Sādhanā Vol. 29, Part 1, February 2004, pp. 57–81. © Printed in India

Comparative efficiencies of three parallel algorithms for nonlinear implicit transient dynamic analysis

A RAMA MOHAN RAO[1], T V S R APPA RAO[1] and B DATTAGURU[2]

[1]Structural Engineering Research Centre, CSIR Campus, Taramani, Chennai 600 113, India
[2]Department of Aerospace Engineering, Indian Institute of Science, Bangalore 560 012, India
e-mail: arm2956@yahoo.com; apparaot@vsnl.com; datgur@aero.iisc.ernet.in

MS received 1 August 2002; revised 27 December 2002

Abstract. The work reported in this paper is motivated by the need to develop portable parallel processing algorithms and codes which can run on a variety of hardware platforms without any modifications. The prime aim of the research work reported here is to test the portability of the parallel algorithms and also to study and understand the comparative efficiencies of three parallel algorithms developed for implicit time integration technique. The standard message passing interface (MPI) is used to develop parallel algorithms for computing nonlinear dynamic response of large structures employing implicit time-marching scheme. The parallel algorithms presented in this paper are developed under the broad framework of non-overlapped domain decomposition technique. Numerical studies indicate that the parallel algorithm devised employing the conventional form of Newmark time integration algorithm is faster than the predictor–corrector form. It is also accurate and highly adaptive to fine grain computations. The group implicit algorithm is found to be extremely superior in performance when compared to the other two parallel algorithms. This algorithm is better suited for large size problems on coarse grain environment as the resulting submeshes will obviously be large and thus permit larger time steps without losing accuracy.

Keywords. Transient dynamic analysis; parallel processing; Newmark algorithm; group implicit algorithm; domain decomposition.

1. Introduction

Considerable research efforts have been made during the past three decades to develop efficient and reliable time integration algorithms for solving the second-order dynamic equilibrium equations that arise due to finite element discretisations. Mathematically, these governing equilibrium equations are generally stiff, i.e. they exhibit a wide spectrum in linear range. For instance, in the area of structural dynamics, the bending stiffness of beams is typically much smaller than their axial stiffness, which tends to give widely varying eigen frequencies.

Another key characteristic of most of the structural dynamic problems is that the response lies in the lower part of the spectrum. A typical example is a structure subjected to earthquake loads.

The algorithms best suited to this type of applications are those, which accurately integrate the low frequency content of the response and successfully damp out the high frequency modes. This inevitably means that the algorithm must be unconditionally stable and thereby rules out the possibility of employing conditionally stable explicit algorithms for structural dynamic applications. Most of the earlier research is focussed on developing unconditionally stable time-stepping algorithms for linear and nonlinear applications. The most prominent example of that class of algorithms, and the one, which played a pivotal role in all subsequent developments, is Newmark's method (Newmark 1959).

An unconditionally stable Newmark method is implicit. In large scale structural dynamic applications including the nonlinearities, the equation solving phase is the most dominating phase in the implicit algorithm. Considerable efforts have been made in the past two decades to alleviate this source of computational cost while retaining the requisite stability of the algorithm. Some of the works in this direction are partition methods (Belytschko & Hughes 1976), staggered procedures for coupled field problems (Felippa & Park 1980), method of alternating directions (Hughes & Winget 1983), semi-implicit algorithm of Trujillo (1977) and element-by-element methods (Ortiz *et al* 1983), and virtual pulse techniques (Chen *et al* 1995) etc.

However, in recent years the most exciting possibility in the algorithm development area for nonlinear dynamic analysis has been the emergence of parallel processing machines. In the last couple of years, significant advances in hardware and software technologies have been made in parallel and distributed computing which includes development of portable software development environments and cost effective parallel computing using cluster of workstations and also heterogeneous computing etc. At present, it looks as if that parallel processing, as a discipline is fairly matured and ready for development of serious commercial applications. Programming models required to take advantage of the parallel and distributed computer architecture are significantly different from the traditional paradigm for a serial program. Implementation of an engineering application, besides optimising matrix manipulation kernels for the new computing environment, must take careful consideration of the overall organisation and data structures of the program.

Many researchers have devised algorithms for nonlinear dynamic analysis exploiting parallelism in both explicit and implicit time integration techniques. The explicit time integration algorithms like central difference method can easily be moved on to parallel processing machines as the resulting dynamic equilibrium equations are decoupled when mass and damping matrices are diagonal. These explicit algorithms are ideally suited for wave propagation type of problems (e.g. structures subjected to impact and blast loads) where very stringent time step length need to be employed to accurately integrate intermediate and high frequency modes. However, for structural dynamic problems, where the low frequency modes are of interest, these conditionally stable explicit algorithms cannot be employed efficiently. On the other hand implicit algorithms are unconditionally stable, but parallel implementation is not so straightforward.

Considerable efforts have been made to improve the performance of these implicit algorithms in parallel processing environment. In recent years, a number of methods have been proposed for developing parallel processing strategies for transient dynamic nonlinear finite element analysis employing implicit time integration. Here no attempt has been made to present a comprehensive review. However, some of the selected earlier works are outlined to give a flavor of directions in which attempts have been made to devise parallel approaches for dynamic analysis employing implicit time integration techniques. Hajjar & Abel (1988) have employed the implicit Newmark constant average acceleration algorithm for solution of dynamic analysis of framed structures on a network of workstations. For devising parallel algorithms, the domain decomposition strategy coupled with Preconditioned Conjugate Gradient (PCG) algorithm for the iterative solution of the interface stiffness coefficient matrix has been employed. Profile storage scheme has been employed for storing the global matrices. It was concluded that the PCG algorithm is attractive for parallel processing as it requires less interprocessor communication and is easier to balance the workload among processors than direct methods. The authors have reported the speedup of their parallel algorithms as 1·95 and 2·32 on three and four Micro-VAX workstations respectively while solving a 30 storey 3D frame dynamic analysis problem. Chiang & Fulton (1990) investigated implicit Newmark type methods with a skyline Cholesky decomposition strategy for FLEX/32 shared memory multi-computer and the Intel iPSC Hypercube message passing system. It was shown that the shared database nature of the decomposition algorithm made the FLEX/32 multicomputer a more efficient parallel environment than the Hypercube computer. Chiang & Fulton (1990) have reported a moderate speedup of 2·81 on six processors of Flex/32 machine while solving a shell structure idealised using sixty triangular shell elements. Ali & Parsons (2000) have implemented an MPI based Newmark time integration algorithm for dynamic analysis. A preconditioned conjugate gradient algorithm with diagonal preconditioner has been employed for the solution of the system of equations and implemented their parallel code on both IBM SP-2 and SGI origin2000 parallel computers. The authors have reported a speedup of 23·75 and 24·02 on IBM SP-2 and SGI origin2000 respectively, while solving a 3D finite element mesh of size 6126 degrees of freedom. However, the timings of the sequential implementations considered by them for arriving at the speedup are not the actual ones but estimated values.

Benningof & Wu (1991a) have devised domain decomposition methods that permit independent subdomain computation. In these algorithms, independently computed subdomain responses were corrected to obtain response for the global problem using interface portions of independent subdomain responses. These corrections are carried out less frequently (say every fifth time step) in order to reduce the computations for correcting the independent subdomain responses. However, a drawback of these methods is that they are conditionally stable and, their stability behaviour is much more complex than that of standard global time stepping algorithms. Benningof & Wu (1991b) have later proposed unconditionally stable parallel algorithms with multi-step independent computations for linear dynamic analysis. These algorithms however are not suitable for nonlinear implementations.

It can be observed from the brief review that most of the algorithms for implicit transient dynamic analysis have been devised by using a parallel solver or by reordering the computations and also most of the implementations are on outdated hardware and software development environment. It also has been observed that most of the research efforts carried out so far has concentrated on evaluating the performance of a particular algorithm on a particular hardware platform. However, little effort has been made so far to implement and compare the relative efficiencies of different parallel algorithms for nonlinear dynamic analysis on a single or multiple parallel processing platforms. Under these circumstances, it is extremely difficult to appreciate the relative performance of various algorithms on a particular hardware platform.

Keeping this in view, in this paper, an effort has been made to implement three parallel algorithms for implicit time integration technique to compute nonlinear dynamic response of large structures. Two hardware platforms available in India have been chosen for evaluating the implemented parallel algorithms. One is the Param at National Param supercomputing facility, Pune and the other one is the IBM SP-2, which is installed at Supercomputing Education and

Research Centre (SERC), Indian Institute of Science (IISc), Bangalore. Moreover the parallel algorithms presented in this paper are implemented employing message passing interface (MPI) software development environment (Gropp *et al* 1994). Till date only very few MPI-based parallel algorithms for nonlinear dynamic analysis employing explicit and implicit time integration techniques have been reported in the literature (Danielson *et al* 1998; Ali & Parsons 2000; Sziveri *et al* 2000).

