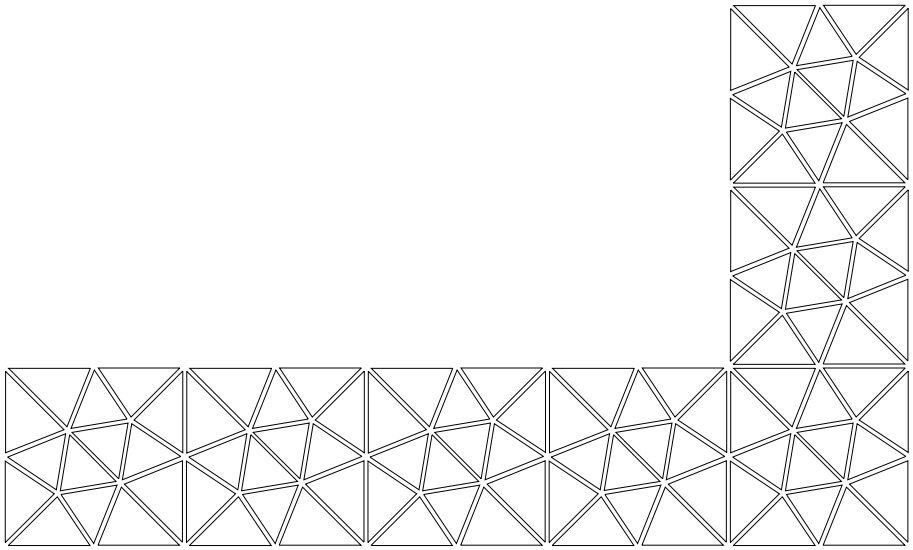

Figure 19. *The first mesh with 66 nodes and 98 elements*

timer and the time was calculated by dividing the total timer values on each processor with the low priority timer frequency i.e. 15625 ticks/second.

The processor times thus computed for each processor measured the time spent by each processor, within the integration loop, for computation and communication with the other slave processors. For the sake of calculation of *"speed-ups"* and efficiencies the times taken for the sequential code to perform the explicit time stepping integration scheme were compared with those of the parallel code performing the same procedure. The overheads for the partioning and re-ordering of data were added to the parallel times for the sake of calculation of *"speed-ups"* and efficiencies. It may be added that since the purpose of the study was to implement an effecient parallel algorithm for the DR method hence the mesh partitioning condsiderations were not accounted for in great detail. Therefore it is anticipated that the mesh partioning timing could have been reduced considerably. The mesh partitioning overheads have been added while calculating the *"speed-ups"* and effeciencies in the examples in order to obtain a conservative estimate on the efficiency of the proposed parallel algorithm.

Example 1

For the L-shaped domain shown in Fig. 19 comprising 66 nodes and 98 elements was divided into 7 subdomains as shown in Figure 20. The domain was analysed using the sequential DR code and then the parallel DR code. The time for the execution of integration scheme by the sequential code was 2.25 minutes and the time for the parallel code running on ROOT + 6 transputers came to as 0.43333 minutes. The final shape of the structure obtained under the applied loading is shown in Fig. 21.

The L-shaped domain shown in Figure. 19 was remeshed and the mesh comprising 123 nodes and 196 elements was generated, Figure 22. The sequential code and the parallel code running on ROOT

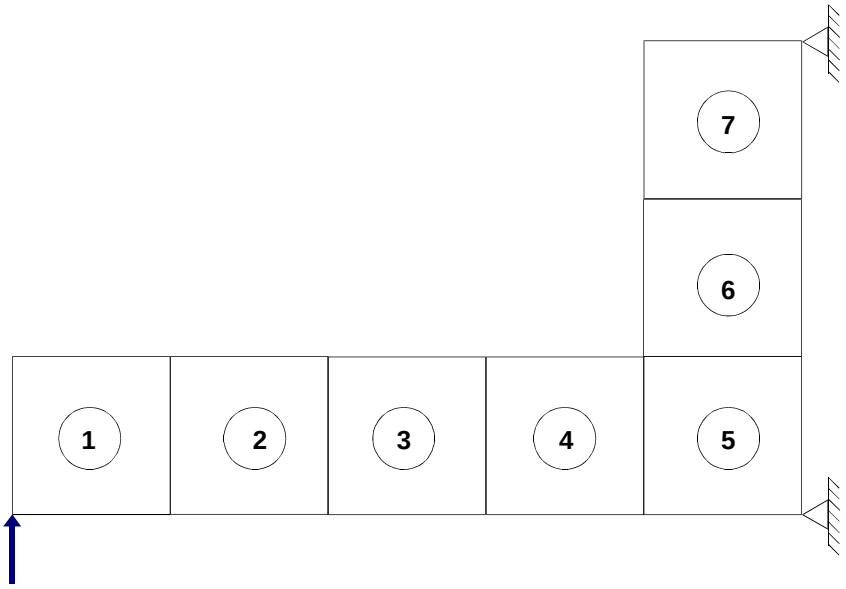

Figure 20. *The loading, boundary conditions and the subdomains for Example 1*

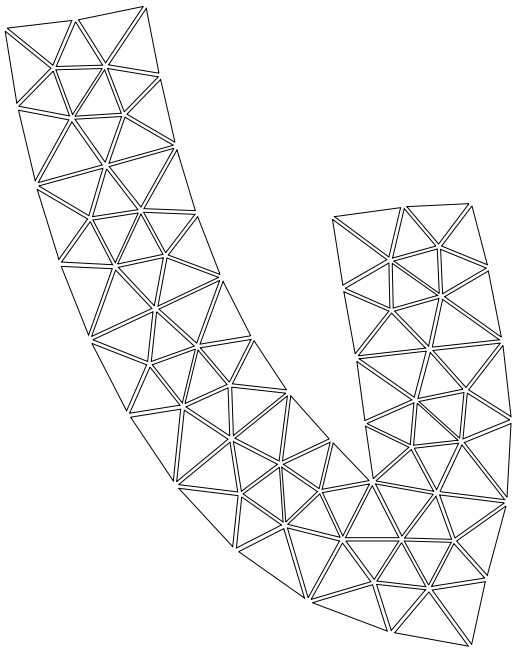

Figure 21. *The final geometry obtained after the DR analysis for Figure. 19*

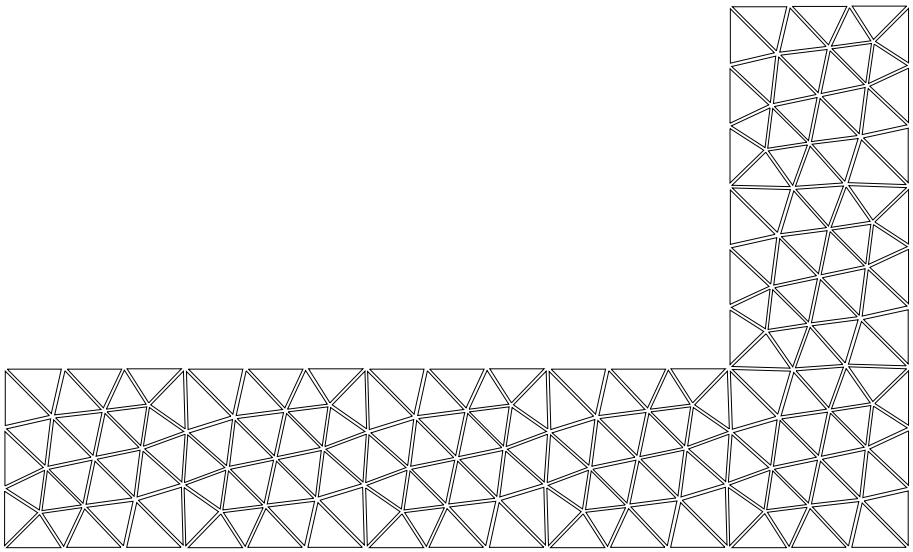

Figure 22. *The second mesh with 123 nodes and 196 elements*

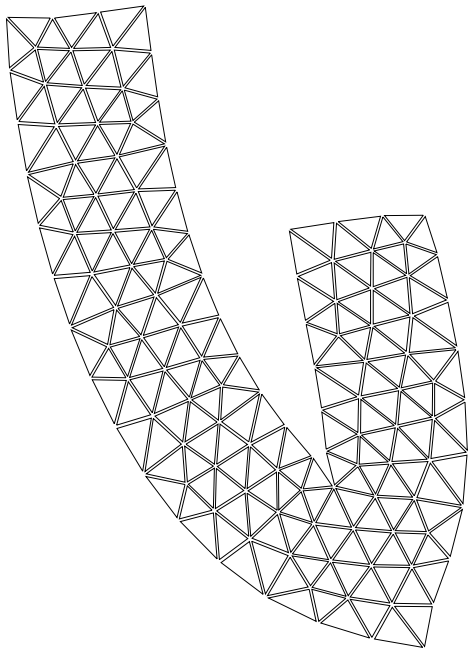

Figure 23. *The final geometry given by the DR method for the descritization shown in Fig. 22*