2. Newmark's time-stepping algorithm

The spatial discretisation of the structure leads to the governing equilibrium equation of structural dynamics and can be expressed as:

$$[M]\{a\} + [C]\{v\} + [K]\{d\} = \{f(t)\} \tag{1}$$

with $d(0) = d_0$ and $v(0) = v_0$ where, M is the mass matrix, C is the damping matrix and K is the stiffness matrix. a, v and d are the acceleration, velocity and displacement vectors respectively. Solution of this initial value problem requires integration through time. This is achieved numerically by discretising in time the continuous temporal derivatives that appear in the equation. Any one of the time integration procedures can be used for this purpose. The most widely used family of direct time integration methods for solving equation (1) is the Newmark family of methods (Newmark, 1959). These methods are based on equilibrium at the $(n+1)^{\text{th}}$ time step, i.e., at time $(t_{n+1} = t_n + \Delta t)$, and is given by

$$[M]\{a\}_{n+1} + [C]\{v\}_{n+1} + [K]\{d\}_{n+1} = \{f(t)\}_{n+1}. \tag{2}$$

These methods are based on the following finite difference approximations of the velocity and displacement vectors:

$$d_{n+1} = d_n + \Delta t v_n + \Delta t^2/2\{(1-2\beta)a_n + 2\beta a_{n+1}\}, \tag{3}$$
$$v_{n+1} = v_n + \Delta t\{(1-\gamma)a_n + \gamma a_{n+1}\}. \tag{4}$$

The parameters β and γ define the Newmark family of algorithms. Equations (2)–(4) are the three equations for determining the three unknowns d_{n+1}, v_{n+1} and a_{n+1} using the known values d_n, v_n and a_n obtained from the previous step of computations. Solving a_{n+1} and v_{n+1} from (3) and (4) and substituting them in (1), the following system of equations can be obtained, which can be solved for d_{n+1}

$$\begin{aligned}\left[[K] + \frac{\gamma}{\beta\Delta t}[C] + \frac{1}{\beta\Delta t^2}[M]\right]\{d\}_{n+1} &= \{f\}_{n+1} \\ &+ [C]\left\{\frac{\gamma}{\beta\Delta t}\{d\}_n + \left(\frac{\gamma}{\beta} - 1\right)\{v\}_n + \Delta t\left(\frac{\gamma}{2\beta} - 1\right)\{a\}_n\right\} \\ &+ [M]\left\{\frac{1}{\beta\Delta t^2}\{d\}_n + \frac{1}{\beta\Delta t}\{v\}_n + \left(\frac{1}{2\beta} - 1\right)\{a\}_n\right\}\end{aligned} \tag{5}$$

The matrix $([K] + (\gamma/\beta\Delta t)[C] + (1/\beta\Delta t^2)[M])$ in the above equation is usually referred to as the effective stiffness matrix $[K]^{\text{eff}}$. For the linear case, the effective stiffness matrix remains constant in all the computational steps unless the step size is changed. However, for nonlinear analysis the effective stiffness changes at every time step, and linear algorithm must be modified. For nonlinear analysis the effective stiffness matrix can be written as $([K_T] + (\gamma/\beta\Delta t)[C_T] + (1/\beta\Delta t^2)[M])$, where K_T is the tangent stiffness matrix.

The effective residual force, r can be obtained from the equation,

$$\{r\}_{n+1} = [M]\{a\}_{n+1} + \{p_{n+1}(d_{n+1}, v_{n+1})\} - \{f_{n+1}(d_{n+1}, t_{n+1})\}, \tag{6}$$

where p is the internal force vector. An iteration scheme such as Newton–Raphson, modified Newton–Raphson or quasi-Newton methods can be employed to resolve the nonlinearity.

The parameters β and γ determine the stability and accuracy characteristics of the algorithm. Constant acceleration method is obtained when $\beta = 1/4$ and $\gamma = 1/2$. The method is implicit, unconditionally stable, second-order accurate, and one of the most effective and popular methods for structural dynamics problems (Newmark 1959).

In the present work, unconditionally stable Newmark-β implicit time integration technique has been chosen to solve for nonlinear dynamic response of structures. In order to compute nonlinear dynamic response employing Newmark time integration algorithm, an iterative procedure needs to be employed in each time step to solve for incremental displacements, velocities and accelerations. The complete algorithm for nonlinear transient dynamic analysis employing Newmark's technique can be found in Cook *et al* (1989).

It is also possible to implement the Newmark scheme in a predictor–corrector form (Hughes & Liu 1978), which involves predicting initially the displacements, velocities and accelerations using

$$d_{n+1} = d^P_{n+1} = d_n + \Delta t v_n + \Delta t^2(0{\cdot}50 - \beta)a_n, \tag{7}$$

$$v_{n+1} = v^P_{n+1} = v_n + \Delta t(1 - \gamma)a_n, \tag{8}$$

$$a_{n+1} = (d_{n+1} - d^p_{n+1})/(\Delta t^2\beta) = 0. \tag{9}$$

Based on the predicted values, the internal force vector and the effective stiffness matrix are computed. The incremental displacement Δd obtained by solving the system of equations is used to correct the values.

$$d_{n+1} = d^P_{n+1} + \Delta d, \tag{10}$$

$$a_{n+1} = (d_{n+1} - d^P_{n+1})/(\Delta t^2\beta), \tag{11}$$

$$v_{n+1} = v^P_{n+1} + (\Delta t\gamma)a_{n+1}. \tag{12}$$

The predictor–corrector algorithm can be found in Hughes & Liu (1978).

3. Domain decomposition algorithms

In dynamic analysis of structures, domain decomposition begins by performing substructuring. The simultaneous set of linear algebraic equations arising from the implicit algorithm for nonlinear analysis is given by

$$[\tilde{K}]_n\{\Delta\tilde{d}\} = \{\tilde{f}\}_{n+1}, \tag{13}$$

where $\tilde{K}_n$, $\Delta\tilde{d}$ and $\tilde{f}_{n+1}$ are respectively submesh effective stiffness, incremental displacement vector, and effective residual force vector. $\tilde{K}_n$ and $\tilde{f}_{n+1}$ are obtained using any suitable implicit procedure for a submesh.

For each submesh, the degrees of freedom (d.o.f.) are however partitioned into internal d.o.f.s i, and border d.o.f.s, b. Dropping the time-step subscripts n and $n+1$, the above equation for any submesh, s can be written in partitioned form as

$$\begin{bmatrix} \tilde{K}^{\mathrm{s}}_{\mathrm{ii}} & \tilde{K}^{\mathrm{s}}_{\mathrm{ib}} \\ \tilde{K}^{\mathrm{s}}_{\mathrm{bi}} & \tilde{K}^{\mathrm{s}}_{\mathrm{bb}} \end{bmatrix} \begin{Bmatrix} \Delta d^{\mathrm{s}}_{\mathrm{i}} \\ \Delta d^{\mathrm{s}}_{\mathrm{b}} \end{Bmatrix} = \begin{Bmatrix} \tilde{f}^{\mathrm{s}}_{\mathrm{i}} \\ \tilde{f}^{\mathrm{s}}_{\mathrm{b}} \end{Bmatrix}. \tag{14}$$

By eliminating the internal degrees of freedom, the condensed form of the equations can be written as:

$$[\tilde{K}]^{\mathrm{s}}_{\mathrm{B}}\{\Delta d\}^{s}_{b} = \{\tilde{f}\}^{\mathrm{s}}_{\mathrm{B}}, \tag{15}$$

where

$$[\tilde{K}^{\mathrm{s}}_{\mathrm{B}}] = [\tilde{K}^{\mathrm{s}}_{\mathrm{bb}}] - [\tilde{K}^{\mathrm{s}}_{\mathrm{bi}}][\tilde{K}^{\mathrm{s}}_{\mathrm{ii}}]^{-1}[\tilde{K}^{\mathrm{s}}_{\mathrm{ib}}] \text{ and} \tag{16}$$

$$\{\tilde{f}^{\mathrm{s}}_{\mathrm{B}}\} = \tilde{f}^{\mathrm{s}}_{\mathrm{b}} - [\tilde{K}^{\mathrm{s}}_{\mathrm{bi}}][\tilde{K}^{\mathrm{s}}_{\mathrm{ii}}]^{-1}[\tilde{f}^{\mathrm{s}}_{\mathrm{i}}]. \tag{17}$$

The subscript B is used to indicate the condensed problem. The resulting matrix $\tilde{K}^{\mathrm{s}}_{\mathrm{B}}$ can be considered the effective stiffness matrix for the submesh, s, in terms of the interface d.o.f.s. Once the condensed stiffness matrices for all submeshes are obtained, the global interface stiffness coefficient matrix can be established by means of finite element assembling operation.

$$[K]^{*} = \Sigma[\tilde{K}]^{\mathrm{s}}_{\mathrm{B}}. \tag{18}$$

The resulting set of equations, K^{*} can then be solved either using a direct solver or an iterative solver. Once the boundary d.o.f.s are solved, the internal d.o.f.s of each submesh can be computed by using the following equation.

$$\{\Delta d\}^{\mathrm{s}}_{\mathrm{i}} = [\tilde{K}^{\mathrm{s}}_{\mathrm{ii}}]^{-1}\left(\{\tilde{f}^{\mathrm{s}}_{\mathrm{i}} - [K]^{\mathrm{s}}_{\mathrm{ib}}\{\Delta d\}^{\mathrm{s}}_{\mathrm{b}}\right). \tag{19}$$

It is important to note that the interface solution is performed on a greatly reduced set of equations. This whole formulation resembles the static condensation procedure popularly adopted in static analysis. The only difference between the static condensation and the dynamic domain decomposition is that in the later one, the operator is the effective stiffness matrix, which is a linear combination of K, C and M, rather than K alone. This linear combination depends on the basic implicit solution algorithm used. This domain decomposition algorithm can effectively be implemented on parallel processing machines. The entire internal assembly and condensation procedure can be performed without any processor interaction. However, the processors need to interact while assembling the interface stiffness coefficient matrices of the submeshes and subsequent parallel solution of the global interface stiffness coefficient matrix. Since the amount of interprocessor communication overheads are directly related to the number of interface nodes, these domain decomposition based parallel algorithms depends rather heavily on the efficiency of the partitioning of the finite element mesh into submeshes and subsequent mapping of these submeshes on to processors. Optimal partitioning of finite element mesh to maintain equal computational load and at the same time minimising the interface nodes is an NP-hard problem (Garey & Johnson, 1979) and cannot be solved efficiently by conventional optimisation techniques. In view of this, several heuristic partitioning algorithms, that can provide sub-optimal solutions, have been developed and reported in the literature. A detailed review on these mesh partitioning algorithms can be found elsewhere (Rama Mohan Rao 2001). In the present work, an efficient optimal mesh partitioning technique developed by the authors employing genetic algorithms (Rama Mohan Rao *et al* 2004) has been employed for partitioning the finite element meshes.