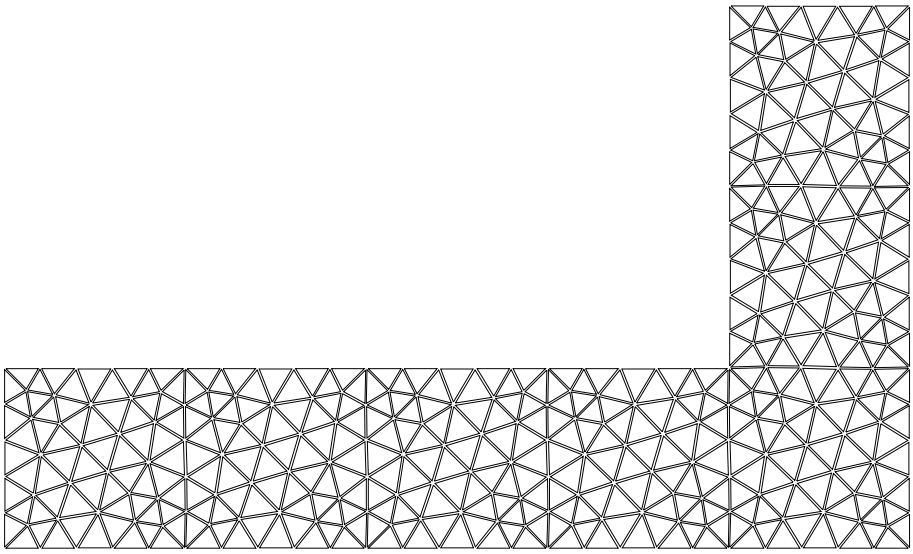

Figure 24. *The third mesh with 258 nodes and 434 elements*

Mesh	t_{seq}	t_{parr}	t_{part}	t_{calc}/t_{comm}	$S = \frac{t_{seq}}{t_{parr}+t_{part}}$	$E = \frac{S}{P}\%$
First	2.25	0.433	0.283	$0.32/0.09 = 3.559$	3.14	44.85
Second	8.12	1.45	0.633	$1.18/.25 = 4.72$	3.9	55.68
Third	40.21	6.66	1.9	$5.886/.741 = 7.943$	4.7	67.1

Table 6. *The sequential times, parallel times , partitioning of data times, ratio of calculation time to communication time,* "speed-ups" *and efficiencies for Example 1 with number of processors* $P = 7$.

+ 6 processors took 8.12 and 1.45 minutes, respectively to solve the integration scheme. The final shape of the structure obtained after the analysis is shown in Fig. 23

The the third remeshing resulted in 258 nodes and 434 elements as shown in Fig. 24. The timings for the sequential code and the parallel code running on ROOT + 6 processor were 40.21 and 6.667 minutes respectively. The final shape of the structure is shown in Fig. 25.

For the three meshes shown in figures 19, 22 and 24 the sequential times t_{seq}, parallel times t_{parr}, times for partioning of data t_{part}, *"speed-ups"* S and efficiencies E are listed in table 6. The effects of changing the mesh density on the computational & communication times, *"Speed-up"* and efficiency are shown in figures 26, 27 and 28, respectively.

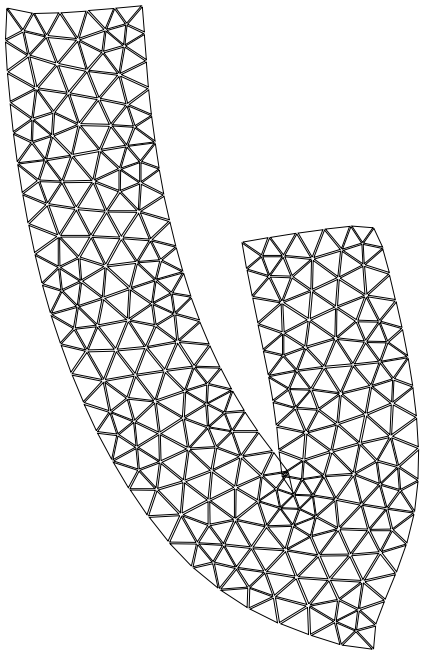

Figure 25. *The final geometry given by the DR method for the descritization shown in Fig. 24*

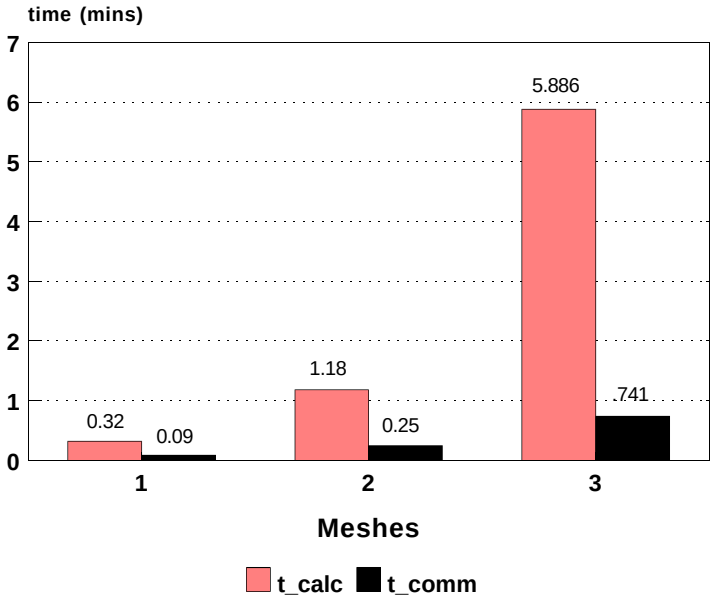

Figure 26. *The computational and communication times for the first, second and the third meshes of Example 1*

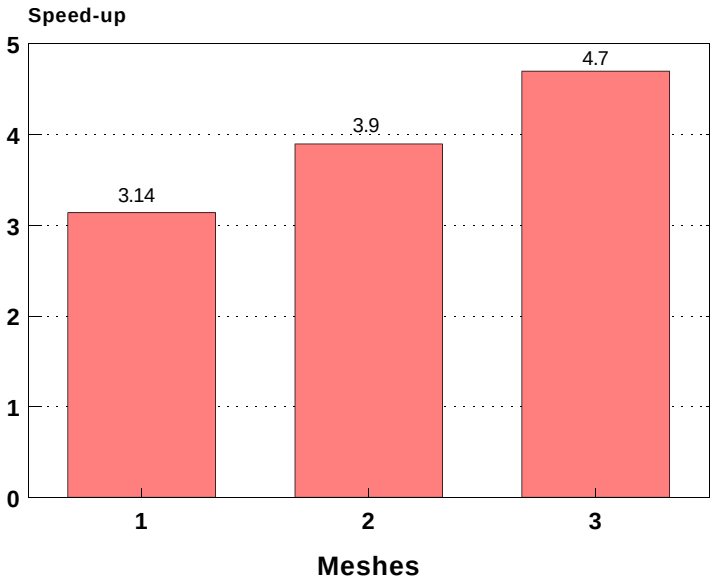

Figure 27. *The* "Speed-ups" *for the first, second and the third meshes of Example 1*

Figure 28. *The* "Efficiencies" *for the first, second and the third meshes of Example 1*

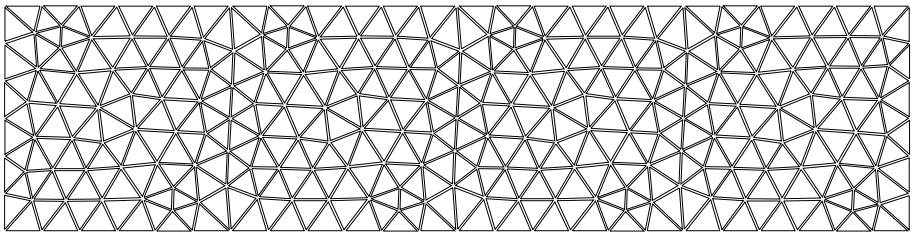

Figure 29. *Example 2: A mesh comprising 227 nodes and 392 elements*

Mesh	t_{seq}	t_{parr}	t_{part}	P	t_{calc}/t_{comm}	$S = \frac{t_{seq}}{t_{parr}+t_{part}}$	$E = \frac{S}{P}\%$
First	10.733	5.35	0.8833	2	$5.29/0.03 = 176.333$	1.72	86.07
Second	10.733	2.766	1.2000	4	$2.67/0.075 = 35.6$	2.71	67.65

Table 7. *The sequential times, parallel times, partitioning of data times, number of processors, ratio of calculation time to communication time,* "speed-ups" *and efficiencies for Example 2*

Example 2

The domain shown in Fig. 29 was partitioned into two identical subdomains, Figure 30 (a) and solved on two processors in the parallel code implementation. The times for the sequential and the parallel code to perform the integration scheme came as 10.7333 and 5.35 minutes respectively.