The main question that arises when devising the parallel algorithms is how efficient the particular algorithm is when compared to the other parallel algorithms. In order to answer this question, three alternative parallel implementations are investigated in this paper and the comparative efficiencies of these algorithms are evaluated on the chosen hardware platform.

4. Parallel implementation of domain decomposition algorithms

In this paper, the non-overlapped domain decomposition algorithm has been employed as a method of solution for the implicit Newmark-β constant average acceleration method. No new inaccuracies or approximations are introduced in the method when it is implemented in parallel. Therefore, the properties of the Newmark-β algorithm is retained in full. As such the algorithm is unconditionally stable, consistent and second-order accurate.

The two main components of the solution of incremental displacements by domain decomposition are the condensation of the internal nodes and the solution of the interface unknowns by solving the interface stiffness coefficient matrix. Profile storage scheme has been employed for storing the global submesh matrices. For the interface solution, a parallel direct solver has been employed.

In the parallel domain decomposition based implicit algorithm, nodes lying along the interface are divided into primary boundary nodes and secondary boundary nodes, with each interface node being a primary boundary node in a submesh and secondary boundary node in all other submeshes sharing that particular node. The nodes, which are not interface nodes, are called internal nodes and internal elements are those, which do not have any boundary nodes. The elements, which consist of boundary nodes, are boundary elements. The division of data for parallel implicit algorithm is shown in figure 1. With this data structure, the parallel non-overlapped domain decomposition algorithm employing the conventional Newmark's algorithm and predictor–corrector algorithm can be formulated as given in figures 2 and 3 respectively. In these two algorithms all the stages of solution are executed in parallel. A

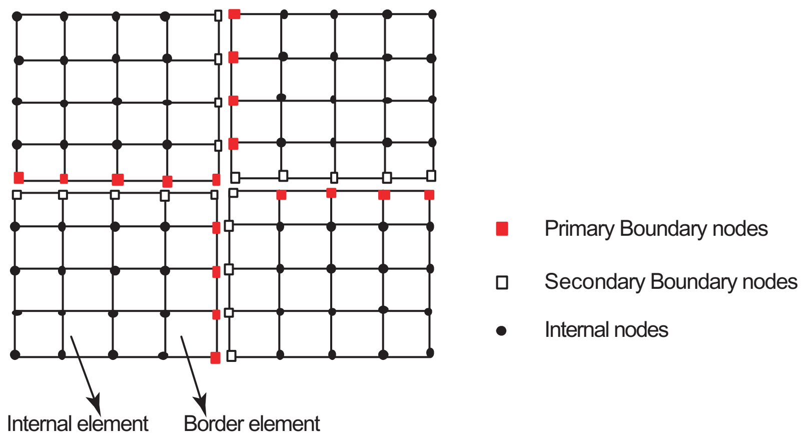


Figure 1. Division of structural data for parallel implicit analysis.

```
1. Input and setup basic analysis data
2. In each processor:
     (i) Compute internal force vector
    (ii) Compute effective global stiffness matrix and load vector
   (iii) Condense internal degrees of freedom
3. Communication:
     (i) Send relevant parts of condensed stiffness and load vectors to adjacent processors
    (ii) Receive relevant parts of data of condensed stiffness and load vectors from adjacent
         processors
   (iii) Assemble the relevant columns of the global interface stiffness coefficient matrix and force
         vector
4. Use parallel solver to solve boundary degrees of freedom
5. In each processor:
     (i) Solve internal degrees of freedom
    (ii) Compute velocities and accelerations
   (iii) Compute stresses and strains
6. Check equilibrium for nonlinear analysis
     (i) Assemble internal force and load vector
    (ii) Compute unbalanced force vector
   (iii) Assemble unbalanced force norm
    (iv) If converged go to (7)
     (v) Solve for displacements (same as 4, 5 (i))
    (vi) Compute velocities and accelerations (same as 5(ii))
   (vii) Recover stresses and strains (same as 5(iii)) and go to (6(i))
7. Go to step 2 for the next time step
```

Figure 2. Parallel algorithm for nonlinear dynamic analysis (conventional form).

parallel profile solver has been employed for the solution of the interface stiffness coefficient matrices. The algorithmic details of the parallel direct solver can be found in Rama Mohan Rao *et al* (1993). Since, during the solution phase the columns of the global (stiffness) matrix are assigned in round-robin fashion to achieve proper load balancing, the assembly of submesh effective stiffness matrices are carried out only for the respective assigned columns of the global matrix in each submesh.

5. Group implicit (GI) algorithm

Ortiz & Nour-Omid (1988) first presented the group implicit time integration algorithm. In this paper, the group implicit algorithm has been implemented for nonlinear dynamic analysis of plates and shells. The GI algorithm is potentially the most promising method for parallel processing. The extension of the algorithm for nonlinear dynamic analysis is discussed first. The data structures, flow of information and communication requirements of the algorithm are then presented.

In the group implicit algorithm, the finite element mesh is partitioned into convenient number of submeshes (or groups of elements) as dictated by the number of processors configured

```
1. Input and set up basic analysis data, set up time step counter $n = 1$
2. In each processor
     (i) Set iteration counter $i = 0$
    (ii) Predict the displacement, velocity and acceleration vectors
   (iii) Compute the internal force vector
    (iv) Compute the residual forces
     (v) Compute effective global stiffness matrix
    (vi) Condense internal degrees of freedom
3. Communication
     (i) Send relevant parts of condensed stiffness and load vectors to adjacent processors
    (ii) Receive relevant parts of data of condensed stiffness and load vectors from adjacent
         processors
   (iii) Assemble the relevant columns of the global interface stiffness coefficient matrix and force
         vector
4. Use parallel solver to solve boundary degrees of freedom
5. In each processor:
     (i) Solve internal degrees of freedom
    (ii) Correct the displacement, velocity and acceleration vectors
6. Communication
     (i) Check for convergence, if not converged set $i = i + 1$ go to step 2(iii)
    (ii) Else, if converged
7. In each processor
   Set
         $d_n =^{(i+1)} d_n$
         $v_n =^{(i+1)} v_n$
         $a_n =^{(i+1)} a_n$
   next time step: $n = n + 1$, and go to next time step.
```

Figure 3. Parallel algorithm for nonlinear dynamic analysis (predictor–corrector form).

for the application. Each group is processed independently employing the Newmark implicit time integration technique over a time step. Free boundary conditions are assumed along all group interfaces. Therefore, the submeshes, which do not have any boundary conditions, are subjected to potential rigid body motion. However, the size of the time step and the inertia force caused by the mass of the structure contributes to alleviate this motion.

The advantages of group implicit algorithm are twofold. One is the efficient inter-processor communications. The processors need to communicate only with their neighbouring processors and it is more or less similar to the communications in explicit algorithm. The second advantage is that they speed up the computations by reducing the equation solving effort even on a single processing machine. As the submesh sizes are reduced, the bandwidths of the local arrays decrease steadily. Fill-in by off-diagonal zeros is consequently diminished as well. The result is a net gain in efficiency during equation solution phase.

The stability and consistency of GI algorithm were derived by Ortiz & Nour-Omid (1988) and hence it is not repeated here. As already mentioned in the previous chapter, GI algorithm is unconditionally stable. Essentially this occurs because the solution of each domain itself is unconditionally stable. In addition, if mass averaging is used, Ortiz *et al* (1988) proved that the GI algorithm is consistent, and also studied the accuracy of GI algorithm. The averaging of multiple solutions of the submesh interface degrees-of-freedom creates inaccuracies in the solution. This is due to lack of full simultaneous solution of the implicit equations, which affects the flow of information from each domain to all other groups. Although information flows properly within each group, it's transmission across the submesh boundaries is more limited than a full implicit solution. Ortiz & Nour–Omid (1988) and Ortiz *et al* (1988) accounted for this inaccuracy by deriving a maximum suggested time step size, above which accuracy clearly degraded in the example problems reported by the authors.

It may be noted that the critical time step size for the central difference algorithm is related to a characteristic dimension of an element in the mesh divided by the wave speed of the material of that element. Similarly, the maximum step size of the group implicit algorithm is related to a characteristic dimension of a group and the wave speed of the material in the group Ortiz & Nour–Omid (1988) (Ortiz *et al* 1988). It is given by

$$\Delta t_{\text{critical}} \leq \min(L^{\text{s}}/C^{\text{s}}), \tag{20}$$

where L^s is the characteristic dimension of group s and c^{s} is the wave speed of the group. The improvement over the stability limit of the explicit central difference algorithm is linear with the increase in the size of the submesh. Hence, this algorithm will be more efficient when applied to very large problems and implemented in coarse grain environment.