The same problem was then partitioned into four subdomains, Figure 30 (b) and run on four processors. The time the parallel code to perform the integration scheme was 2.71 minutes. The final shape of the problem as given by the DR solution is shown in Fig. 31.

For the problem shown in Fig. 29 descretized for parallel code by using two and four subdomains respectively the sequential times t_{seq}, parallel times t_{parr}, times for partioning of data t_{part}, *"speed-ups"* S and efficiencies E are listed in table 7. The effects of increasing the number of processors, while retaining the same mesh, on the computational & communication times, *"Speed-up"* and efficiency are shown in figures 32, 33 and 34, respectively.

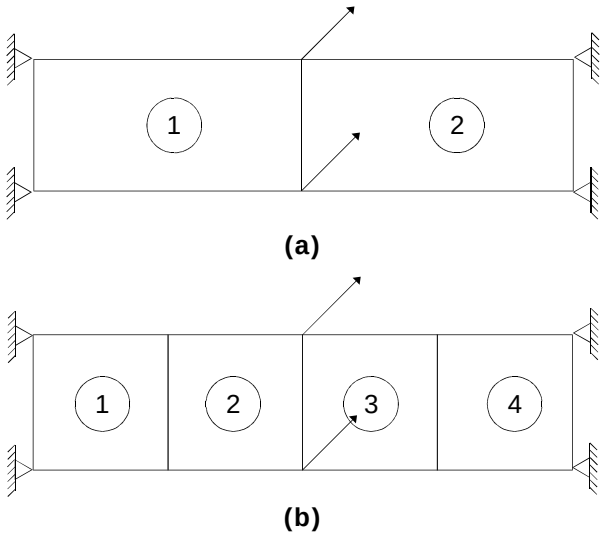

Figure 30. *Loading, boundary conditions and the subdomain topologies for Example 2*

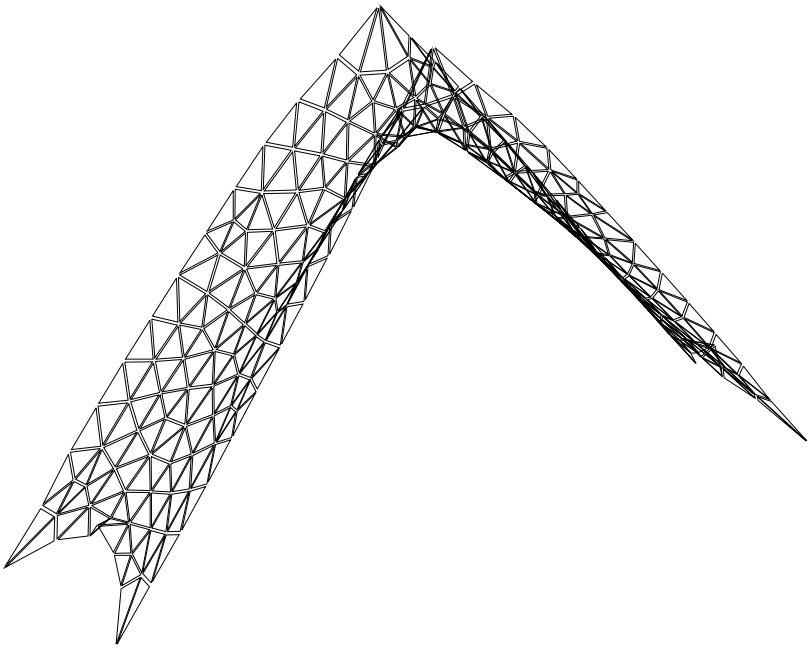

Figure 31. *The DR solution for the mesh problem shown in Fig. 29*

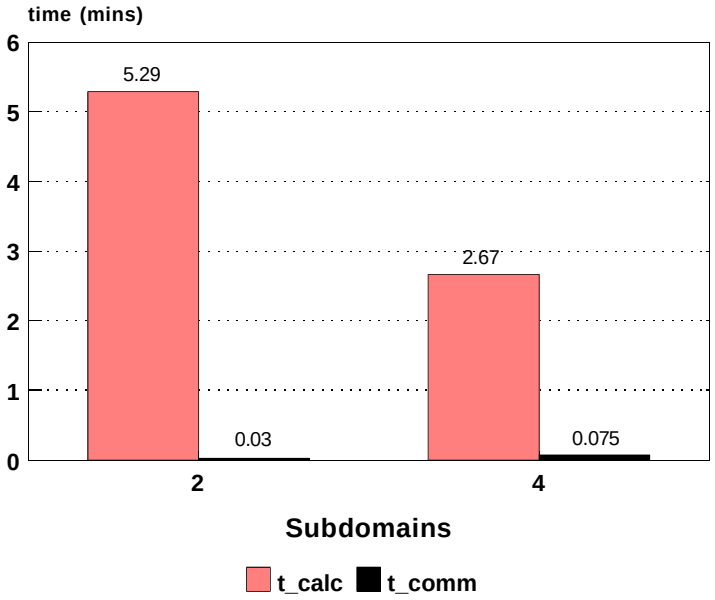

Figure 32. *The compuatational and communication times for the first and the second subdomain topologies for Example 2*

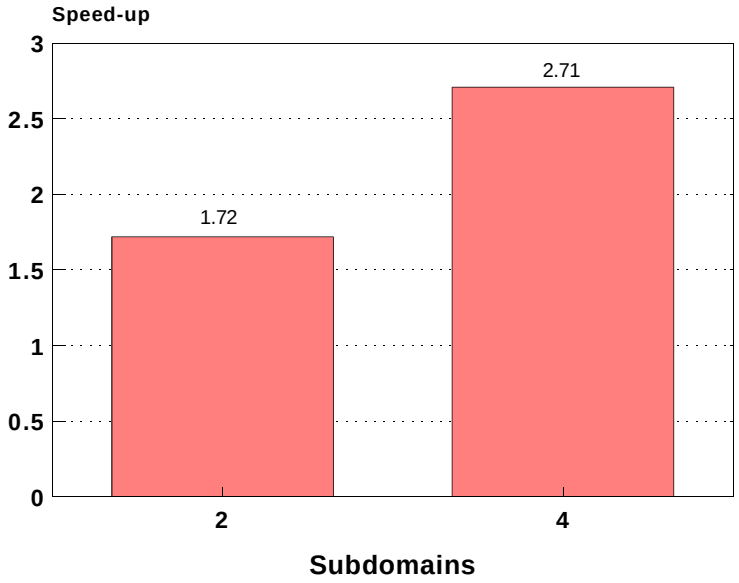

Figure 33. *The* "Speed-ups" *for the first and the second subdomain topologies for Example 2*

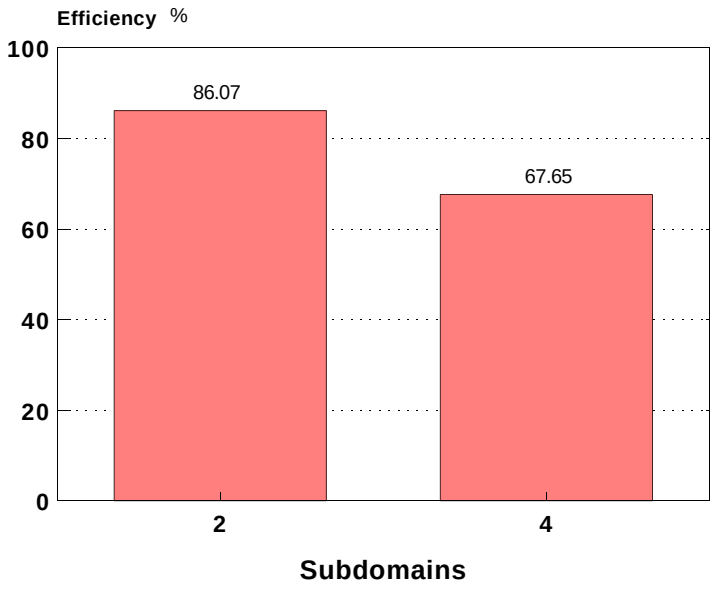

Figure 34. *The* "Efficiencies" *for the first, second and the third meshes of Example 2*

11 Conclusions and Remarks

The need for a high performance DR method was urgently felt on account of the in-ordinate increase in computational time which resulted in the case of large problems.

Some of the popular parallel processing strategies i.e

- Processor Farming
- Algorithmic Parallelism
- Geometric Parallelism

were investigated and it was shown that for transputer-based systems *"geometric parallelism"* offered best opportunities for the development of such an algorithm. An algorithm based upon *"geometric parallelism"* was implemented for the DR analysis of membrane structures using constant strain triangular finite elements. Inspite of the fact that the problem of mesh partitioning was not formally addressed in this study, which could have reduced the time spent on mesh partitioning, very encouraging results were obtained regarding the *"speed-ups"* and effeciencies of the parallel algorithm.