For nonlinear dynamic analysis, incremental displacements are the primary unknowns and are averaged on the boundaries by using mass averaging rule. Therefore for any typical submesh m:

$$[K]^{m^*}\{\Delta d\}^m = \{r\}^m_{n+1} \tag{21}$$

where

$$[K]^{m^*} = [K]^*_T + (1/\beta(\Delta t)^2)[M]^m + (\gamma/\beta\Delta t)[C]^m, \tag{22}$$

and

$$\begin{aligned}\{r\}^m_{n+1} = \{f\}^m_{n+1} - \{p\}^m_{n+1} + [M]^m\left(\frac{1}{\beta\Delta t}\{v\}^m_n + \frac{1}{\beta}(1/2-\beta)\{a\}^m_n\right)\\ + [C^m]\left\{\left(\frac{\gamma}{\beta}-1\right)\{v\}^m_n + \left(\frac{\gamma}{2\beta}-1\right)\Delta t\{a\}^m_n\right\}.\end{aligned} \tag{23}$$

The multiple boundary values at the interfaces are averaged by

$$\{\Delta d\} = [M]^{-1}\left\{\sum_{m=1}^{NS}[M]^m\{\Delta d\}^m\right\}. \tag{24}$$

Each submesh m, forms its incremental displacement vector $\{\Delta d\}^m$ by extracting the appropriate values from the global incremental displacement vector $\{\Delta d\}$.

6. Parallel Implementation of group implicit algorithm

The group implicit technique discussed in the previous chapter is implemented for computing nonlinear dynamic response of large structures. This algorithm is exceptionally efficient for parallel processing because little communication is needed during the entire time step, and few extraneous calculations must be performed to compensate for this decoupling of the problem. The distributed data structure has been employed for parallel implementation of the group implicit algorithm. It involves partitioning the mesh into as many numbers of submeshes as the number of processors configured in the system for the problem. The distributed data structure employed for implementation of the group implicit time-marching scheme is same as the one employed for parallel domain decomposition method discussed earlier in this paper. The parallel solution employing group implicit technique can be described using this data structure. In general the entities which are built from element contributions, including mass matrix, the stiffness matrix, the global internal nodal forces, applied loading, contain contributions only from their resident domain. For example, the total mass of a particular interface node can be obtained only by summing up mass contributions of all submeshes sharing that particular node. The parallel algorithm for the total nonlinear transient dynamic analysis procedure employing the group implicit technique for a typical submesh is presented in figure 4.

7. Message passing interface (MPI)

Each processor in a distributed memory parallel computer has local memory and local address space. Since the only way to exchange data between these processors is to use explicit message passing, any nontrivial parallel system requires communication among processors. The message passing model assumes a set of processes that have only local memory but are able to communicate with other processes by sending and receiving messages. Data transfer for the local memory of one processor to the local memory of another requires operations to be performed by both processors. The message passing model fits well on separate processors connected by a communication network. Thus it matches the hardware of most of today's parallel super computers as well as workstation networks that are beginning to compete with them. This model has been found to be a useful and complete, to express parallel algorithms.

MPI (message passing interface) (Gropp *et al* 1994) was the first effort to produce a message passing interface standard across the whole parallel processing community. Sixty people representing forty different organisations, users, vendors of parallel systems from both the US and Europe collectively formed the "MPI Forum" in May 1994 and updated it in June 1995. The discussion was open to the whole community and was led by a working group with in-depth experience of the use and design of message passing systems (including developers of popular message passing libraries like PVM, P4, PARMACS etc.). The two-year process of proposals, meetings and review resulted in a document specifying a standard message passing interface (MPI). This standard, needed for portability and maintainability, provides hardware vendors with a well-defined set of routines to implement efficiently. MPI is a large specification consisting of over 100 functions and offers many basic and advanced features including point-to-point communications, collective communications, process groups, communications domains, and process. MPI's prime goals are to provide standardisation, source code compatibility, high performance, a great deal of functionality and to allow efficient implementation across a range of architectures. The rigor of the standard procedure has helped MPI reach prime place amongst message passing systems, offering real portability between HPC platforms. Many vendors of HPC (High Performance Computing) systems now offer native MPI support on their machines, and many generic versions are available free

```
1. Input and set up basic analysis data, set up time step counter $n = 1$
2. In each processor
    (i) Set iteration counter $i = 0$
    (ii) Predict the displacement, velocity and acceleration vectors
            $d_n^p = d_n = d_{n-1} + \Delta t v_{n-1} + 0{\cdot}50 \Delta t^2 (1 - 2\beta) a_{n-1}$
            $v_n^p = v_n = v_{n-1} + \Delta t (1 - \gamma) a_{n-1}$
    (iii) Compute global current tangent stiffness matrix $K_T$ and nodal force vector $f_n$
    (iv) Compute the internal force vector, $p({}^i d_n, {}^i v_n)$
    (v) Compute the effective residual forces ${}^i\varphi = f_n - \mathbf{M}^i \mathrm{a}_n - p({}^i d_n, {}^i \mathrm{v}_n)$
    (vi) Compute effective global stiffness matrix
            $\mathbf{K}^* = \mathbf{M}/(\Delta t^2 \beta) + \gamma \mathbf{C}_T/(\Delta t \beta) + \mathbf{K}_T({}^i d_n)$
    (vii) Factorise and solve $\mathbf{K}^* \, {}^i\Delta d = {}^i\varphi$ using profile solver
    (viii) Correct displacements and acceleration vectors
            ${}^{(i+1)}d_n = {}^i d_n + {}^i \Delta d_n$
            ${}^{(i+1)}a_n = [{}^{(i+1)}d_n - {}^i d_n]/(\Delta t^2 \beta)$
    (ix) Compute weighted displacement vector $M^{*(i+1)}d_n$, for all interface nodes,where $M$ is lumped mass
3. Communication
    (i) Send relevant parts of weighted displacement vector and interface mass to adjacent processors
    (ii) Receive relevant parts of weighted displacement vector and interface mass from adjacent processors
    (iii) Assemble the interface nodal weighted displacements and compute average incremental displacements
            ${}^{(i+1)}d_n = M^{-1}\{\Sigma \, {}^{i+1}M^s d_n^s\}$
          where $M$ is the assembled global mass of interface nodes
4. Correct the velocity and acceleration vectors of the submesh
            ${}^{(i+1)}a_n = [{}^{(i+1)}d_n - {}^i d_n]/(\Delta t^2 \beta)$
            ${}^{(i+1)}v_n = v_n + (\Delta t \gamma)^{(i+1)} a_n$
5. Communication
    (i) Check for convergence, if not converged set $i = i + 1$ go to step 2(iii)
    (ii) Else, if converged
6. In each processor
   Set
            $d_n = {}^{(i+1)} d_n$,
            $v_n = {}^{(i+1)} v_n$,
            $a_n = {}^{(i+1)} a_n$
   next time step: $n = n + 1$, and go to next time step
```

Figure 4. Parallel algorithm for nonlinear dynamic analysis group implicit technique.

of charge. For example MPICH implementation of MPI from Argonne National Laboratory (http:// www-unix.mcs.anl.gov/mpi/mpich/), is one of the most popular implementation and the source code is publicly available on the web for variety of hardware platforms.

The 1994 standard (version 1·1) defined only the interface for Fortran 77 and C, said nothing regarding process management, and did not allow dynamic process creation. Apart from that, MPI included many of the features of the parallel virtual machine (PVM), another popular message passing library. In addition, MPI added the concept of a ‘virtual topology’ from that of the underlying hardware. The recently published update to the standard (version 2·0, 1997) has C++ and Fortran 90 bindings, additional functions for dynamic processes, one-sided communication (useful for shared memory architectures), and parallel I/O.

8. Finite Element code for parallel nonlinear dynamic analysis

A parallel finite element code called SPANDAN (Software for PArallel Nonlinear Dynamic Analysis) has been developed by integrating all the three parallel algorithms for time integration discussed in this paper. SPANDAN is developed using MPICH implementation of MPI software development environment. SPANDAN is capable of solving problems with material nonlinearities. Material nonlinearity due to elastic–plastic material response is considered and anisotropy effects are included in the yielding behaviour by employing Huber Mises Yield criterion. Associated flow rule is employed to define incremental stress strain relations.

At present, SPANDAN’s element library consists of only Mindlin plate and shell elements. Even though, SPANDAN permits modelling structures with serendipity, Lagrangian and heterosis elements, all the numerical studies conducted in this paper uses numerically accurate heterosis element. Detailed element and nonlinear formulations are reported elsewhere (Hughes & Cohen 1978; Hughes & Liu 1981) . SPANDAN permits use of both consistent and lumped mass matrices (Hinton *et al* 1976).

As already mentioned in the earlier sections, the proposed domain decomposition algorithms are highly sensitive to the effectiveness of mesh partitioning. Since manual partitioning of finite element meshes of complex geometries is extremely difficult, it is usually accomplished using automatic mesh-partitioning algorithms. The basic requirement of any good mesh-partitioning algorithm is to maintain approximately equal computational load in each submesh and at the same time minimise the interface nodes. SPANDAN uses a software called PSTRAIN [Parallel STRuctural Analysis Interface] (Rama Mohan Rao 2001) for this purpose. PSTRAIN consists of a suite of mesh-partitioning algorithms ranging from simple coordinate bisection to highly sophisticated algorithms like multilevel spectral algorithms and multilevel graph partitioning algorithms. Genetic algorithm based mesh-partitioning techniques (Rama Mohan Rao *et al* 2004) developed by us are also included in PSTRAIN. In the present work, the GA-based mesh-partitioning algorithm has been employed for partitioning finite element meshes.

PSTRAIN partitions the finite element mesh into the desired number of submeshes and renumbers each submesh separately, identifies the interface nodes in each submesh and generates all the desired data required for each submesh depending upon the type of algorithm being employed for nonlinear dynamic analysis. For domain decomposition algorithms, each of the submeshes are renumbered in such a way that the internal nodes are numbered first, then all the interface nodes in order to facilitate the static condensation for generating the interface stiffness coefficient matrix. In the case of group implicit algorithms, the submeshes are renumbered in order to minimise the profile size using a profile minimization algorithm

(Sloan 1986). PSTRAIN maintains the mapping information of nodes/elements between the original finite element mesh and the renumbered submeshes for presentation of results.

In order to create necessary data structures in SPANDAN for parallel assembly of interface stiffness coefficient matrices and subsequent parallel solution of interface matrices for domain decomposition based algorithms, the following three new arrays have been introduced.
INFNOD *(numinf)*: List of interface nodes (*numinf*: number of interface nodes in the respective submesh)
GINF (*totinf*): Associated list of global interface node numbers which is required for interface assembly and subsequent solution (*Totinf*: Total number of interface nodes)
MARK (*numinf*): To distinguish between primary and secondary interface nodes. This information is essential for parallel implementation of the computations related to convergence check.
The same data structures have been employed for treating interfaces in the case of the group implicit algorithm by mass averaging.

A schematic flow chart of SPANDAN for one typical time step is shown in figure 5. Details related to the domain decomposition algorithm employing the predictor–corrector scheme have not been shown in the flowchart to maintain clarity. However, the flow of computations in a predictor–corrector based domain decomposition algorithm closely resembles the conventional domain decomposition algorithm shown in figure 5.

9. Features of computer hardware architecture employed for evaluation

The parallel code SPANDAN was initially developed on a Unix workstation (Sun ultra 300) under a simulated MPI environment. It was later ported on to Param-10000 at NPSF (National Param Supercomputing Facility), Pune and IBM SP-2 at the Supercomputing Education and Research Centre (SERC), IISc, Bangalore. It is appropriate to mention here that no change has been made in the original code while running either on Param-10000 or IBM SP-2. This demonstrates the portability of the parallel code developed under the MPI software development environment.

Param open frame scalable parallel computer is a distributed memory machine based on the heterogeneous open frame architecture. The open frame architecture unifies cluster computing with Massive Parallel Processing (MPP). Param-10000 is built with a cluster of 40 workstations. Each workstation in the cluster has 4 Ultra SPARC processors running at 300 MHz and 512 MB of Random Access Memory (RAM). It supports networks such as MYRINET, Fast ETHERNET, and PARAMNET. The architecture of Param-10000 permits the parallel processing system to be viewed as an assembly of independent workstations, a cluster of workstations or as an MPP system connected through a scalable high-bandwidth network or any combination of these. The operating system of Param-10000 is Solaris 2·6. The message passing libraries of Param-10000 consist of MPICH implementation of MPI and PVM3.

The IBM-SP2 is a distributed memory multicomputer (if a MIMD parallel computer is made up of a number of processors and each processor has access to its own memory, then the computer is referred to as a multicomputer) based on three types of IBM RISC System (RS)/6000 multiprocessors. The IBM-SP2 at SERC, IISc, Bangalore is configured with 32 processors. Of these 32 processors, 8 are IBM RS6000/590 processors running at 67 MHz with 512MB of RAM and 8 are IBM RS6000/591 processors running at 77 MHz with 256 MB of RAM. The rest of the processors (16 numbers) are IBM RS 6000/595 processors running at 135 MHz, each with 256 MB RAM. All 32 processors are connected internally through RS/232 and Ethernet. Processors 1–24 form an FDDI ring, which is connected to the Giga

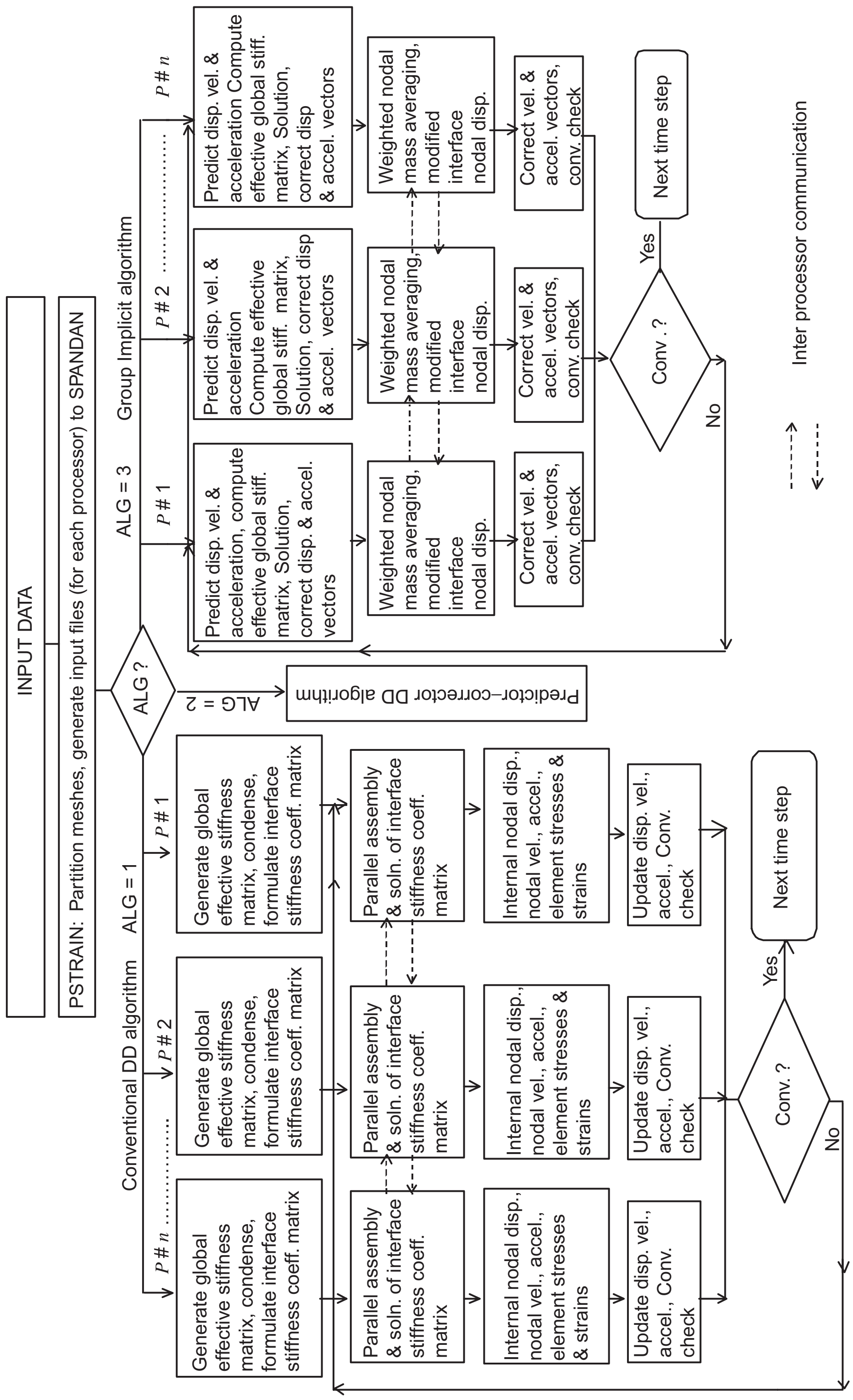

INPUT DATA
PSTRAIN: Partition meshes, generate input files (for each processor) to SPANDAN
ALG ?
ALG = 1
Conventional DD algorithm
P # 1
P # 2
P # n
Generate global effective stiffness matrix, condense, formulate interface stiffness coeff. matrix
Parallel assembly & soln. of interface stiffness coeff. matrix
Internal nodal disp., nodal vel., accel., element stresses & strains
Update disp. vel., accel., Conv. check
Conv. ?
Yes
No
Next time step
ALG = 2
Predictor–corrector DD algorithm
ALG = 3
Group Implicit algorithm
Predict disp. vel. & acceleration, compute effective global stiff. matrix, Solution, correct disp. & accel. vectors
Weighted nodal mass averaging, modified interface nodal disp.
Correct vel. & accel. vectors, conv. check
Conv . ?
Inter processor communication

Figure 5. Schematic flow diagram of SPANDAN.

Switch. The rest of the processors are connected through Ethernet with appropriate FDDI to Ethernet routers. Login and file serving are done through FDDI. Apart from all this, all 32 processors are connected internally via a High Performance Switch (HPS), which provides much larger communication bandwidth in dedicated mode.

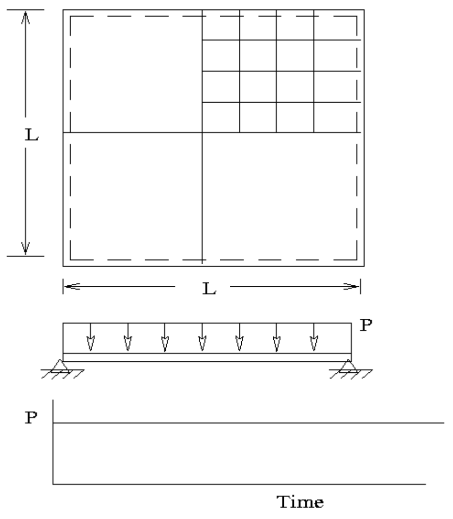
10. Numerical studies

Two different sets of numerical experiments have been conducted and reported in this paper. The first set of numerical studies is intended to test the accuracy of the parallel algorithms built into SPANDAN. For this purpose, numerical examples of smaller size are considered. The second set of numerical studies is conducted to test the performance of the developed parallel nonlinear dynamic analysis code on parallel processing machines using the three parallel implicit time integration algorithms discussed earlier in this paper. For this purpose, larger finite element models are employed, which can exploit the strengths and weaknesses of the parallel algorithms.

10.1 *Validation*

The parallel code is expected to give results identical to those of its sequential counterpart. In order to verify this, two standard numerical examples available in the literature are considered.