It was noted that the effecient use of the transputer resources increased with the increase in the computational load per processor. It can therefore be safely inferred that the large DR problems can be dealt more effectively under the proposed algorithm. Problems requiring several hours of users time on sequential DR based code may now be solved in fraction of the time on a personal computer fitted with an expansion card fitted with 9 or 10 transputer modules (TRAMS). The computational advantage thus obtained may be used for further research and development in the areas such as form finding for flexible structures, design optimization of membrane structures, modelling of non-linear

materials and possibly in the further enhancement of the DR algorithm itself with respect to e.g. the selection of the optimal time steps in the method.

Acknowledgment The authors would like to acknowledge the useful discussions with other members of the Structural Engineering Computational Technology Group (SECT) at Heriot-Watt University. In particular the contribution of John Hampshire, Jørgen Stang and János Sziveri was appreciated.

The support of Mike Cahill, Neil Clausen and Mark Wilson of Transtech Limited and Nakhee Nory and John Machar of *3L* Ltd for contributions to the SECT projects described in this paper is gratefully acknowledged.

References

[1] Topping, B.H.V., Khan, A.I., "Parallel Finite Element Analysis", Saxe-Coburg Publications, Edinburgh, 1992.

[2] Khan, A.I., Topping, B.H.V., "Parallel Adaptive Mesh Generation", Computing Systems in Enginering, v2, 1991

[3] Argyris, J.H., "Recent Advances in Matrix Methods of Structural Analysis in Aeronautical Science", vol. 4, Pergamon Press, London, 1963.

[4] Barnes, M.R., "Formfinding and Analysis of Tension Space Structures by Dynamic Relaxation", PhD thesis, The City University, London, 1977.

[5] Barnes, M.R., "Dynamic relaxation analysis of tension networks", International Conference on Tension Structures, London, April, 1974.

[6] Barnes, M.R., "Applications of dynamic relaxation to the design and analysis of cable, membrane and pnematic structures", Second International Conference on Space Structures, Guildford, 1975.

[7] Barnes, M.R., "Form-finding of Minimum Surface Membranes", IASS Congress on Structures for Space Enclosure, Montreal, July, 1976.

[8] Barnes, M.R., "Form-finding, Analysis and Patterning of Tension Structures", Third International Conference on Space Structures, Guildford, September, 1984.

[9] Barnes, M.R., "Computer Aided Design of the shade membrane roofs for EXPO 88", Structural Engineering Review, v.1, 3-13, 1988.

[10] Barnes, M.R., Topping, B.H.V., Wakefield, D.S., "Aspects of Form-finding by dynamic relaxation", International Conference on Slender Structures, The City University, London, September, 1977.

[11] Barnes, M.R., Wakefield, D.S., "Dynamic Relaxation applied to Interactive Formfinding and Analysis of Air-Supported Structures",

[12] Biggs, J.M., "Introduction to Structural Dynamics", McGraw Hill Book Company, New York, 1964.

[13] Cundall, P.A., "Explicit finite-difference methods in Geomechanics", Blacksburg, Va, June, 1976.

[14] Day, A.S., An Introduction to Dynamic Relaxation", The Engineer, January, 1965

[15] Hampshire, J., Topping, B.H.V., Chan, H.C., "Three node triangular bending elements with one degree of freedom per node", to be published in Engineering Computations, 1992.

[16] Hudson, P., Topping, B.H.V., "The design and construction of reinforced concrete 'tents' in the Middle East', The Structural Engineer, v.69, n.22, 379-385, 1991.

[17] Moncrieff, E., Topping, B.H.V., "Computer Methods for the Generation of Membrane Cutting Patterns", Computers and Structures, v. 37, n.4, 441-450, 1990.

[18] Moncrieff, E., Topping, B.H.V., "The Optimization of Stressed Membrane Surface Structures using Parallel Computational Techniques", v. 17, 205-218, 1991.

[19] Otter, J.R.H., Cassell, A.C., Hobbs, R.E., "Dynamic Relaxation", discussion, Proc. ICE, v. 723-750, 1967.

[20] Papadrakakis, M., "Discrete statical analysis of structural mechanisms", PhD thesis, The City University, October, 1978.

[21] Tan, K.Y., Barnes, M.R., "Numerical representation of stress/strain relations for coated fabrics", in *The Design of Air-Supported Structures,* Proceedings of the International Conference, The Institution of Structural Engineers, Bristol, July, 1984.