A simply supported plate shown in figure 6 is considered as a first numerical example to verify the nonlinear response of SPANDAN with the proposed parallel Newmark implicit time integration algorithm. The time step ΔT is taken to be $0{\cdot}223 \times 10^{-4}$ s, which is 1/48 of the fundamental period of the plate. Four processors have been employed for the solution. The dynamic elastic–plastic response of a simply supported plate for both isotropic and anisotropic (yield in different directions) material properties is shown in figure 7, which has been compared with the solutions of Huang & Hinton (1985) and Owen & Li (1988) reported in the literature. The results obtained using the proposed parallel implicit algorithm are found to be in good agreement with the results reported in the literature.



Length	$L = 254$ mm
Thickness	$t = 12.7$ mm
Young's modulus	$E = 689.472 \times 10^2$ MPa
Mass density	$\rho = 27.65 \times 10^{-10}$ N-s^2/mm^4
Poisson's ratio	$\nu = 0.30$
Yield stress	$\sigma = 206.84$ MPa
Plastic modulus	$G_P = 0.0$

Uniform load, $P = 2.0684$ MPa
Shear modulus = $265.17 \text{X} 10^2$ MPa
Shear stress $\tau_{012} = 139.27$ MPa
Anisotropic properties
Yield stress $\sigma_{02} = 310.26$ MPa
Yield stress $\sigma_{45} = 227.53$ MPa

Figure 6. Simply supported square plate: Problem description and finite element mesh.

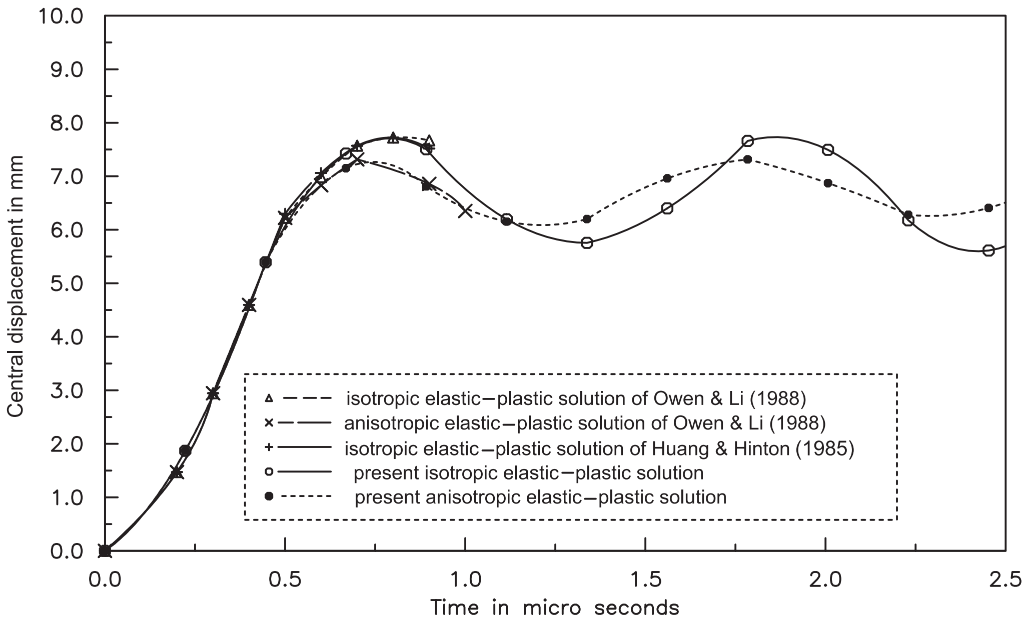


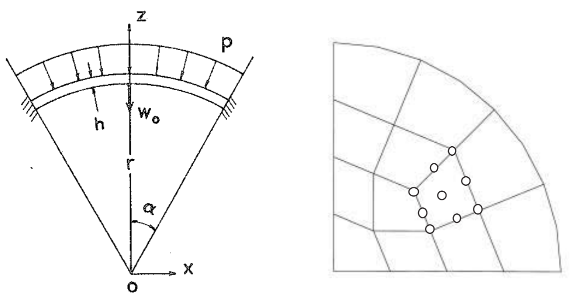
Figure 7. Elastic–plastic transient dynamic response of simply supported plate employing SPANDAN with parallel time integration algorithms.

A deep moderately thick spherical shell cap with central angle of 120° is considered as a second numerical example. Cap dimensions and material properties are given in the figure 8. A symmetric quadrant of the shell is considered and is described by a mesh shown in figure 8. Both isotropic and anisotropic (yield in different directions) cases are considered. The time step, Δt, is taken as $0{\cdot}10 \times 10^{-4}$ s. Only two processors are employed for the solution. Figure 9 shows a comparison of elastic, and elastic–plastic responses with isotropic and anisotropic material properties. The nonlinear dynamic response obtained using the parallel implicit algorithms of SPANDAN are compared with the results reported by Nagarajan & Popov (1973) and found to be in good agreement.

10.2 *Performance evaluation of parallel algorithms*

To evaluate the performance of the three parallel algorithms, larger size problems are employed. The problem sizes are increased either by changing the dimension of the plate or by refining the finite element mesh.

The principal indicator of the suitability of an algorithm to parallel processing and the scalability of the algorithm is the speedup factor. Speedup is defined as the ratio of the execution time on a conventional von Neumann machine by the fastest sequential algorithm to the execution time on a parallel processor. Since the definition of speedup dictates that parallel algorithms need to be compared with the fastest sequential algorithm (on a sequential machine with a same processor), execution timings are obtained only with full in core solutions. In view of this, even moderate size problems could not be solved with full in-core solvers on the sequential machine available with us. The other alternative with out-of-core solutions is highly I/O (input/output) bound. Hence it considerably slows down the sequential analysis and thereby makes the comparisons unfair. However, sequential code with full in-core oper-



Radius $R = 508.00$ mm	Thickness $t = 76.20$ mm
Angle $\alpha = 60^{\circ}$	Mass density $\rho = 78.34 \times 10^{-10}$ N-s^2/mm^4
Uniform load $P = 4.137$ MPa	Poison's ratio n = 0.30
Isotropic properties	*Anisotropic properties*
Young's modulus:	Young's modulus
$E_1 = E_2 = 206.84 \times 10^3$ MPa	$E_1 = E_2 = 206.84 \times 10^3$ MPa
Uniaxial yield stress in MPa	Uniaxial yield stress in MPa
$\sigma_{01} = \sigma_{02} = \sigma_{45} = 206.84$	$\sigma_{01} = \sigma_{45} = 206.84 \ = \sigma_{02} = 6894.72$
Plastic modulus $E_P = 1061.79 \times 10^2$ MPa	Plastic modulus $E_P = 1061.79 \times 10^2$ MPa

Note: σ_{01} and σ_{01} indicate uniaxial yield stress in 1, 2 principal axes and σ_{45} indicates yield stress at 45° to the 1-axis.

Figure 8. Problem description for deep moderately thick spherical shell cap. **(a)** Geometric detail of cap; **(b)** finite element mesh (nine noded elements).

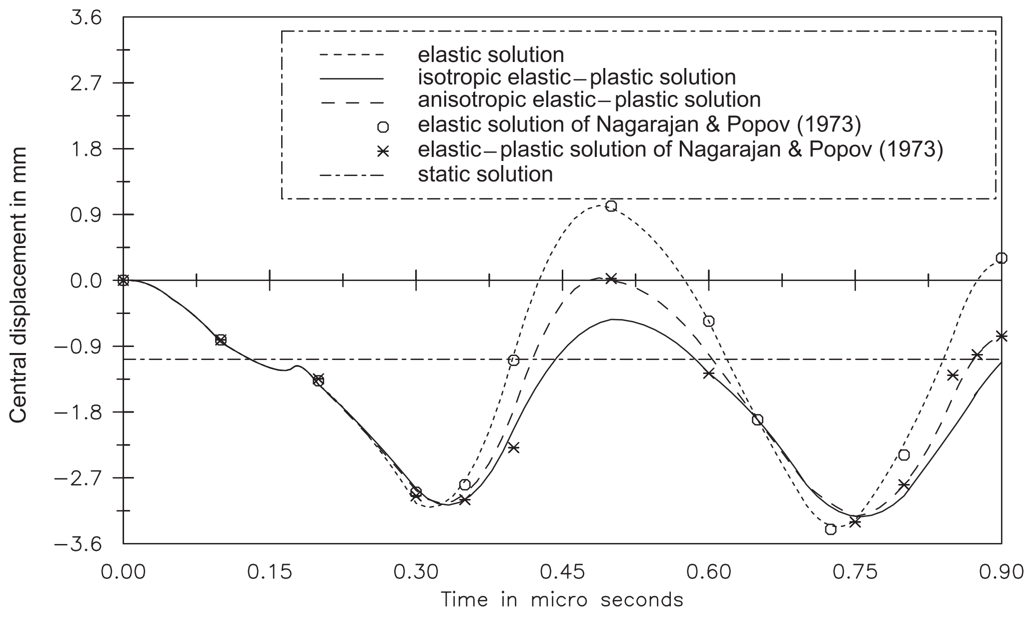


Figure 9. Elastic–plastic transient dynamic response of sperical shell cap employing SPANDAN with parallel time integration algorithms.

ations cannot solve problems of even moderate size without making use of virtual memory. Moreover, the page faults in the multi user sequential machine substantially degrade the performance of these problems. The timings of the parallel algorithm on parallel machines, when compared with the timings of the corresponding sequential algorithm on sequential machine may apparently exhibit superlinear speedups. Obtaining superlinear speedup is impractical in most of the situations.

In view of the reasons stated above, in this paper, the speedup of the parallel algorithms is evaluated only for smaller problems. For larger problems, the CPU timings have been shown to demonstrate the effectiveness of the algorithms.

For all the numerical examples considered for evaluation, the damping has been neglected and emphasis is given to maintain higher degree of accuracy with stringent convergence tolerance levels.

While implementing these parallel algorithms on Param-10000 at Pune and IBM-SP2 at IISc, Bangalore, no effort has been made to use advanced optimization options during compilation, which are likely to moderately improve the performance of the algorithms discussed here. On Param-10000, the code has been compiled using Sun Solaris F77 compiler with '$-O5$-*fast*' options. On IBM SP-2, the Fortran compiler (mpxlf) has been used for compiling the SPANDAN code with optimisation options as '$-O3$-*qarch*=*pwr*2-*qtune* = *pwr*2'.

Figure 10 shows the speedup of all three parallel algorithms while solving a plate problem of size 5445 degrees of freedom on four and eight processor configurations on Param-10000. The results have been compared with the theoretical speedup. Similarly, figure 11 shows the speedup performance of parallel algorithms for problem size of 12005 degrees of freedom. At least two observations can be made from the results given in figures 10 and 11.

The first observation is regarding the performance of the various algorithms. It can be observed that the Group Implicit algorithm is fastest among the three algorithms considered for evaluation in this section. As already discussed in the previous chapter, the communication requirement of a parallel GI algorithm is much lesser than the other two parallel implicit time stepping algorithms being evaluated in this section. Therefore, the performance of the GI algorithm is likely to be superior to that of the other two parallel implicit algorithms considered for evaluation in this paper. The numerical results shown in figures 10

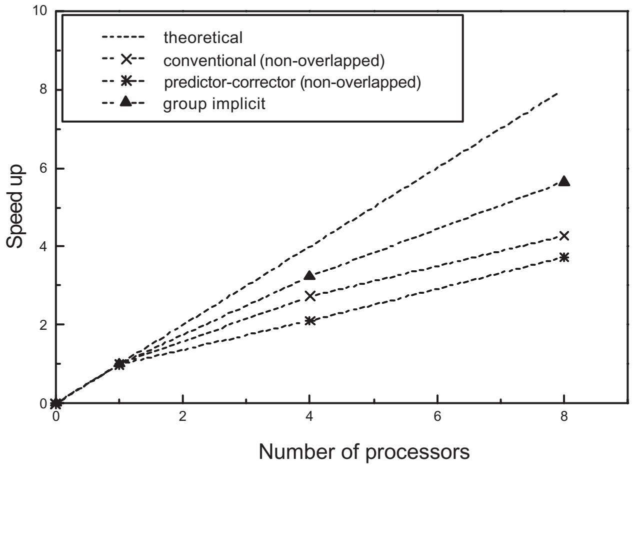


Figure 10. Performance of SPANDAN for a problem size of 5445 degrees of freedom on Param-10000.

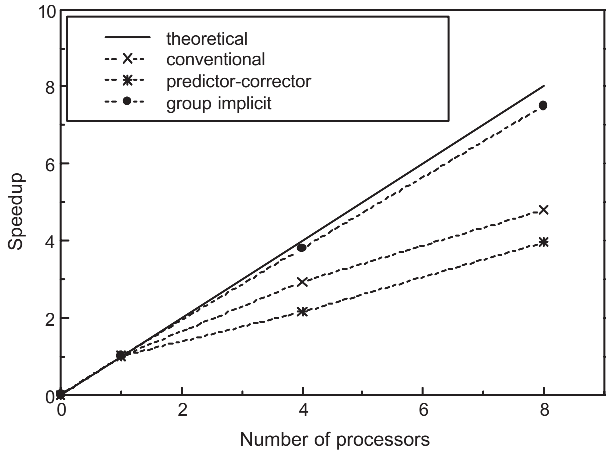


Figure 11. Performance of SPANDAN for a problem size of 12005 degrees of freedom on Param-10000.

and 11 in fact reflect the same. However, the GI algorithm has the drawback that it suffers in accuracy if the time step length is larger than the Courant condition given in (20). From the results shown in figures 10 and 11, it can be observed that the parallel algorithm based on the predictor–corrector form of the Newmark method is by far the slowest one among all the parallel algorithms considered for evaluation in this section. This is mainly due to slow convergence of the predictor–corrector iterative scheme and also the communication overheads of the profile solver. Finally, one can also observe that there is improvement in the performance of all the parallel algorithms with increase in problem size.

In order to test and evaluate these algorithms with larger problems and larger processor configurations, three test meshes with 21125, 32805, 83205 degrees of freedom are solved employing all the three parallel algorithms slated for evaluation in this section. The wall clock timings for all these test problems on four, eight, sixteen and thirty-two processors are presented in figures 12, 13 and 14 respectively. Due to paucity of computer time, the simulations have been carried out for 200 time steps on Param-10000. From the results presented in figures 10 and 11, it has been observed that the overall computational performance of all the parallel algorithms improves with increase in the problem size. Similarly, it can

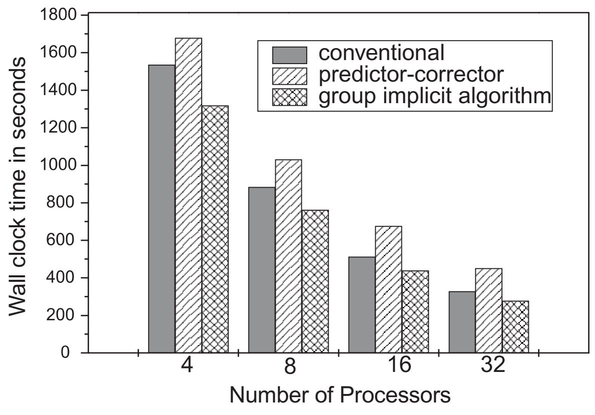


Figure 12. Wall clock timings of three parallel time-integration algorithms for a problem size of 21125 degrees of freedom on Param-10000.

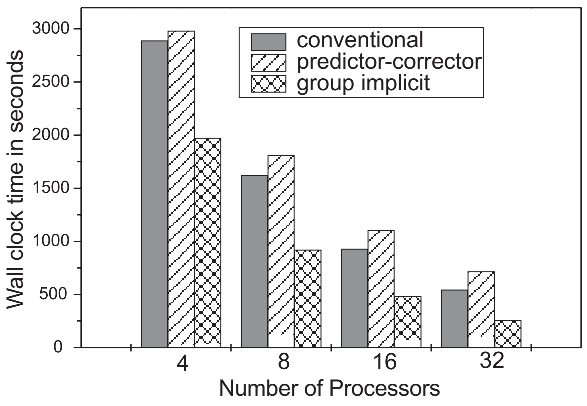


Figure 13. Wall clock timings of three parallel time-integration algorithms for a problem size of 32805 degrees of freedom on Param-10000.

be observed from figures 12–14 that the behaviour of the parallel algorithms on the varied number of processors is consistent.

On IBM-SP2, only limited studies have been carried out due to paucity of computing time. The wall clock timings of two typical test meshes on Param-10000 and IBM SP-2 is shown in tables 1 and 2 respectively, for all the three parallel algorithms. It can be observed that the results are consistent even though IBM SP-2 is faster than Param-10000.

In order to evaluate the interprocessor communication overheads, the group implicit algorithm has been solved on 2, 4 and 8 processors of IBM SP-2 and the computation as well as communication timings for plate problem of size 32805 degrees of freedom is presented in figure 15. The wall clock timings recorded are for 500 time steps. A close look at figure 15 indicates that the interprocessor communication overheads are only a small fraction of the overall computation of group implicit algorithm.

In order to compare the interprocessor communication overheads in all the three parallel algorithms, wall clock time spent on interprocessor communications for a typical test mesh of 32805 degrees of freedom is shown in figure 16. A close look at this figure indicates that the communication overheads are minimal in the GI algorithm and maximum in the predictor–corrector Newmark algorithm. This is clearly reflected in the overall performance of each of the algorithms presented and discussed earlier.

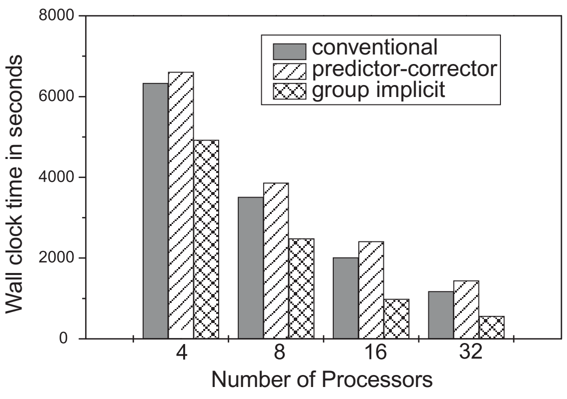


Figure 14. Wall clock timings of three parallel time-integration algorithms for a problem size of 83205 degrees of freedom on Param-10000.

Table 1. Wall clock timings in seconds for a 32 × 32 FE mesh on Param-10000 and IBM SP-2 (200 time steps).

	Four processors		Eight processors	
Parallel algorithm	Param-10000	IBM SP-2	Param-10000	IBM SP-2
Conventional Newmark algorithm	1536	1311	884	712
Predictor–corrector form of Newmark algorithm	1679	1309	1031	814
Group implicit	1318	1066	703	558

Table 2. Wall clock timings in seconds for a 64 × 64 FE mesh on Param-10000 and IBM SP-2 (200 time steps).