[22] Topping, B.H.V., Khan,A.I., "", to be published, 1992

[23] Khan, A.I., Topping, B.H.V., "—", to be published, 1993

[24] Vincent, J., Jeronimidis, G., Topping, B.H.V., Khan, A.I., "The Meachanical Design of Skin: towards the development of new materials", presented at the SFB 230 Second International Symposium on Natural Structures, Stuttgart University, Sept, 1991.

[25] Wakefield, D.S., "Pretensioned Networks supported by Compression Arches", PhD thesis, The City University, April, 1980.

[26] Wakefield, D.S., "Geodesic Strings in Formfinding and Patterning", Buro Happold, Consulting Engineers, Bath, Internal Report, 1982.

[27] Williams, C.J.K., "Form-finding and cutting patterns for air-supported structures", Symp. on Air Supported Structures, Proceedings of the International Structural Engineering Conferecnce, London, April, 1980.

[28] *"Parallel C user Guide (compiler version 2.1)", Reference Manual,* 3L Ltd, Livingston, Scotland, 1989.

[29] F. P. Preparata, M. I. Shamos, "Computational Geometry An Introduction",Springer-Verlag New York Inc., 1985.

[30] O. C. Zienkiewicz, J. Z. Zhu, *"A simple error estimator and adaptive procedure for practical engineering analysis"*, Int J for Nunmerical Methods in Engineering, vol. 24, 337-357, 1987.

[31] J. Peraire, M. Vahdati, K. Morgan, O. C. Zeinkiewicz, *"Adaptive remeshing for compressible flow computations"*, Journal of Computational Physics, vol. 72, 449-466, 1987.

[32] W. Atamaz-Sibia, E. Hinton, *"Adaptive mesh refinement with the Morley plate element"*, NUMETA 90, Proceedings of the Third International Conference on Numerical Methods in Engineering and Applications, University College of Swansea, Wales, 7-11 January, 1990, Edited by G. Pande and J. Middleton, vol. 2 pp. 1044-1055, Elsevier Applied Science, London, 1990.

[33] J. Peraire, K. Morgan, J. Peiro, *"Unstructured mesh methods for CFD"*, Department of Aeronautics, Imperial College, London, IC Aero Report 90-04, June, 1990.

[34] J. Bonet, J. Peraire, *"An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problem"*, Int. J for Numerical Methods in Engineering, vol. 31, 1-17, 1991.

[35] J. S. R. Alves Filho, D. R. J. Owen, *"Using transputers in Finite Element Calculations : a first approach"*, SERC Loan Report No: TR1/034, Science and Engineering Research Council, 1989.

[36] J. Favenesi, A. Daniel, J. Tomnbello and J. Watson, *"Distributed finite element analysis using a transputer network",* Computing Systems in Engineering, vol. 1, Nos 2-4, 171-182, 1990.

[37] E. Moncrieff, B. H. V. Topping , *"The optimization of stressed membrane surface structures using parallel computational techniques",* Engineering Optimization, to appear May 1991.

[38] S.H. Lo, *"A new mesh generation scheme for arbitrary planar domains",* Int. J. for Numererical Methods in Engineering, vol.21, 1403-1426, 1985.

[39] H. Jin and N. E. Wiberg, *"Two-dimensional mesh generation, adaptive remeshing and refinement",* Int. J. for Nunerical Methods in Engineering, vol. 29, 1501-1526, 1990.

[40] TMB08, *Installation and User Manual*, Transtech Devices Ltd, Wye Industrial Estate, London Road, High Wycombe, Buckinghamshire.

[41] TMB12, *Installation and User Manual*, Transtech Devices Ltd, Wye Industrial Estate, London Road, High Wycombe, Buckinghamshire.

[42] INMOS Limited, *"Transputer Reference Manual"*, Prentice-Hall, Hertfordshire, UK, 1988.

[43] K.C. Bowler, R.D. Kenway, G.S. Pawley, D. Roweth, *An Introduction to OCCAM 2 Programming",* Chartwell-Bratt Ltd, Studentlitteratur, Lund, 1991.

[44] *"Tbug User Guide",* 3L Ltd, Livingston, Scotland, 1989.

[45] *"Parallel C user Guide (compiler version 2.1)", Reference Manual,* 3L Ltd, Livingston, Scotland, 1989.

[46] A. S. Tanenbaum, *"Operating Systems: Design and Implementation",* Prentice-Hall, Englewood Cliffs, NJ, USA, 1987.