	Four processors		Eight processors	
Parallel algorithm	Param-10000	IBM SP-2	Param-10000	IBM SP-2
Conventional Newmark algorithm	6231	5797	3515	3158
Predictor–corrector form of Newmark algorithm	6611	5696	3864	3512
Group implicit	4928	4484	2484	2194

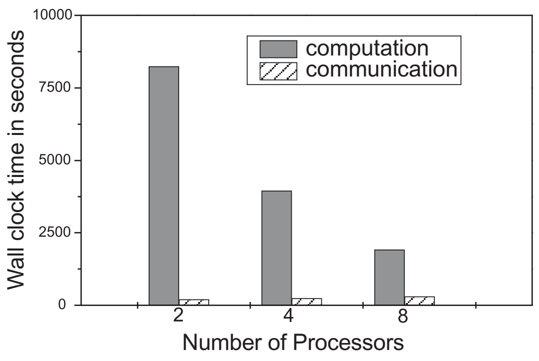


Figure 15. Communication overheads in group implicit time-integration algorithm for a problem size of 32805 degrees of freedom on IBM SP-2.

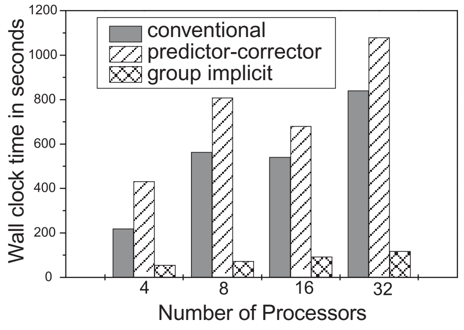


Figure 16. Comparison of communication timings of three parallel time integration algorithms for a problem size of 32805 degrees of freedom on Param-10000.

11. Conclusions

In this paper, parallel processing techniques for computing nonlinear dynamic response employing finite element method is presented. The implicit time integration algorithm is the most time consuming segment of the algorithm. Hence three different parallel algorithms for implicit time integration employing Newmark constant average acceleration technique are presented in this paper and their relative efficiencies are evaluated. The parallel code SPANDAN has been developed integrating all the three parallel time integration algorithms discussed in this paper. Portable MPI has been employed in the development of the parallel code. The portability of SPANDAN is demonstrated by porting on two distinct parallel processing machines without making any changes in the code.

The performance of the parallel code SPANDAN is evaluated by solving number of test meshes of varying size and employing variable number of processors. Numerical experiments indicate that the parallel Group Implicit algorithm is by far the fastest of the three algorithms discussed in this paper. However, the time step sizes in parallel Group Implicit algorithm need to be restricted to obtain results of desired accuracy. This is basically due to semi implicit nature (improper transfer of information to the entire domain during time integration) of the Group Implicit technique. This often becomes too restrictive when the submesh size is smaller.

The major source of error in Group Implicit algorithm is in employing the ‘mass averaging rule’ in order to maintain compatibility at interface nodes between submeshes. With this, the equilibrium is no longer satisfied at both interface and the interior nodes that are adjacent to the interfaces. In other words, the error crept into the GI algorithm because the interface reactive force is not accounted for. These reactive forces are assumed to be zero during solution phase. However during averaging phase an unknown amount of interface reactive forces are generated disrupting the equilibrium conditions. The accuracy of the GI algorithm can be improved by incorporating the effect of these reactive forces. However this correction to the basic GI algorithm offsets the advantages interms of efficient interprocessor communications, as it invariably require additional implicit computations. In view of this, it may be better to use GI algorithm in the present form, in order to take advantage of its lesser communication overheads while solving large size problems on coarse grain environment.

While the predictor–corrector form of Newmark algorithm is the slowest of the three parallel algorithms, the parallel algorithm developed employing conventional form of Newmark time integration algorithm permits larger time steps like its sequential counterpart and faster than the predictor–corrector form. Hence conventional form parallel algorithm is recommended for fine grain parallelism where the submesh size may be smaller, while the Group Implicit algorithm performs better in coarse grain environment for large size problems. The performance of the three algorithms found to be consistent on the two hardware platforms chosen for evaluation.

References

Ali N, Parsons I D 2000 An MPI implementation of Newmark’s method. *Comput. Aided Civil Infrastruct. Eng.* 15: 189–195

Belytscho T, Hughes T J R 1976 Mesh partitions of explicit-implicit time integration. *Formulations and computational algorithms in finite element analysis* (eds) K Bathe, J T Oden, W Wunderlich (Cambridge, MA: MIT press) pp 673–690

Bennighof J K, Wu J Y 1991a Parallel transient algorithm with multistep substructure computation. *AIAA J.* 29: 984–991

Benninghof J K, Wu J Y 1991b An unconditionally stable parallel transient algorithm with multi-step independent subdomain computation. *Comput. Sys. Eng.* 2: 217–230

Chen X, Kumar K T, Sha D 1995 Virtual pulse time integral methodology: A new approach for computational dynamics Part 2: Theory for nonlinear structural dynamics. *Finite Elements Anal. Design* 20: 195–204

Chiang, Fulton R 1990 Structural dynamics methods for concurrent processing computers. *Comput. Struct.* 36: 1031–1037

Cook R D, Malkus D S, Plesha M E 1989 *Concepts and applications of finite element analysis* (New York: John Wiley and Sons)

Danielson K T, Namburu R R 1998 Nonlinear dynamic finite element analysis on parallel computers using Fortran 90 and MPI. *Adv. Eng. Software* 29: 179–186

Felippa C A, Park K C 1980. Staggered solution transient analysis procedures for coupled mechanical systems: Formulation. *Comput. Meth. Appl. Mech. Eng.* 24: 61–111

Garey M R, Johnson D S 1979 *Computers and intractability: A guide to the theory of NP-completeness* (New York: Freeman W.H)

Gropp W, Lusk E, Skjelum A 1994 *Using MPI portable parallel programming model with message passing interface* (Cambridge, MA: MIT Press)

Hajjar J F, Abel J F 1988 Parallel processing for transient nonlinear structural dynamics of three dimensional framed structures using domain decomposition. *Comput. Struct.* 30: 1237–1254

Hinton E, Rock T, Zienkiewicz O C 1976 A note on mass lumping and related processes in finite element method. *Earthquake Eng. Struct. Dyna.* 4: 246–249

Huang H C, Hinton E 1985 Elastic–plastic dynamic analysis of plate and shell structures using a new nine node element. In *Proceedings of the ASME Symposium on Material Nonlinearity in vibration problems*. (AMD) 71: 41–60

Hughes T J R, Cohen M 1978 The 'Hetorosis' finite element for plate bending. *Comput. Struct.* 9: 445–450

Hughes T J R, Liu W 1978 Implicit-explicit finite elements in transient analysis: Implementation and numerical examples. *J. Appl. Mech.* 45: 375–378

Hughes T J R, Liu W K 1981a Nonlinear finite element analysis of shells: part 1, Three-dimensional shells. *Comput. Meth. Appl. Mech. Eng.* 26: 331–362

Hughes T J R, Liu W K 1981b Nonlinear finite element analysis of shells: part II, Two-dimensional shells. *Comput. Meth. Appl. Mech. Eng.* 27: 167–181

Hughes T J R, Winget J, Levit I, Tejduyar 1983 New alternating directions procedures in finite element analysis based upon EBE approximate factorisations. *Recent developments in computer methods for nonlinear solid and structural mechanics* (eds) N Perrone, S N Atluri (New York: ASME) pp 75–109

MPI Forum 1994 MPI: A message-passing interface standard. *Int. J. Supercomput. Appl.* 8: 159–416

MPI Forum. 1997 MPI 2: Extensions to the message passing interface. Technical Report, University of Tennessee, Knoxville, TN

Nagarajan S, Popov E P 1973 Elastic–plastic dynamic analysis of axisymmetric solids. Report No. UC SESM 73-9, University of California, Berkeley

Newmark N M 1959 A method of computation for structural dynamics. *J. Eng. Mech. Div. ASCE* 85(EM3): 67–94

Ortiz M, Nour-Omid B 1988 Unconditionally stable concurrent procedures for transient finite element analysis. *Comput. Methods Appl. Mech. Eng.* 58: 151–174

Ortiz M, Nour-Omid B, Sotelino E D 1988 Accuracy of a class of concurrent algorithms for transient finite element analysis. *Int. J. Numer. Meth. Eng.* 26: 379–391

Ortiz M, Pinsky P M, Taylor R M 1983 Unconditionally stable element-by-element algorithms for dynamic problems. *Comput. Meth. Appl. Mech. Eng.* 36: 223–239

Owen D R J, Li Z H 1988 Elastic–plastic dynamic analysis of anisotropic laminated plates. *Comput. Meth. Appl. Mech. Eng.* 70: 349–365

Rama Mohan Rao A 2001 *Efficient parallel processing algorithms for nonlinear dynamic analysis*. Ph D thesis, Indian Institute of Science, Bangalore

Rama Mohan Rao A, Loganathan K, Raman N V 1993 Studies on two concurrent solvers for finite element analysis. *Adv. Eng. Software* 18: 161–166

Ram Mohan Rao A, Appa Rao T V S R, Dattaguru B 2004 Generating optimised parameters for parallel finite element computations employing float-encoded algorithms. *Int. J. Comput. Modeling Eng. Sci.* 53: (in press)

Sloan S W 1986 An algorithm for profile and wave front reduction of sparse matrices. *Int. J. Numer. Meth. Eng.* 23: 239–251

Sziveri J, Topping B H V 2000 Transient dynamic nonlinear analysis using MIMD computer architectures. *J. Comput. Civil Eng.* 14: 79–91

Trujillo D M 1977 An unconditionally stable explicit method for structural dynamics. *Int. J. Numer. Meth. Eng.* 11: 1579–1592